

# Orchard: An Open-Source Agentic Modeling Framework

Baolin Peng<sup>†1</sup>✉, Wenlin Yao<sup>†1</sup>, Qianhui Wu<sup>†1</sup>, Hao Cheng<sup>†1</sup>  
 Xiao Yu<sup>‡2</sup>, Rui Yang<sup>‡3</sup>, Tao Ge<sup>1</sup>, Alessandro Sordoni<sup>1</sup>, Xingdi Yuan<sup>1</sup>, Yelong Shen<sup>1</sup>  
 Pengcheng He<sup>1</sup>, Tong Zhang<sup>3</sup>, Zhou Yu<sup>2</sup>, Jianfeng Gao<sup>1</sup>✉  
<sup>1</sup>Microsoft Research <sup>2</sup>Columbia University <sup>3</sup>UIUC  
 {baolinpeng, jfgao}@microsoft.com

 GitHub  Hugging Face

## Abstract

Agentic modeling aims to transform large language models (LLMs) into autonomous agents that can solve complex tasks through planning, reasoning, tool use, and multi-turn interaction with external environments. Despite substantial investment, open research in this area remains constrained by infrastructure and training gaps. Many high-performing agentic systems rely on proprietary codebases, models, or services, whereas open-source frameworks focus primarily on agent orchestration and harness design rather than improving agentic capabilities of LLMs through scalable model training. We present **Orchard**, an open-source framework for scalable agentic modeling. At its core is **Orchard Env**, a thin, Kubernetes-native environment service that provides reusable primitives for sandbox lifecycle management. Orchard Env is designed to operate across task domains, agent harnesses, and pipeline stages – including trajectory distillation, on-policy reinforcement learning (RL) rollouts, and evaluation. On top of Orchard Env, we build three agentic modeling recipes. **Orchard-SWE** targets software-engineering agents: we distill 107K trajectories from MiniMax-M2.5 and Qwen3.5-397B, introduce credit-assignment supervised fine-tuning (SFT) to learn from productive segments of unresolved trajectories, and apply Balanced Adaptive Rollout for sparse-reward RL. With Qwen3-30B-A3B-Thinking, Orchard-SWE achieves 64.3% on SWE-bench Verified after SFT and 67.5% after SFT+RL, setting a new state of the art among open-source models of comparable size. **Orchard-GUI** trains a 4B vision-language computer-use agent with only 0.4K distilled trajectories and 2.2K open-ended training tasks. It achieves success rates of 74.1%, 67.0%, and 64.0% on WebVoyager, Online-Mind2Web, and DeepShop, respectively (68.4% average), making it the strongest open-source model while remaining competitive with proprietary systems from OpenAI and Google Gemini. **Orchard-Claw** targets personal assistant agents for productivity workflows such as email, calendar, and daily tool-use tasks. Trained with only 0.2K synthetic tasks, it achieves 59.6% pass@3 on Claw-Eval, and improves to 73.9% pass@3 when paired with a stronger ZeroClaw harness. Collectively, these results demonstrate that a thin, open, harness-agnostic environment layer enables the reuse of agentic data, training recipes, and evaluation protocols across domains and harnesses. We release **Orchard** to accelerate agentic modeling research and drive innovation in the open-source AI community.

† First authors ‡ Second authors

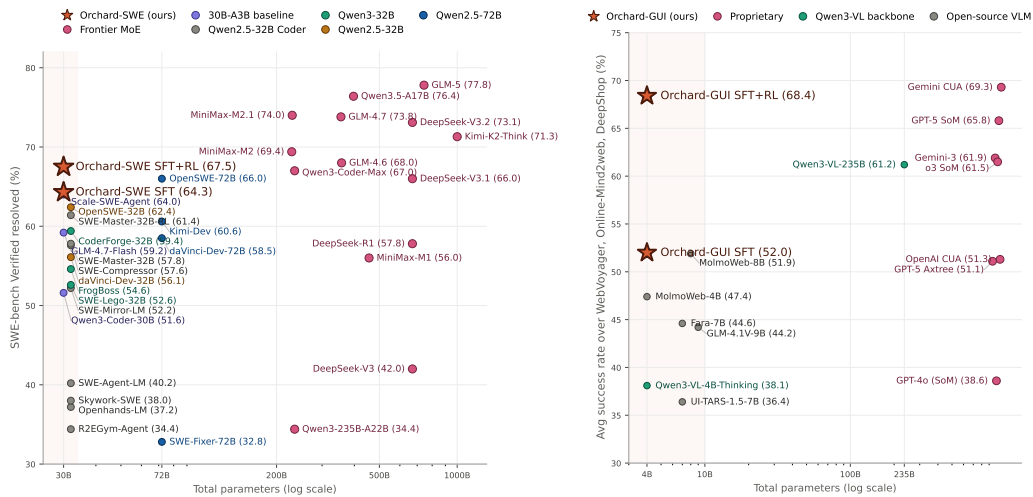


Figure 1: **Performance comparison.** *Left:* Orchard-SWE (30B) reaches 67.5% on SWE-bench Verified, approaching frontier MoE systems 10–30× larger. *Right:* Orchard-GUI (4B) achieves 68.4% average success across WebVoyager, Online-Mind2web, and DeepShop, making it the strongest open-source GUI agent while staying on par with proprietary systems from OpenAI and Google.

