OPENFORGE RL: TRAIN HARNESS-NATIVE AGENTS IN ANY ENVIRONMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern AI agents rely on elaborate *inference harnesses* such as Claude Code, Codex, and OpenClaw to drive multi-turn reasoning, tool use, and access to external systems. While powerful, these complex harnesses also make agents hard to train end-to-end with open infrastructure, whose SFT/RL stacks cannot natively express stateful, multi-process harness inference. To address this, we present **OPENFORGE RL**, an open-source framework for training harness-based agents end-to-end in diverse environments. OPENFORGE RL achieves this with a lightweight proxy that serves the harness's model calls while recording them as training data for a standard RL codebase (e.g., veRL), and a Kubernetes orchestrator that runs each rollout in its own remote container, together enabling training on *any harness in any environment* at scale. By decoupling training and inference, OPENFORGE RL allows researchers to easily train, study, and improve agents directly in the real harnesses and environments they are deployed with. We validate our framework across diverse, complex harnesses and environments, spanning tool/claw-based agents and multimodal GUI browser- and computer-use agents. Using only hundreds to a few thousand tasks, OpenForge-Claw reaches 31.7 (pass^3) and 55.9 (pass@3) on ClawEval and 33.7 on Qwen-ClawBench. OpenForge-GUI reaches 37.7 on OSWorld-Verified, 63.0 on Online-Mind2Web, and 72.3 on WebVoyager. Both outperform open baselines of similar size on nearly all benchmarks, and in the GUI setting match or surpass models several times larger. Beyond benchmarks, we analyze how harness choice (e.g., ZeroClaw, OpenClaw, Codex) and RL shape agent behavior. We find that some harnesses are substantially harder to learn than others, and that RL improves *agentic reliability*, such as self-verification, tool coverage, and completing multi-step plans, though critical abilities such as error recovery remain weak.

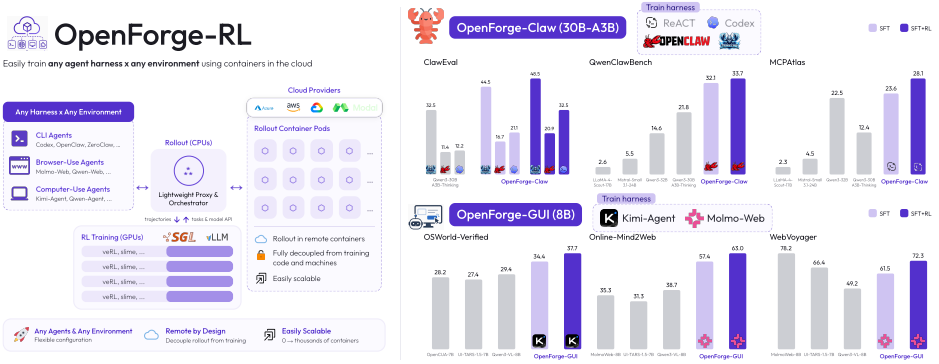


Figure 1: *Left*: OPENFORGE RL builds on Orchard Env (Peng et al., 2026) and connects *any harness* $\times$ *any environment* to standard RL codebases such as veRL, with no train–deploy mismatch. *Right*: OPENFORGE-trained models evaluated with six harnesses across six Claw and GUI environments.

1 INTRODUCTION

Modern AI agents are increasingly deployed in complex, open-ended environments, from software engineering (Jimenez et al., 2024; Yang et al., 2024; Merrill et al., 2026) and tool-use (Bandi et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2026; Patil et al., 2025) to controlling real web browsers and desktops (Zhou et al., 2024; Xue et al., 2025; Xie et al., 2024). To operate effectively in these settings, state-of-the-art agents are rarely a bare language or vision-language model; instead, they are wrapped in increasingly sophisticated *inference harnesses*: orchestration scaffolds such as Claude Code (Anthropic, 2025), Codex (OpenAI, 2025a), and OpenClaw (OpenClaw, 2025) that manage multi-turn interaction, tool use, and context while linking the model to external systems such as MCP servers (Anthropic, 2024), browsers, and GUIs. This harness layer is now central to agentic capability: recent progress on agentic benchmarks comes as much from harnesses as from stronger base models (Yang et al., 2024; OpenAI, 2026).

Yet improving these agents *end-to-end* remains out of reach for much of the open research community, for two reasons. First, a harness turns inference into a stateful, multi-process procedure, with nested tool calls, subagents, and long-horizon context, that open training stacks (Sheng et al., 2024; Zhu et al., 2025; Cao et al., 2025) cannot natively express. Open efforts often need to reimplement a simplified harness for training (Wei et al., 2025; Wang et al., 2026), creating a train–deploy mismatch. Second, running harnesses at scale requires dedicated, containerized environments that cannot be co-located on training nodes, yet most open RL frameworks assume rollouts run locally inside the trainer. Together, these gaps leave proprietary harness-based systems increasingly ahead of what the research community can train and study.

To close these gaps, we introduce **OPENFORGE RL**, an open framework for training harness-based agents end-to-end, built on two lightweight components (Figure 1 left and Figure 2). First, a lightweight proxy abstracts the harness's inference process and decouples it from training, so that *any* harness can run its own arbitrary inference. Paired with automatic trajectory reconstruction, the recorded prompt-response pairs become standard samples compatible with any RL codebase (e.g., veRL (Sheng et al., 2024)). Second, a Kubernetes orchestrator, following Peng et al. (2026), launches each rollout as a remote container on cloud providers such as Microsoft Azure, scaling elastically to many concurrent environments. Together, this reuses the rich harness ecosystem directly, spans environments from tool use to GUIs, and remains agnostic to the underlying RL algorithm.

We validate OPENFORGE RL across a broad spectrum of complex agentic settings, ranging from daily tool-use and claw-based agents to multimodal GUI browser- and computer-use agents. In the daily tool-use setting, **OpenForge-Claw** (30B-A3B MoE) trains on 3 popular harnesses (ZeroClaw, OpenClaw, and Codex) in addition to the standard ReACT loop, and reaches 31.7 (pass^3) and 55.9 (pass@3) on ClawEval (Ye et al., 2026), 33.7 on QwenClawBench (Qwen Team & Data Team, 2026), and 28.1 on MCPAtlas (Bandi et al., 2026). In the GUI setting, **OpenForge-GUI** (8B) trains on (a modified version of) Kimi-Agent (Xie et al., 2024) and Molmo-Web (Gupta et al., 2026), and attains 37.7 on OSWorld-Verified, 63.0 on Online-Mind2Web, and 72.3 on WebVoyager, outperforming open baselines of similar scale and matching or surpassing models several times larger. Crucially, because OPENFORGE RL trains agents in their real deployment harnesses, it also lets us study how harness choice and RL training shape agent behavior, a question prior open work could not easily ask (Section 5): some harnesses prove far harder to learn than others, and RL broadly improves *agentic reliability* though abilities such as error recovery remain weak.

In summary, our contributions are threefold:

- We introduce OPENFORGE RL, a scalable training infrastructure that flexibly pairs *any harness with any environment*, connecting real harness ecosystems and remote sandboxes in cloud-service providers to powerful RL codebases (e.g., veRL)
- We conduct a broad empirical study across tool-use/claw and GUI (browser and computer-use) agents, improving over open models of similar size on nearly all benchmarks and, in several cases, over models several times larger.
- Enabled by training in real deployment harnesses, we analyze how harness choice and RL shape agents: simpler, better-aligned harnesses are easier to learn; training generalizes across similar harnesses; and RL overall improves *agentic reliability* (e.g., self-verification, tool coverage, and task completion), though error recovery remains challenging to learn.

2 RELATED WORK

Agents with inference harnesses. With the advent of LLM-based coding agents, early work such as SWE-Agent (Yang et al., 2024) showed that a carefully designed agent-computer interface sub-

stantially improves task success (Jimenez et al., 2024; Xia et al., 2024; Wang et al., 2025a). Subsequent harnesses such as Claude Code (Anthropic, 2025) and Codex (OpenAI, 2025a) refined this recipe for software engineering, and open-source efforts such as OpenClaw (OpenClaw, 2025) extended it to everyday tasks as well as to GUI tasks (Hong et al., 2024; Qin et al., 2025; Xie et al., 2024; Agashe et al., 2024). Rather than developing harnesses, we study how to train LLM- and VLM-based agents end-to-end using these harnesses, aligning agent training with real-world usage so that models can learn in the same setting in which they are deployed.