## 1 Introduction

Large language model (LLM) agents that interact with external environments over multiple turns have become a central paradigm for tasks ranging from software engineering (Jimenez et al., 2024; Yang et al., 2024) and web navigation (Zhou et al., 2024; Zhang et al., 2025; Ning et al., 2025) to general computer use (Xie et al., 2024; Hu et al., 2025). Training such agents—through supervised fine-tuning on expert trajectories or reinforcement learning from environment rewards—requires generating large numbers of rollout trajectories, each involving dozens of sequential interactions with a sandboxed execution environment.

As agentic training and evaluation scale to new domains and larger datasets, the need for open, scalable, affordable, and research-friendly infrastructure becomes increasingly acute. For example, generating a single trajectory for a software engineering task may involve cloning a repository, installing dependencies, applying code edits, and running a test suite—all within an isolated container that must be provisioned, managed, and cleaned up. At scale, thousands of such environments must run concurrently, each with distinct base images, resource requirements, and network isolation constraints.

We identify the environment layer as the foundational bottleneck. When it is closed or rigidly coupled to a particular training stack, every layer above it—training recipes, evaluation pipelines, trajectory collection—inherits those constraints and cannot be independently reproduced or reused. Existing systems make different choices about where to place environment management, each with trade-offs. Managed sandbox platforms such as E2B (E2B, 2024), Daytona (Daytona, 2025), and Modal (Modal Labs, 2024) provide convenient hosted runtimes, but give researchers limited control over infrastructure configuration, cost, and reproducibility. Vertically integrated training stacks such as ProRL Agent (Zhang et al., 2026a) and MegaFlow (Zhang et al., 2026b) include environment management as part of a larger rollout or training system, coupling it with inference scheduling, reward computation, and training-loop orchestration. Broader environment frameworks such as ROCK (Wang et al., 2026) provides rich platform functionality, but do not isolate the environment layer as a minimal service boundary. As a result, trajectory datasets, training recipes, and evaluation pipelines are often tied to a particular harness or infrastructure implementation, making them difficult to reproduce, compare, or reuse.

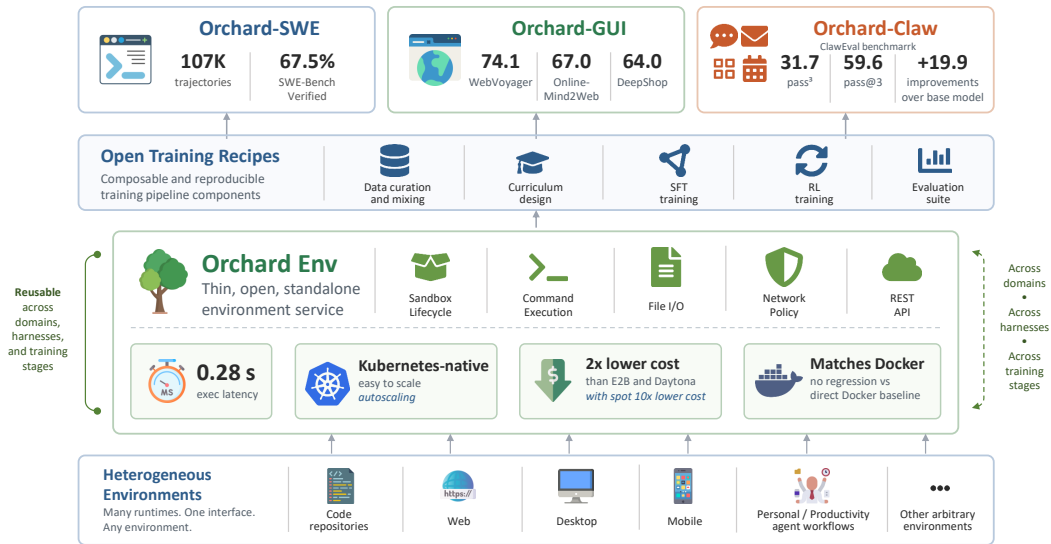


Figure 2: Overview of the Orchard framework. Orchard Env (center) is a thin, Kubernetes-native environment service that exposes generic primitives—sandbox lifecycle, command execution, file I/O, network policy, a REST API, and lightweight agent injection—and supports heterogeneous task environments (bottom row). Open training recipes (second row) compose with this service without coupling to it, and we instantiate the same stack in three domains (top row): **Orchard-SWE** (software engineering), **Orchard-GUI** (browser navigation), and **Orchard-Claw** (AI personal assistant); per-domain headline numbers are summarized inside each domain box.