Training harness-based agents. As agents take on increasingly complex environments, training them end-to-end has become a central goal, with reinforcement learning (RL) emerging as the primary tool. Several open-source RL frameworks, such as veRL (Sheng et al., 2024), Slime (Zhu et al., 2025), and more (Hu et al., 2024; Cao et al., 2025; Fu et al., 2026), support advanced algorithms (e.g., PPO and GRPO) and asynchronous distributed training. However, they assume relatively simple rollouts: single-turn generation, or multi-turn interaction with lightweight tool calls such as sandboxed code execution. These assumptions break for complex inference harnesses, which manage multi-turn interaction, tool use, and context internally, and whose rollouts require containerized environments with dedicated CPU and memory that cannot be co-located on the training nodes at scale. As a result, initial attempts training agents in these complex environments requires heavily modifying the training loop to fit specific task case-by-case (Bai et al., 2024; Jin et al., 2025; Qi et al., 2025; Yu et al., 2025b) — making the codebase hard to extend and maintain. Our work presents a first step towards designing a training flow that natively supports complex harnesses and environments at scale, while remaining agnostic to the underlying RL framework[1]

3 METHODS

3.1 NOTATION

Completing tasks in complex, long-horizon environments is commonly formulated as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$. At each step t of a multi-step task, an LLM-based agent π_θ receives a task instruction together with an observation[2] $s_t \sim \mathcal{S}$, generates an action $a_t \sim \pi(\cdot|s_t)$, and transitions to the next observation $s_{t+1} \sim \mathcal{S}$. Under naive inference (e.g., ReACT (Yao et al., 2023)), the agent is given the raw interaction history $\langle s_{t-H}, a_{t-H}, \ldots, s_t \rangle$ and reasons before emitting the next action a_t. This loop repeats until the task is completed or a maximum number of steps is reached, at which point a terminal reward $r_T \sim \mathcal{R}(a_T, s_T)$ indicates success or failure. However, for complex tasks such as coding and GUI control, prior work has shown that carefully designed tools and advanced control flows—subagents, skills, and planning modes—substantially improve agent performance (Wang et al., 2023; Wu et al., 2023; Agashe et al., 2024). In practice, the agent is therefore wrapped in a harness $\mathcal{H}(\pi)$ that supplies these tools and control flows internally, so that its effective context is *no longer* the raw interaction history. To abstract away this complexity, we denote (i) each prompt-response pair produced during harness inference as $(\mathcal{H}(s_t), a_t)$, and (ii) the full trajectory as an time-ordered collection $\tau = \langle (\mathcal{H}(s_0), a_0), (\mathcal{H}(s_1), a_1), \ldots, (\mathcal{H}(s_T), a_T) \rangle$.

3.2 OPENFORGE RL

While the emerging agent domains beyond software engineering are increasingly well benchmarked (Patil et al., 2025; Wu et al., 2025; Bandi et al., 2026), little open-source infrastructure supports training harness-based agents end-to-end in them. This is challenging for two reasons. First, existing RL frameworks assume the trainer has direct access to the model's prompts and full control over generation throughout a rollout; harnesses break this assumption, since their control flows, subagents, and context management are not exposed to the trainer. Second, harness rollouts need containerized environments with dedicated CPU and memory that cannot be co-located on the train-

[1] Concurrent work such as Polar (Xu et al., 2026) proposes a similar approach but focuses on software-engineering tasks (SWE-Bench-Verified). Our work covers a substantially broader setting, spanning text-based claw tasks and vision-based GUI tasks across six benchmarks, and also further analyzes how harness choice and RL shape agent behavior.

[2] Technically, any input to the agent from our environments is an observation (as in a POMDP); to simplify notation, we use s to denote input data received from the environment.

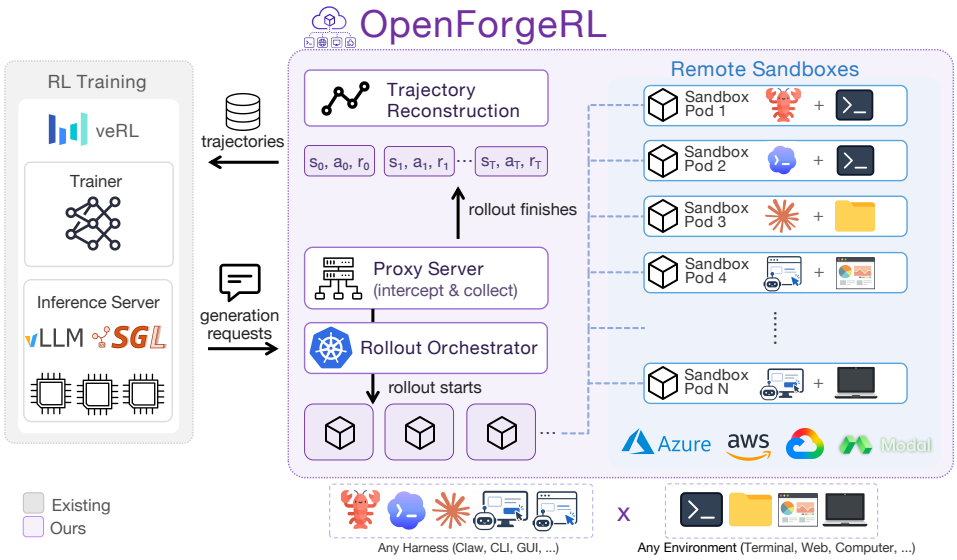


Figure 2: Overview of OpenForge RL. An orchestrator spawns remote sandboxes in which an LLM/VLM interacts with its environment through a harness. A proxy intercepts the harness's LLM calls, routes them to the RL framework's inference engines, and records the exchanged io-pairs as training trajectories. Supporting a new harness or environment only modifies the sandbox.

ing nodes *at scale*. More broadly, open models are rarely trained in the harnesses they are deployed with, widening the gap to closed-source frontier models.

We address both with OpenForge RL, a plug-and-play rollout interface connecting popular RL frameworks (e.g., veRL) to distributed harness rollouts (Figure 2). Given an inference server (e.g., vLLM Kwon et al. (2023)), OpenForge RL launches (1) a Kubernetes orchestrator that creates, manages, and deletes rollout container pods on cloud providers such as Microsoft Azure, and (2) a proxy that wraps the inference server and intercepts all generation requests issued by those containers. When a rollout finishes, the proxy collects the terminal reward r_T (typically task success) and the prompt-response pairs $(s^{\mathcal{H}}, a)$ from the container, and reconstructs training samples as:

$$\tau = (s_0^{\mathcal{H}}, a_0, r_0), (s_1^{\mathcal{H}}, a_1, r_1), \ldots, (s_T^{\mathcal{H}}, a_T, r_T); \quad r_t = \gamma^{T-t} \cdot r_T \tag{1}$$

typically with $\gamma = 1.0$. When optimizing with group-based algorithms such as GRPO, we follow Feng et al. (2025) and compute advantage by comparing the average sample rewards in *different* trajectories in the same group. While offloading rollouts to remote containers enables scaling to many concurrent environments, it raises three practical challenges, which we address below.

Rollout orchestration. To manage the lifecycle of many rollout containers, we build on Orchard (Peng et al., 2026) and implement a Kubernetes orchestrator that creates, deletes, and allocates resources for rollout containers on cloud providers such as Microsoft Azure. This allows us to easily manage and scale the number of concurrent rollouts elastically, without overloading training nodes.

Asynchronous rollout and timeouts. Because each rollout runs remotely and outside the trainer's control, a single unresponsive rollout will block the collection of an entire training batch and subsequently stall training. Capping the number of turns per rollout (Jin et al., 2025; Yu et al., 2025b), a common safeguard, is unreliable in this setting: some harnesses (e.g., Codex) do not expose a turn limit, and a "turn" is defined inconsistently across harnesses. We instead impose a (generous) wall-clock timeout on each rollout job. When a job exceeds its timeout, we terminate it and return an error signal to the proxy and the trajectory-reconstruction module, so training continues collecting from the remaining rollouts rather than blocking by the terminated one.

Error handling. Rollouts in these complex environments can fail for reasons unrelated to the policy model, such as network issues, harness crashes, or timeouts. Following DAPO (Yu et al.,

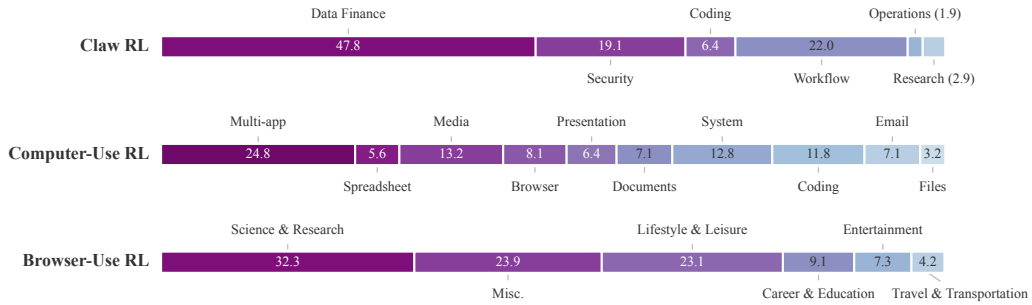


Figure 3: Distribution (%) of tasks used in the Claw, Computer-Use, and Browser-Use domains (top to bottom) for OPENFORGE RL training. SFT-task distributions are similar and shown in Figure A7.

2025a), we consider a simple strategy to discard all samples from a trajectory that ended in such an error, since a partial rollout can inject misleading training signal (e.g., a correct prefix that receives a negative reward). Designing better credit assignment for these partial rollouts is a promising direction in this setting, which we leave to future work.

3.3 DATA SYNTHESIS

Unlike coding (Badertdinov et al., 2026), the domains we target offer few RL-ready training tasks (*i.e.* executable environment and a verifier). We therefore build a simple agentic pipeline that synthesizes tasks together with the environments they run in: it proposes instructions grounded in realistic scenarios, builds a containerized environment and a verifier script for each, and then tests and refines the task until it passes an end-to-end rollout. The resulting tasks support both SFT and RL; we report their statistics in Table A2 and their category distribution in Figure 3. Since data synthesis is not a core contribution of this work, we defer the details to Section C.

4 EXPERIMENTS

In this section, we use OPENFORGE RL to train agents across a range of harnesses and environments. We then evaluate the trained agents across six popular benchmarks, spanning both text-based tool-use (claw) and multimodal GUI (browser and computer use) domains. In the next section, we then study *how* OPENFORGE RL training reshapes agent behavior, examining tool-use patterns, generalization to unseen harnesses, and the behavioral changes that RL introduces on top of SFT.

4.1 CLAW AGENTS

We first use OPENFORGE RL to train agents across *diverse harnesses*. Here we hold the environment type fixed to text-based tasks, and train and evaluate LLM-based agents on everyday tasks that require diverse tool use, such as reading email, searching a knowledge base, or updating a helpdesk ticket, inside harnesses such as ZeroClaw, OpenClaw, and Codex.

SFT and RL Data We use our pipeline in Section 3.3 to automatically generate a large pool of tasks and executable environments for both SFT and RL. To seed the proposal stage, we give the pipeline an assets and reference pool drawn from (1) ClawHub access[3] and (2) tasks from ZClaw-Bench (AI, 2026), which together supply examples that reflect people's daily usage of claw-based agents. To study the effect of training on diverse harnesses, we pair each task with one of four harnesses for its rollouts: ReACT (solving the task via a simple loop), ZeroClaw, OpenClaw, and Codex. For more details, please see Section C.3.

Training Details We use Qwen3-30B-A3B-Thinking (Yang et al., 2025) as the backbone model for training. For SFT, we distill trajectories from a stronger teacher model (MiniMax-M2.5, MiniMax (2026)): we sample $N = 3$ rollouts per task and keep only the successful ones for training. For

[3]The agent can use command lines to lookup popular skills/use cases from `https://clawhub.ai/`

Table 1: Claw-agent performance on Claw-Eval, QwenClawBench, and MCPAtlas. We use the general domain (0408) for Claw-Eval. *marks numbers from the Claw-Eval leaderboard. †marks numbers from the QwenClawBench leaderboard. MCPAtlas♠ results are pass@1 over the 89-task without credential-server configuration.

System	ClawEval		QwenClawBench	MCPAtlas♠
	$pass^3$	$pass@3$	$pass@1$	$pass@1$
SOTA Large Language Models				
Claude Opus 4.6*†	70.8	80.8	59.5	76.4
GPT 5.4*†	60.2	75.8	56.7	68.5
Gemini 3.1 Pro*	55.9	80.8	–	–
Qwen3.5 397A17B*	57.8	70.8	54.2	66.3
GLM 5 Turbo*	52.8	73.3	–	67.4
MiniMax M2.7*	49.7	72.0	50.5	50.6
MiniMax M2.5	47.2	65.2	51.2	57.3
Kimi K2.5*†	36.6	67.1	51.9	52.8
Similar-size baselines and our model (30B-A3B; ~3B active)				
LLaMA-4-Scout-17B-16E-Instruct	0.6	16.8	2.6	2.3
Mistral-Small-3.1-24B-Instruct	3.1	19.3	5.5	4.5
Qwen3-32B	6.8	31.7	14.6	22.5
Qwen3-30B-A3B-Thinking	14.3	39.8	21.8	12.4
Qwen3-Coder-30B-A3B-Instruct	30.4	49.7	24.3	19.1
OpenForge-Claw(SFT)	21.7	52.1	32.1	23.6
OpenForge-Claw(SFT+RL)	**31.7**	**55.9**	**33.7**	**28.1**

RL, we continue from the SFT checkpoint and train with GRPO (Guo et al., 2025; Shao et al., 2024). To instantiate OPENFORGE RL, we use veRL (Sheng et al., 2024) as the training backend and Microsoft Azure as the cloud provider for rollout containers. We use a batch size of 8 and a group size of 8, and train on 8×B200 GPUs. For additional details, including training hyperparameters and training curves, see Section B.1.

Evaluation Benchmarks We evaluate on two popular claw-based benchmarks, ClawEval (Ye et al., 2026) and QwenClawBench (Qwen Team & Data Team, 2026), and one broader tool-call benchmark, MCPAtlas (Bandi et al., 2026). For ClawEval, we use the 2026-04-08 release and follow the official protocol, reporting $pass^3$ and $pass@3$ with the benchmark's default ReACT loop as the harness. For QwenClawBench, we follow the official implementation and solve the tasks with OpenClaw (OpenClaw, 2025). We report $pass@1$ as the main metric. For MCPAtlas, we use the benchmark's default 20-server configuration, which excludes optional servers requiring third-party credentials or service-specific data initialization for more reproducible results. In total, this includes 89 tasks whose ground-truth expected tool calls are fully supported by this server set. We then follow the official setting and use the benchmark's LLM-as-judge claim-coverage evaluator, counting a task as successful when its coverage score is at least 0.75. We report $pass@1$ as the main metric. For more details on evaluation, please see Section D.

Results We present our results in Table 1. Compared to models of similar size (around 30B, or MoE models with ∼3B active parameters) and to the untrained Qwen3-30B-A3B-Thinking backbone, our OpenForge-Claw models achieve superior results across all three benchmarks. This indicates the effectiveness of our curated tasks and environments, which provide useful learning signal for the model to solve everyday tasks through harnesses such as OpenClaw. Next, compared to OpenForge-Claw(SFT), our OpenForge-Claw(SFT+RL) shows a significant improvement in both robustness ($pass^3$ on ClawEval) and average success rate ($pass@1$ on QwenClawBench and MCPAtlas). This indicates the effectiveness of our OPENFORGE RL training infrastructure, which allows the model to explore and learn from online interaction with the environments and harnesses. For evaluation across different harnesses and generalization to unseen harnesses, please refer to Section 5.1 and Section E.1, respectively.

Table 2: GUI-agent performance on OSWorld-verified, OnlineMind2Web, and WebVoyager. All results are $pass@1$. *marks numbers reported by model's official technical report; †marks numbers reported in Yang et al. (2026); Gupta et al. (2026). Best in shown in **bold**, second shown in *gray*.

System	#Steps	OSWorld-Verified	OnlineMind2Web	WebVoyager
SOTA Vision Language Models				
GPT 5.4*†	100	75.0	92.8	–
Claude Opus 4.6*†	100	72.7	84.0	–
Gemini 3.1 Pro*	100	76.2	–	–
Kimi K2.5*	100	63.3	60.4	74.3
Qwen3-VL 235BA22B*	–	38.1	63.7	66.4
OpenCUA-32B*	50	34.1	–	–
Similar-size baselines and our model (~8B)				
MolmoWeb-8B†	30	–	35.3	**78.2**
OpenCUA-7B*	50	28.2	–	–
UI-TARS-1.5-7B*†	100	27.4	31.3	66.4
Qwen3-VL-8B	50	29.4	38.7	49.2
OpenForge-GUI(SFT)	30	*34.4*	*57.4*	61.5
OpenForge-GUI(SFT+RL)	30	**37.7**	**63.0**	*72.3*

4.2 GUI AGENTS

Beyond diverse harnesses, we also explore OPENFORGE RL training in *diverse environments*, such as multimodal GUI environments that require visual perception and low-level mouse and keyboard control in computer-use and browser-use tasks.

SFT and RL Data Following the same procedure as in Section 4.1, we use our pipeline in Section 3.3 to automatically generate a large pool of tasks and environments. To seed the proposal stage for computer-use tasks, we give the pipeline a reference and artifact pool consisting of (1) X (formerly Twitter) search API access[4], (2) 22k instructions from AgentNet (Wang et al., 2025b), and (3) synthetic files and data from Synthetic-Computer-Use (Ge et al., 2026) that populate each environment with realistic assets. To build containerized computer-use environments, we render a virtual display with Xvfb and pre-install a GUI harness (e.g., Kimi-Agent), so that a VLM can control the machine through simulated mouse clicks (e.g., `left_click(x,y)`) and keyboard actions (e.g., `type("text")`). For browser-use, since synthesizing realistic websites and their databases is impractical, we instead follow OpenWebRL (Yang et al., 2026) and draw real-website tasks from existing datasets. Specifically, we (1) start with tasks from WebGym (Bai et al., 2026); (2) filter tasks that overlap the evaluation benchmarks and also restricting to popular websites; (3) perform deduplication. This results in an SFT pool of 2500 tasks and 900 RL tasks. As in OpenWebRL, we prompt GPT-4.1 (OpenAI, 2025b) as the evaluator to determine task success during both SFT data construction and RL training. To build containerized browser-use environments, we pre-install a GUI-browser harness (i.e., Molmo-Web) and used remote browser service from Browser-Use (Browser Use, 2026) to interact with real websites. For more details on our GUI training data, please see Section C.4.

Training Details We use Qwen3-VL-8B-Thinking (Yang et al., 2025) as the backbone model for all training. For SFT, we distill trajectories from a stronger teacher model (Kimi-K2.5, Team et al. (2026)): we sample $N = 3$ rollouts per task and keep only the successful ones for training. For RL, we also follow the previous section - we instantiate OPENFORGE RL with veRL as the training backend and Microsoft Azure as the cloud provider for rollout containers. We use GRPO with a batch size of 8 and a group size of 8, and train on 8×B200 GPUs. All models use screenshots as their visual input for both training and evaluation. For additional details, such as training hyperparameters and training curves, please see Section B.2.

[4]The agent can use the X search API to find real-world use cases of computer-use agents.

Table 3: Comparing OpenForge-Claw on ClawEval with different harnesses.

	ClawEval($pass@1$)				**ClawEval**($pass@3$)			
Model	ReACT*	ZeroClaw	OpenClaw	Codex	ReACT*	ZeroClaw	OpenClaw	Codex
Qwen3-30B-A3B-Thinking	26.1	32.5	11.4	12.2	39.8	44.7	19.3	18.6
OpenForge-Claw(SFT)	36.2	44.5	16.7	21.1	52.1	66.5	24.2	35.4
OpenForge-Claw(SFT+RL)	**45.1**	**48.5**	**20.9**	**32.5**	**55.9**	**67.1**	**27.8**	**51.5**

Evaluation Benchmarks We evaluate on three popular GUI benchmarks: OSWorld-Verified (Xie et al., 2024) for computer-use environments, and Online-Mind2Web (Xue et al., 2025) and WebVoyager (He et al., 2024) for browser-use. For OSWorld-Verified, we follow the official protocol and report the average success rate. For browser-use, we follow Yang et al. (2026); Awadallah et al. (2025) and report average success rate using the AgentTrek protocal (Xu et al., 2025) with o4-mini (OpenAI, 2025c) for Online-Mind2Web and the official protocal with GPT-4o (OpenAI, 2024) for WebVoyager. More details about each benchmark can be found in Sections D.3 and D.4.

Results We present our results in Table 2. Compared to similar-size (around 8B) models that were specifically fine-tuned for computer-use or browser-use (e.g., OpenCUA, UI-TARS, and MolmoWeb), our OpenForge-GUI models achieve superior results on nearly all benchmarks. Notably, while MolmoWeb is trained on over 200k tasks, OpenForge-GUI uses only 2.5k tasks yet outperforms it on Online-Mind2Web and stays competitive on WebVoyager. More importantly, OpenForge-GUI(SFT+RL) improves substantially over OpenForge-GUI(SFT) on all three benchmarks. This is a considerably harder test of OPENFORGE RL than the text-only setting: every GUI rollout runs a full VLM harness inside a containerized virtual display, perceives the screen visually, and issues long sequences of low-level mouse and keyboard actions against real applications and websites. These consistent gains show that OPENFORGE RL generalizes beyond diverse harnesses to diverse, highly complex environments, training agents end-to-end even under demanding multi-turn, multimodal GUI tasks.

5 DISCUSSION

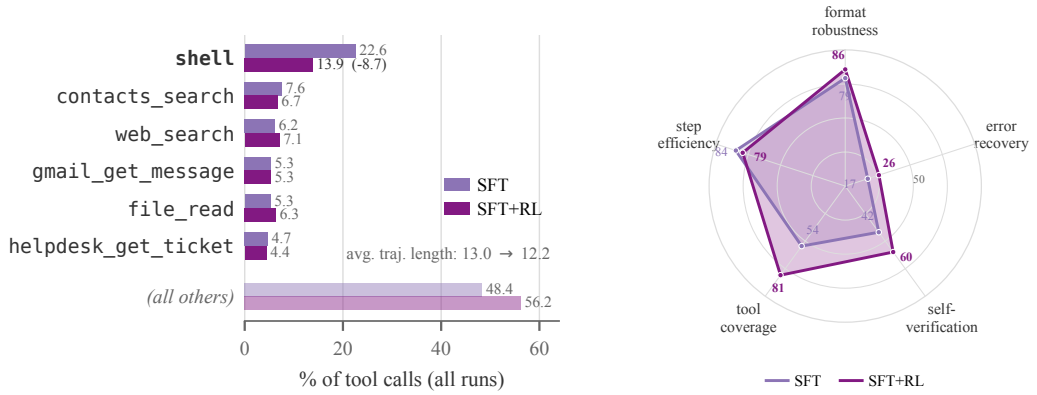
In this section, we analyze how the choice of harness affects learning and what our models learn from harness-based SFT and RL training. We organize the analysis around two questions. (1) *Are some harnesses harder to master than others?* (Section 5.1). (2) *What capabilities does RL add on top of SFT?* (Section 5.2). For simplicity, we focus on our study with OpenForge-Claw models evaluated on the ClawEval benchmark.

5.1 EVALUATION ACROSS DIFFERENT HARNESSES

First, we study whether some harnesses are harder to learn than others. In principle, any task is solvable by any harness; in practice, prior work (Yang et al., 2024; Wang et al., 2024) also find tools and control flows are better aligned with the model's capabilities can often reach higher performance.

We therefore evaluate our trained OpenForge-Claw models on ClawEval using four harnesses of increasing sophistication: ReACT*, ZeroClaw, OpenClaw, and Codex. ReACT* is the benchmark's original harness, a ReACT-like loop that repeatedly prompts the model to reason and call tools, and ZeroClaw (ZeroClaw, 2025) is a lightweight version of OpenClaw that adds an easy way to register new tools. OpenClaw (OpenClaw, 2025) and Codex (OpenAI, 2025a) are far more advanced, offering a rich set of built-in tools and control flows, but neither easily supports custom tools; we therefore expose each ClawEval-specific tool to them through `SKILL.md` files (see Section D.1).

Table 3 reports the results, from which two trends stand out. First, the harnesses that support adding custom tools directly (ReACT and ZeroClaw) reach the highest performance. Second, SFT+RL brings large gains on every harness except OpenClaw, which only has moderate gains while consuming far longer prompts and contexts than the others. This corroborates prior findings that simpler, better-engineered tools and control flows are essential for agentic performance, and further shows that training on such harnesses advances the model's ability to solve tasks in complex environments.



(a) ZeroClaw: tool usage

(b) Codex: behavioral profile

Figure 4: SFT vs. SFT+RL behavior on ClawEval. RL shifts calls from the generic `shell` tool to dedicated service tools (a), and improves agentic capabilities such as self-verification (b).

Beyond comparing harnesses, we also test whether training on one harness transfers to harnesses unseen during training. Training on ZeroClaw alone already generalizes to the unseen OpenClaw (+3.3) and Codex (+4.6), and training on all three harnesses is best across the board (+9.5 on OpenClaw and +20.3 on Codex). Due to limited space, we defer the full setup and results to Section E.1.

5.2 Capability Learned by RL

Finally, we investigate what RL learns on top of SFT. We run the SFT and SFT+RL OpenForge-Claw checkpoints on ClawEval and compare 100 trajectories from each, examining tool-call statistics under the ZeroClaw harness (Figure 4, left) and higher-level behavioral capabilities under the Codex harness (Figure 4, right).

On ZeroClaw, we find RL changes *how* the model uses tools. It reduces generic `shell` calls from 22.6% to 13.9% of all tool calls and redistributes them toward dedicated service tools, while also slightly shortening trajectories (Figure A6 gives the full breakdown). This suggests that RL teaches the model to reach for the right specialized tool instead of falling back on a general-purpose shell.

On Codex, we find RL improves several agentic capabilities (see Section E.2 for detailed definitions) that matter for long-horizon tool use. In particular, it strengthens error recovery (correctly solving a task after a failed command) and self-verification (reading back its own writes to confirm them), and, echoing the ZeroClaw analysis, it exercises a wider set of the tools each task requires. These are the behaviors that keep an agent reliable across many steps. Error recovery, however, remains the weakest capability even after RL. We hypothesize that such capabilities are difficult to acquire from RL alone, and may require dedicated data or training methods to strengthen further.

6 Conclusion

We present OpenForge RL, an open framework that makes *any harness* × *any environment* trainable end-to-end with standard RL codebases such as veRL, so LLM- and VLM-based agents are optimized inside the harnesses they are actually deployed with rather than in simplified reimplementations. Using only hundreds to a few thousand automatically curated tasks, we train OpenForge-Claw and OpenForge-GUI models that surpass open models of similar size on nearly all of our tool-use and GUI benchmarks, and in the GUI setting match or exceed models several times larger. Beyond training, OpenForge RL also lets us study how harness choice and RL shape agent behavior, which prior open work could not easily do: some harnesses are substantially harder to learn than others, gains transfer to harnesses unseen during training, and RL mainly improves *agentic reliability* such as self-verification and tool coverage, though error recovery remains weak and likely needs dedicated data. We release our code, data, and models to lower the barrier to training and studying agents in their real deployment settings.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human, 2024. URL https://arxiv.org/abs/2410.08164.

Zhipu AI. Zclawbench. https://huggingface.co/datasets/zai-org/ZClawBench, 2026. Accessed: 2026-07-20.

Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, 2024. Accessed: 2026-07-20.

Anthropic. Overview - Claude Code Docs. https://code.claude.com/docs/en/overview, 2025. Accessed: 2026-07-20.

Anthropic. Introducing claude 4.6. https://www.anthropic.com/news/claude-opus-4-6, 2026. Accessed: 2026-07-20.

Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-7b: An efficient agentic model for computer use. *arXiv:2511.19663*, 2025.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. Swe-rebench v2: Language-agnostic swe task collection at scale, 2026. URL https://arxiv.org/abs/2602.23866.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning, 2024. URL https://arxiv.org/abs/2406.11896.

Hao Bai, Alexey Taymanov, Tong Zhang, Aviral Kumar, and Spencer Whitehead. Webgym: Scaling training environments for visual web agents with realistic tasks, 2026. URL https://arxiv.org/abs/2601.02439.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, Manasi Sharma, Andrew Park, Martin Dimakis, Ernesto Gabriel Hernandez Montoya, Dan Rambado, Ivan Salazar, Rafael Cruz, MohammadHossein Rezaei, Chetan Rane, Ben Levin, Daniel Yue Zhang, Brad Kenstler, and Bing Liu. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers, 2026. URL https://arxiv.org/abs/2602.00933.

Browser Use. Stealth browsers for ai agents. https://browser-use.com/stealth-browsers, 2026.

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2026. URL https://arxiv.org/abs/2505.24298.

Tao Ge, Baolin Peng, Hao Cheng, and Jianfeng Gao. Synthetic computers at scale for long-horizon productivity simulation, 2026. URL https://arxiv.org/abs/2604.28181.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli

Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Tanmay Gupta, Piper Wolters, Zixian Ma, Peter Sushko, Rock Yuren Pang, Diego Llanes, Yue Yang, Taira Anderson, Boyuan Zheng, Zhongzheng Ren, Harsh Trivedi, Taylor Blanton, Caleb Ouellette, Winson Han, Ali Farhadi, and Ranjay Krishna. Molmoweb: Open visual web agent and open data for the open web, 2026. URL https://arxiv.org/abs/2604.08516.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024. URL https://arxiv.org/abs/2401.13919.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2024. URL https://arxiv.org/abs/2312.08914.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.

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?, 2024. URL https://arxiv.org/abs/2310.06770.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.09516.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

MiniMax. Minimax-m2.5. https://github.com/MiniMax-AI/MiniMax-M2.5, 2026. Accessed: 2026-07-20.

OpenAI. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/, 2024. Accessed: 2024-09-28.

OpenAI. Introducing Codex. https://openai.com/index/introducing-codex/, 2025a. Accessed: 2026-07-20.

OpenAI. Introducing GPT-4.1 in the api. https://openai.com/index/gpt-4-1/, 2025b. Accessed: 2025-09-17.

OpenAI. Introducing OpenAI o3 and o4-mini, 2025c. URL https://openai.com/index/introducing-o3-and-o4-mini/.

OpenAI. Harness engineering: leveraging Codex in an agent-first world. https://openai.com/index/harness-engineering/, 2026. Accessed: 2026-07-20.

OpenClaw. OpenClaw — Personal AI Assistant. https://openclaw.ai/, 2025. Accessed: 2026-07-20.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Baolin Peng, Wenlin Yao, Qianhui Wu, Hao Cheng, Xiao Yu, Rui Yang, Tao Ge, Alessandro Sordoni, Xingdi Yuan, Yelong Shen, Pengcheng He, Tong Zhang, Zhou Yu, and Jianfeng Gao. Orchard: An open-source agentic modeling framework, 2026. URL https://arxiv.org/abs/2605.15040.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning, 2025. URL https://arxiv.org/abs/2411.02337.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

Qwen Team and Alibaba Group Data Team. QwenClawBench: Real-user-distribution benchmark for openclaw agents, April 2026. URL github.com/SKYLENAGE-AI/QwenClawBench.

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.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, Yanru Chen, Yanxu Chen, Yicun Chen, Yimin Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Ziwei Chen, Dazhi Cheng, Minghan Chu, Jialei Cui, Jiaqi Deng, Muxi Diao, Hao Ding, Mengfan Dong, Mengnan Dong, Yuxin Dong, Yuhao Dong, Angang Du, Chenzhuang Du, Dikang Du, Lingxiao Du, Yulun Du, Yu Fan, Shengjun Fang, Qiulin Feng, Yichen Feng, Garimugai Fu, Kelin Fu, Hongcheng Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Chengyang Gong, Xiaochen Gong, Zhuoma Gongque, Qizheng Gu, Xinran Gu, Yicheng Gu, Longyu Guan, Yuanying Guo, Xiaoru Hao, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Jiaxi Hu, and et al. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Brandon Trabucco, Gunnar Sigurdsson, Robinson Piramuthu, and Ruslan Salakhutdinov. Insta: Towards internet-scale training for agents, 2025.

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, 2023. URL https://arxiv.org/abs/2305.16291.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents, 2024. URL https://arxiv.org/abs/2402.01030.

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, 2025a. URL https://arxiv.org/abs/2407.16741.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. Opencua: Open foundations for computer-use agents, 2025b. URL https://arxiv.org/abs/2508.09123.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution, 2025. URL https://arxiv.org/abs/2502.18449.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use, 2025. URL https://arxiv.org/abs/2509.24002.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents, 2024. URL https://arxiv.org/abs/2407.01489.

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents, 2026. URL https://arxiv.org/abs/2506.14205.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

Binfeng Xu, Hao Zhang, Shaokun Zhang, Songyang Han, Mingjie Liu, Jian Hu, Shizhe Diao, Zhenghui Jin, Yunheng Zou, Michael Demoret, Jan Kautz, and Yi Dong. Polar: Agentic rl on any harness at scale. *arXiv preprint arXiv:2605.24220*, 2026.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials, 2025. URL https://arxiv.org/abs/2412.09605.

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.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin

Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

Rui Yang, Qianhui Wu, Yuxi Chen, Hao Bai, Wenlin Yao, Hao Cheng, Baolin Peng, Huan Zhang, Tong Zhang, and Jianfeng Gao. Openwebrl: Demystifying online multi-turn reinforcement learning for visual web agents, 2026. URL https://arxiv.org/abs/2606.02031.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-eval: Towards trustworthy evaluation of autonomous agents, 2026. URL https://arxiv.org/abs/2604.06132.

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, 2025a. URL https://arxiv.org/abs/2503.14476.

Xiao Yu, Baolin Peng, Michel Galley, Hao Cheng, Qianhui Wu, Janardhan Kulkarni, Suman Nath, Zhou Yu, and Jianfeng Gao. Dyna-mind: Learning to simulate from experience for better ai agents, 2025b. URL https://arxiv.org/abs/2510.09577.

ZeroClaw. ZeroClaw: The Ultra-Lightweight AI Agent Runtime. https://zeroclaw.net/, 2025. Accessed: 2026-07-20.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025. URL https://arxiv.org/abs/2506.05176.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levione, and Erran Li. Proposer-agent-evaluator (PAE): Autonomous skill discovery for foundation model internet agents. In *ICML*, 2025. URL https://arxiv.org/abs/2412.13194.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A AI USE STATEMENT

This work used LLMs as general-purpose writing assistants to improve the grammar and clarity of the paper. We *did not* use LLMs to generate any research ideas, automate experiments, or analyze results.

B TRAINING DETAILS

B.1 CLAW AGENT TRAINING DETAILS

For OpenForge-Claw RL training, we use veRL as the training backend and Microsoft Azure as the cloud provider for rollout containers. Each rollout runs in its own container, built from a task-specific Dockerfile with the target harness (e.g., OpenClaw, Codex, or ZeroClaw) pre-installed, and is scheduled onto a Kubernetes pod capped at 2 CPUs and 2GB RAM. These pods are packed onto Azure D128ads v5 nodes, while policy training runs on a single node of 8×B200 GPUs. We report the key training hyperparameters in Table A1 and the training curves in Figure A1.



Figure A1: OpenForge-Claw RL training curves on claw tasks. *Left*: training success rate. *Middle*: validation success rate. *Right*: training episode length.

B.2 GUI AGENT TRAINING DETAILS

For OpenForge-GUI RL training, we similarly use veRL as the training backend and Microsoft Azure as the cloud provider for rollout containers.



Figure A2: OpenForge-GUI RL training curves on computer-use tasks. *Left*: training success rate. *Middle*: validation success rate. *Right*: training episode length.

Computer-Use For computer-use agent training, each rollout runs in its own container built from a task-specific Dockerfile with the target harness (a modified version of Kimi-Agent, see Section D.3) pre-installed, and is scheduled onto a Kubernetes pod capped at 4 CPUs and 4GB RAM. These pods are packed onto Azure D64ads v5 nodes, while policy training runs on a single node of 8×B200 GPUs. We report the key training hyperparameters in Table A1 and the training curves in Figure A2.

Browser-Use For browser-use agent training, each rollout drives a dedicated remote browser session on the Browser-Use Cloud service (hosted Chromium) over the Chrome DevTools Protocol. The per-rollout environment, which handles session management, action execution, screenshot capture, and the reward judge, runs in a lightweight Kubernetes sandbox pod capped at 2 CPUs and 6 GB RAM; the browser itself runs remotely rather than inside the pod. These pods are packed onto Azure D128ads v7 nodes, and up to 48 rollouts run concurrently, while policy training runs on a single node of 8×B200 GPUs. We report the key training hyperparameters in Table A1.

Table A1: Key RL training hyperparameters for OpenForge-Claw and OpenForge-GUI.

Hyperparameter	Claw	Computer-Use	Browser-Use
Learning rate	1e-6	5e-7	1e-6
Batch size	8	8	12
Group size	8	8	5
KL coefficient	0.001	0.01	0.0
Entropy coefficient	0.0	0.0	0.001
Max rollout time / step	900s	600s	900s
Training steps	100	80	100
Rollout VM CPU / pod	2	4	2
Rollout VM Memory / pod	2GiB	4GiB	6GiB
Total Training Time	48H	36H	32H

C Data Details

C.1 Data Synthesis Pipeline Overview

This work aims to train harness-based agents end-to-end across diverse environments beyond coding, such as browser- and computer-use. Unlike coding (Badertdinov et al., 2026), these domains offer far fewer training tasks, harnesses, and RL-ready environments. To experiment with our framework in diverse environments, we therefore also built a simple pipeline to synthesize SFT and RL tasks for such data-scarce domains (e.g., claw/daily tool use and computer use). We give a high-level overview below and defer full details to Section C.2.

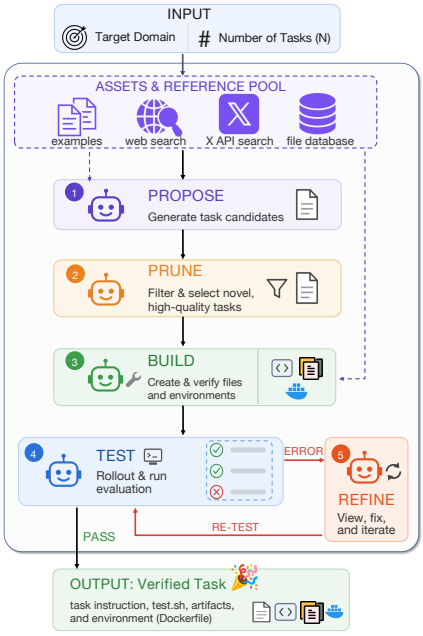


Figure A3: Overview of our data/task synthesis pipeline.

The pipeline is built to mimic how a human would curate a task (Xie et al., 2026; Zhou et al., 2025). Given a target domain and a number of tasks, it spawns agents in parallel that (1) **propose** candidate instructions grounded in realistic scenarios, drawn from the web/X API or a pool of reference assets and instructions; (2) **prune** low-quality and duplicate tasks; (3) **build** an executable environment and a verifier script for each task; (4) **test** the task by rolling out a separate open LLM/VLM; and (5) **refine** it by patching defects until it passes all checks. The test and refine stages are essential: they validate each task end-to-end before it enters the dataset. Figure A3 illustrates the pipeline, and Section C.2 provides full details. Because each environment is defined by a custom Dockerfile, the pipeline extends naturally from Linux/CLI tasks (e.g., claw) to GUI and computer-use tasks (e.g., rendering virtual displays with Xvfb), and can pre-install arbitrary harnesses such as OpenClaw and Codex. The resulting tasks and environments support both SFT (via distillation from a stronger model's rollouts) and RL. We will release these data and this pipeline for future use.

Table A2: Statistics of SFT and RL data used for Claw and GUI agent training. For GUI, we used a slightly modified version of the Kimi-Agent and Molmo-Web harness (see Sections D.3 and D.4).

	Claw	**GUI(Computer)**	**GUI(Browser)**
Harnesses	ReACT*,ZeroClaw,OpenClaw,Codex	Kimi-Agent*	MolmoWeb*
# SFT Trajectories	892	795	1,496
# RL tasks	343	252	900

C.2 More Details on Data Synthesis

We implement our data synthesis pipeline (Section 3.3) on the Claude Agent SDK, using Opus 4.6 (Anthropic, 2026) as the backbone for each of the five agent modules. Given a target domain (as a natural-language prompt) and a target number of tasks N, the pipeline spawns many agents in parallel that run the following five stages.

- **Propose.** Each agent drafts candidate instructions by browsing the assets and reference pool we construct for the target domain, as we find that directly prompting a model to invent tasks from scratch tends to yield infeasible or unoriginal instructions. For Claw agents, this pool comprises (1) skills from ClawHub and (2) tasks from ZClawBench (AI, 2026); for computer-use agents, it comprises (1) X search API access, (2) 22k instructions from AgentNet, and (3) synthetic files and data from Synthetic-Computer-Use. To record drafted instructions and avoid duplicates, we maintain a shared SQLite database and instruct each agent to append its proposals to it.
- **Prune.** From the proposed but unimplemented instructions in the database, these agents select the N highest-quality and most diverse instructions, and mark the remainder as discarded.
- **Build.** For each selected instruction, an agent constructs a fully executable environment: tool servers, mock websites, mock data and files, a Dockerfile that packages everything into an image, and a verifier script that scores task success.
- **Test.** Many environment errors and instruction ambiguities are hard to detect without actually running the task. We therefore provide the agent with a script that invokes a separate open LLM/VLM to attempt the task in the built environment: MiniMax-M2.5 for Claw tasks and Kimi-K2.5 for computer-use tasks.
- **Refine.** Finally, the refine agent inspects the rollout trajectory and the verifier's score and decides whether the environment has any defects or the instruction is ambiguous. If so, it patches the environment and/or revises the instruction, and repeats the test-and-refine loop until the task passes all checks.

C.3 More Details on Claw Data

At the time of the project, we did not find any large scale public dataset suitable for training Claw-based agents, especially for RL training. As a result, we primarily rely on our data synthesis pipeline (Section C.2) to generate tasks for SFT and RL training. While powerful, we find synthesizing a task with a thoroughly tested verifier is costly in both time and money, as it involves multiple rounds of real rollouts, refinement, and re-rollout. To save cost, we instead consider for *SFT tasks*, we skip the test and refinement stage, and directly prompt GPT-5.4 as a judge to determine task success. This is much afforadable and reasonable since for SFT trajectories the success signals are often only used for onetime data filtering. For RL tasks, we maintain the full test-and-refine loop to ensure the verifier is robust and reliable, as RL training requires repeated rollouts and the verifier must be able to consistently judge task success. On average, synthesizing an RL task (with verifier and refining) takes 16.1 minutes and 4.36 USD, while synthesizing an SFT task (without verifier) takes 5.2 minutes and 0.86 USD. A full breakdown of our task distribution is shown in Figure A7.

For harnesses, in addition to the default ReACT loop implemented by most tool-use agents, we also include three popular harnesses: ZeroClaw, OpenClaw, and Codex, to experiment with the capabilities of our OpenForge RL infrastructure.

C.4 More Details on GUI Data

Computer-use tasks At the time of this work, there was no large-scale public dataset for training computer-use agents, especially with RL. One reason is that running a computer-use GUI in a lightweight container is challenging: such environments are commonly run as full Ubuntu virtual machines under QEMU (Xie et al., 2024). We find, however, that Xvfb can instead render a virtual display in memory, which is far more lightweight and lets us run many containers in parallel for RL training. With computer-use GUIs now runnable in lightweight containers, we follow our Claw setup and use the same data synthesis pipeline (Section C.2) to generate SFT and RL tasks. For *SFT tasks*, we again skip the test-and-refine stages and prompt GPT-5.4 as a judge to determine task success. On average, each SFT task takes 4.0 minutes and 1.37 USD to synthesize. For *RL tasks*, we keep the full test-and-refine loop so that the verifier is robust and reliable. On average, synthesizing an RL task (with verification and refinement) takes 21.3 minutes and 6.12 USD.

As the harness, we primarily use Kimi-Agent (Team et al., 2026), following the implementation of Xie et al. (2024), with slight modifications to support additional tools such as bash, following Anthropic's computer-use approach. For the full action space and tool list, see Section D.3.

Browser-use tasks For browser-use, we adapt MolmoWeb (Gupta et al., 2026)'s codebase as the harness and our main modifications involve (1) using json-formatted action space other than MolmoWeb's tool-call format to avoid extra training stages for action alignment, and (2) integrating Browser-Use Stealth Browsers (Browser Use, 2026) following OpenWebRL to solve CAPTCHA and website blockings, which we found reducing the ratio of IP and CAPTCHA block from 40% to nearly zero. We provide a more detailed description and examples of the observation and action spaces for both environments in Section D.4.

For SFT data, we follow a similar data curation pipeline to OpenWebRL including task-filtering and trajectory collection. We start from subsampling the PAE-WebVoyager (Zhou et al., 2025) split from WebGym (Bai et al., 2026) by removing tasks that overlap with evaluation benchmarks and subtasks decomposed from parent intents tasks. Then, we keep only the most popular websites that exists in SimilarWeb Top100 and MOZ Top 500 to remove the distrations of unusual websites in the long-tail distribution. Finally, to remain the diversity of tasks, we embed task instructions with Qwen3-Embedding-8B (Zhang et al., 2025) and apply greedy similarity-based deduplication with a predefined threshold 0.55. The eventual SFT candidate pool contains 2500 tasks.

We apply a stronger teacher model (Kimi-K2.5) to infer on the candidate pool with max turns 30 and 4 repeats. We collect all successful trajectories, and keep only the shortest trajectory among multiple success of a single task. We found that repeated actions commonly appear for complex websites, and we reserve only the last turn for more than 3 consecutive identical actions anf remove the entire trajectory if more than 5 identical actions occur in the process. Following the above operations, we curate 1496 trajectories for distillation.

During RL stage, we apply the same popular website filter as SFT and randomly sample 600 tasks from the Insta-v3 (Trabucco et al., 2025) and 300 tasks from the PAE-WebVoyager (Zhou et al., 2025) split that are distinct from the SFT pool. The maximum turn is restricted to 20 to balance the performance and training cost.

D Evaluation Details

D.1 ClawEval and QwenClawBench Evaluation Details

We evaluate our OpenForge-Claw models on popular claw- and harness-related benchmarks that measure how well an agent can use a harness to solve tasks. Specifically, we use ClawEval (Ye et al., 2026), QwenClawBench (Qwen Team & Data Team, 2026), and MCPAtlas (Bandi et al., 2026), the last serving as a related but "held-out" test set for novel tool use. All results in our main experiments (Section 4.1) use each benchmark's official evaluation protocol: for ClawEval, the official ReACT-like loop repeatedly prompts the model to reason and call tools; for QwenClawBench, the OpenClaw harness is used.

To study the effect of harness choice on model performance, we additionally evaluate on ClawEval under three other harnesses: ZeroClaw, OpenClaw, and Codex (Section 5.1). Because ClawEval re-

quires the model to call custom tool servers, ZeroClaw was straightforward to support, as it natively allows registering new tools. To adapt OpenClaw and Codex, which is non-trivial for adding custom tools, we expose each ClawEval-specific tool to them through `SKILL.md` files. Specifically, for each custom tool server, we prompt an LLM (Claude Opus 4.6) with its API signature and tool descriptions to generate a `SKILL.md` file that explains how to call the tool from a bash command line. At inference time, these files are loaded into the harness, and the model invokes the tools through the bash interface. Evaluation protocols are unchanged from the official ClawEval benchmarks.

D.2 MCP-Atlas Evaluation Details

For reproducibility, we evaluate MCPAtlas (Bandi et al., 2026) under the benchmark's default 20-server configuration, which excludes optional servers that require third-party credentials or service-specific data initialization. This configuration fixes the evaluation set without any manual selection on our part: of the 500 public tasks, exactly 89 have ground-truth expected tool calls that are fully supported by these default servers. We evaluate every model on this same 89-task set, holding the task identifiers and environment configuration fixed.

To evaluate, we use the MCPAtlas official harness which is based on a ReACT-like loop: the harness exposes the task-specific MCP tools to the policy model, executes its tool calls in the MCPAtlas sandbox, and returns the resulting observations until the model terminates. We use the default task prompts and tool configurations, and add no benchmark-specific demonstrations or fine-tuning.

Following the MCPAtlas claim-coverage protocol, we score each final response against the ground-truth factual claims using Gemini 2.5 Pro as the judge. A task counts as successful when its claim-coverage score is at least 0.75, and we report the fraction of successful tasks as pass@1. Every model is evaluated under the same task subset, harness, judge, and threshold.

D.3 Computer-Use Evaluation Details

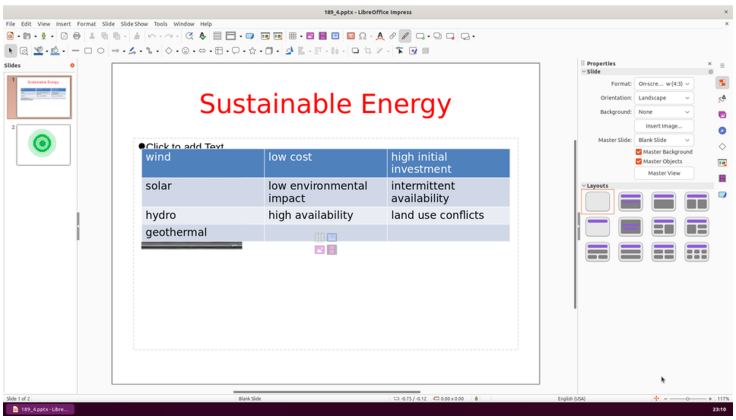
For computer-use environments, we evaluate on OSWorld-Verified (Xie et al., 2024). OSWorld is a popular benchmark that evaluates how well a multimodal agent can complete open-ended, real-world computer tasks on a real Ubuntu desktop, operating applications through the screen with mouse and keyboard and being scored by task-specific, execution-based verifiers. OSWorld-Verified is an in-place upgrade of OSWorld with enhanced infrastructure and improved task quality built from 300+ pieces of feedback from the community.

Evaluating on OSWorld-Verified first requires choosing a harness. Because our computer-use data pipeline is built around Kimi-K2.5 (Team et al., 2026), we use Kimi-Agent, as implemented in the official OSWorld repository, as our main training and evaluation harness. To make the harness more efficient, we add a lightweight modification inspired by Anthropic's computer-use approach: we additionally expose `bash` and a `str_replace_editor` tool, letting the model directly manipulate files and data (e.g., editing text, CSV, and JSON files) without opening a GUI application. We run Kimi-Agent in screenshot-only mode: every observation is a screenshot of the GUI, with no additional text or metadata. We show an example trajectory in Figure A4 and the full action space and tool list in Table A3.

D.4 Browser-Use Evaluation Details

For browser-use, we evaluate on Online-Mind2Web (Xue et al., 2025) (300 tasks) and WebVoyager (He et al., 2024), the latter on the 595-task "Fara-595" subset released with Fara-7B (Awadallah et al., 2025). We use our modified MolmoWeb harness (Section C.4), which exposes a single `computer_use` tool; the full action space is listed in Table A4. Observations are screenshot-only: each step provides the current 1280×720 viewport screenshot, with no DOM or accessibility tree, together with a short text block giving the previous action's result, the page title and URL, and the step count. Click and drag coordinates are normalized to $[0, 1]$ relative to the screenshot, the same convention as our computer-use agent. At inference, we cap each episode at 30 steps, with a 900 s per-task and 90 s per-action timeout, and score Online-Mind2Web with the AgentTrek protocol (o4-mini) and WebVoyager with the official protocol (GPT-4o). We show an example step in Figure A5.

Task instruction. Go to the second slide and name its title as “Online Shopping” with the same color, position, and font size as the previous title.



Model output.
<think>
The user wants me to go to the second slide and set its title to “Online Shopping” with the same formatting as the current title “Sustainable Energy”. The current title is red, in a large font, at the top center. On the left, a slide panel shows “Slide 1 of 2”, and slide 2 appears blank. I should click on slide 2 to navigate to it first, then add the title with the same formatting. [. . .]
</think>
<tool_call>
{“name”: “computer”, “arguments”: {“action”: “left_click”, “coordinate”: [0.054, 0.337]}}
</tool_call>

Figure A4: An example computer-use input-output pair. Given the task instruction and the current screenshot, the model emits a single generation with reasoning (`<think>`, truncated here) and a tool call (`<tool_call>`). To improve readability, in this example we overlay the predicted normalized coordinate with a green marker on the screenshot.

E Analysis Details

E.1 Generalization to Unseen Harness

Next, we ask whether training on one harness transfers to unseen harnesses in evaluation, and whether training on several harnesses at once improves over training on a single one. We compare two OpenForge-Claw models built from the same tasks and the same training recipe, differing only in which harnesses generate their rollouts (for both SFT distillation and RL): one trained on ZeroClaw alone, and one trained on ZeroClaw, OpenClaw, and Codex together. We evaluate both on all three harnesses (Table A5). For the ZeroClaw-only model, OpenClaw and Codex are unseen.

Table A5: Unseen harness evaluation. All training are from Qwen3-30B-A3B-Thinking. Deltas are over the base model.

	ClawEval($pass@1$)		
Training (SFT+RL)	ZeroClaw	OpenClaw	Codex
None (base)	32.5	11.4	12.2
ZeroClaw	46.0 (+13.5)	14.7 (+3.3)	16.8 (+4.6)
ZeroClaw+OpenClaw+Codex	**48.5** (+16.0)	**20.9** (+9.5)	**32.5** (+20.3)

We observe two effects. First, training on a single harness already generalizes to the others: the ZeroClaw-only model improves over the untrained base on the unseen OpenClaw (+3.3) and Codex (+4.6). Second, training on all three harnesses is best across the board, with the largest gains over the base on the more complex harnesses (+9.5 on OpenClaw and +20.3 on Codex), and it even lifts ZeroClaw itself beyond ZeroClaw-only training (48.5 vs. 46.0). We attribute this to the greater diversity of tool calls and control flows the model sees when trained on multiple harnesses, which makes it more robust across different tools and scenarios.

Table A3: Action space of our computer-use agent. The agent controls an Ubuntu desktop through three tools: `computer` (mouse, keyboard, scrolling, and episode control), `bash` (shell access), and `str_replace_editor` (file editing). Coordinates are normalized to $[0, 1]$ relative to the screenshot (x=0 left, x=1 right, y=0 top, y=1 bottom), backed by a 1920×1080 screen.

Action / command	**Arguments**	**Description**
`computer` — desktop mouse, keyboard, scrolling, and control		
`left_click`, `right_click`, `middle_click`, `double_click`, `triple_click`, `left_press`	`coordinate`, `keys`	Click at `coordinate` with the given button and click count; `left_press` presses and holds briefly. `keys` adds modifiers.
`mouse_move`, `left_click_drag`, `left_mouse_down`, `left_mouse_up`	`coordinate`, `start_coordinate`, `coordinate2`	Move the cursor, drag from `start_coordinate` to `coordinate2`, or press/release the left button for manual drags.
`key`, `type`, `hold_key`, `key_down`, `key_up`	`text`, `keys`, `duration`	Type `text`, press a key or hotkey (e.g., `ctrl+l`), or hold / press / release specific `keys`.
`scroll`, `hscroll`	`scroll_direction`, `scroll_amount`, `coordinate`	Vertical or horizontal mouse-wheel scroll by `scroll_amount` in `scroll_direction`, optionally anchored at `coordinate`.
`screenshot`, `wait`, `terminate`, `done`, `fail`	`duration`, `status`	Capture the screen, wait, or end the episode as success / failure (`status`).
`bash` — shell access		
`bash`	`command`, `timeout`, `working_dir`	Run a shell `command` inside the desktop environment (not the host).
`str_replace_editor` — file viewing and editing		
`view`, `create`, `str_replace`, `insert`, `undo_edit`	`path`, `file_text`, `old_str`, `new_str`, `insert_line`, `view_range`	View, create, and edit files with precise text operations at absolute `path`s.

E.2 Behavioral Analysis Details

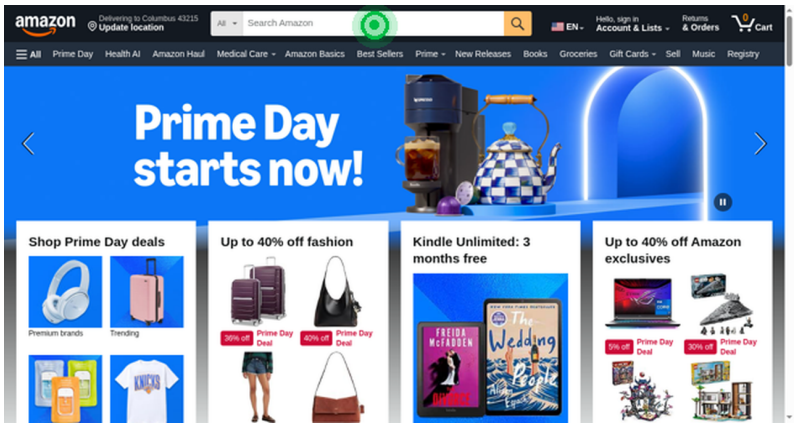
We analyze behavioral differences between the SFT and SFT+RL OpenForge-Claw checkpoints on ClawEval, running each checkpoint under the ZeroClaw and Codex harnesses.

Tool usage (ZeroClaw). For each tool, we report the percentage of all tool calls that invoke it, aggregated over all rollouts. This captures which tools the model relies on, independent of whether a task is ultimately solved.

Behavioral capabilities (Codex). We summarize five capabilities relevant to long-horizon tool use, shown as the radar plot in Figure 4 (right). Each is a percentage in $[0, 100]$ where higher is better, computed over rollouts from the two OpenForge-Claw checkpoints on the same set of tasks.

- **Format robustness.** The percentage of rollouts *not* terminated by a malformed tool call, i.e., 100 minus the fraction of rollouts in which the harness fails to parse the model's function-call payload (typically a long structured output) and the session disconnects. It measures how reliably the model emits well-formed tool calls.
- **Error recovery.** Among rollouts that hit at least one failed command, the percentage that still solve the task. It measures whether the model can proceed after an error rather than crashing or giving up.

Task instruction. Find a pair of wireless headphones on Amazon with active noise canceling for $100 or less and add them to the cart.



Model output.
<think>
Good, I'm now on the Amazon homepage. I need to search for wireless headphones with active noise canceling for $100 or less. I'll click on the search bar and type my search query.
</think>
<tool_call>
{"name": "computer_use", "arguments": {"action": "left_click", "coordinate": [0.469, 0.048]}}
</tool_call>

Figure A5: A representative browser-use step on an Online-Mind2Web task. Given the task instruction and the current screenshot, the model emits a single generation that interleaves its reasoning (`<think>`) and a tool call (`<tool_call>`). We overlay the predicted normalized click coordinate with a green marker on the screenshot, which correctly targets the Amazon search bar.

Table A4: Action space of our browser-use agent. The model controls a Chromium browser through a single `computer_use` tool. Click and drag coordinates are normalized to $[0, 1]$ relative to the screenshot, which is pinned to a 1280×720 viewport.

Action	Arguments	Description
`computer_use` — browser mouse, keyboard, scrolling, navigation, and control		
`left_click`, `right_click`, `middle_click`, `double_click`, `triple_click`	`coordinate`	Click at `coordinate` with the given button and click count (double/triple select a word/line).
`mouse_move`, `left_click_drag`	`coordinate`	Move the cursor to `coordinate`, or drag from the current cursor position to it.
`type`, `key`	`text`, `keys`	Type `text` into the focused field, or press a key or chord (e.g., `["ctrl","a"]`).
`scroll`, `hscroll`	`pixels`	Vertical or horizontal scroll at the cursor by `pixels` (sign gives direction).
`goto`, `go_back`, `new_tab`, `tab_focus`	`url`, `index`	Navigate to `url`, go back in history, open a new tab, or switch to the tab at `index`.
`wait`, `terminate`, `answer`	`time`, `status`, `text`	Wait for `time` seconds, end the episode (`terminate` with success / failure), or return the final `answer`.

- **Self-verification.** The percentage of write (state-changing) tool calls that are followed by a read-back of the same service, e.g., listing tasks after creating one. It measures whether the model checks the effect of its own actions.

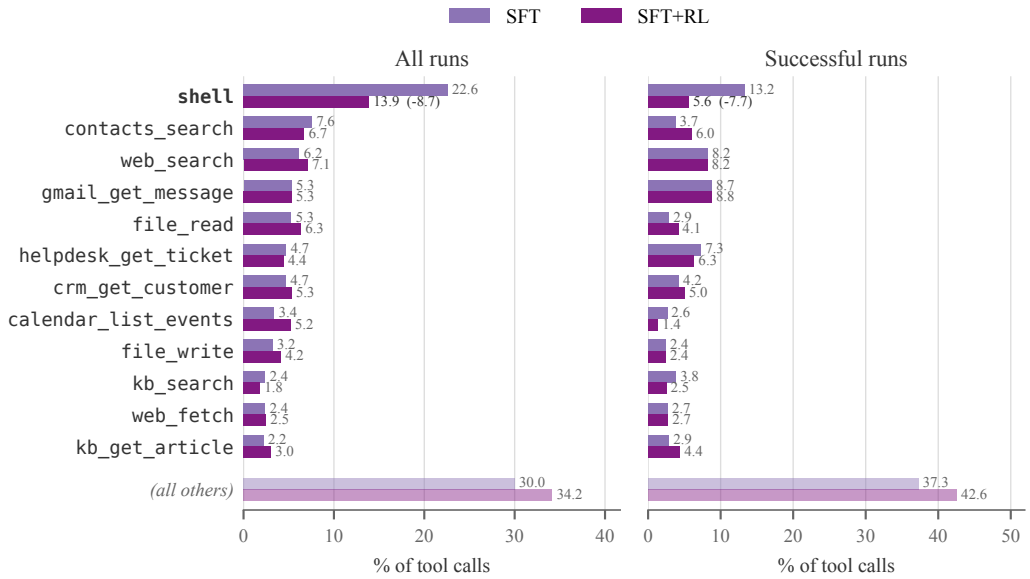


Figure A6: Full tool-usage distribution on ClawEval with the ZeroClaw harness for SFT vs. SFT+RL, over all runs (left) and successful runs (right). Complements Figure 4: the shift away from the generic `shell` tool is even stronger when restricted to successful runs (13.2% $\rightarrow$ 5.6%).

- **Tool coverage.** On tasks that require at least three distinct services, the percentage of rollouts that invoke *every* required service at least once. It measures whether the model carries a multi-part plan through to completion.
- **Step efficiency.** For each task, we define the *best* step count as the fewest steps among all successful runs (from 3 runs per task) of that task across both checkpoints. A checkpoint's score on a task it solves is best/(its own step count), averaged over its solved tasks. A higher value means the model solves tasks in closer to the minimum number of steps.

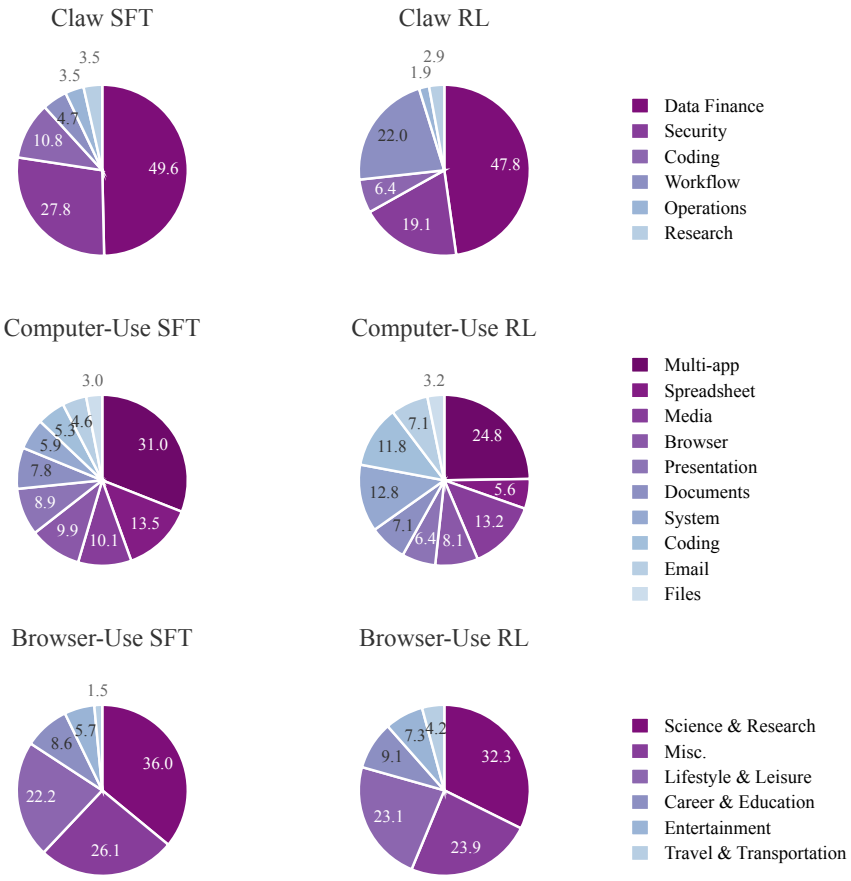


Figure A7: Category distribution (%) of SFT and RL training tasks used in the Claw (top), Computer-Use (middle), and Browser-Use (bottom) domains.