We argue that the environment layer should instead be a *thin, standalone service* reusable along three axes: across (i) task domains, (ii) agent harnesses within a domain, and (iii) pipeline stages, including trajectory distillation, on-policy RL rollouts, and evaluation. When this boundary is clean, the layers above it become reusable as well: data can be collected under one harness and evaluated under another, SFT and RL recipes can share the same execution backend, and new domains can reuse the same infrastructure rather than rebuild it.

Therefore, we present **Orchard** (Figure 2), an open framework for scalable agentic modeling centered on a thin, reusable environment layer. Its core component, **Orchard Env**, is a Kubernetes-native service that exposes generic primitives—sandbox lifecycle management, command execution, file I/O, network policy, and a REST API—without coupling to any agent harness, trainer, inference backend, or task domain. Orchard Env scales through two key choices: runtime agent injection, which allows arbitrary task-specific Docker images to run separately, and direct routing of execution and file requests to sandbox Pod IPs, avoiding Kubernetes exec/WebSocket overhead. Together with network isolation, asynchronous lifecycle management, heartbeat cleanup, and watch-based readiness tracking, these mechanisms make Orchard Env broadly composable and practical for large-scale environment interaction. Empirically, it achieves 0.28s average command-execution latency, sustains a 1,000-sandbox stress test with 100% success, and substantially lowers estimated sandboxing cost relative to other alternatives.

On top of Orchard Env, we develop three agentic modeling (SFT+RL) recipes that compose with the environment service without tight coupling. These recipes handle trajectory collection, data curation, reward computation, and policy optimization. We instantiate them with backbones ranging from Qwen3-VL-4B-Thinking for browser agents to Qwen3-30B-A3B-Thinking (~3B active parameters) for software engineering and personal assistant agents.

Across three domains, the same environment abstraction supports diverse modalities, tool interfaces, agent harnesses, and reward mechanisms.

**Orchard-SWE.** For software engineering, Orchard-SWE targets two key bottlenecks of open SWE-agent training: limited supervision and sparse rewards. We curate 107K trajectories distilled from MiniMax-M2.5 and Qwen3.5-397B across SWE-rebench (Badertdinov et al., 2025), SWE-rebench V2 (Badertdinov et al., 2026), and Scale-SWE (Zhao et al., 2026), using both the OpenHands (Wang et al., 2025b) and mini-swe-agent (Yang et al., 2024) harnesses. Unlike most prior recipes, we retain not only resolved trajectories but also unresolved ones. We introduce *credit-assignment SFT*, which uses retrospective value estimation to extract productive rise segments from failed trajectories, converting partial progresses into supervised signals. We further apply *Balanced Adaptive Rollout (BAR)*, an online rollout-allocation method, to adaptively assemble reward-balanced trajectory groups for sparse-reward RL. With Qwen3-30B-A3B-Thinking, Orchard-SWE achieves 64.3% resolve rate on SWE-bench Verified after SFT and 67.5% after SFT+RL under mini-swe-agent, setting a new state of the art among open-source models of comparable size while remaining competitive with substantially larger models.

**Orchard-GUI.** For browser-based GUI agents, Orchard-GUI shows that the same environment service and recipe transfer beyond text-only computer use tasks. We train a 4B vision-language backbone with a generic ReAct-style (Yao et al., 2023) browser harness and evaluate on WebVoyager (He et al., 2024), Online-Mind2Web (Deng et al., 2023), and DeepShop (Lyu et al., 2025). After SFT+RL training, Orchard-GUI achieves success rates of 74.1%, 67.0%, and 64.0% on the three benchmarks, averaging 68.4% overall with the largest gains observed on long-horizon benchmarks, *i.e.*, Online-Mind2Web and DeepShop. This is a new open-source state of the art while remaining competitive with leading proprietary computer-use systems, despite using a 4B backbone model and only 2.6K training tasks. Remarkably, Orchard-GUI substantially outperforms both prior open-source agents and its 235B teacher model, suggesting that environment-grounded RL can improve model’s agentic capabilities beyond those of the teacher.

**Orchard-Claw.** For personal assistant agent, Orchard-Claw studies whether machine learned agent skills can transfer across different harnesses. We synthesize training tasks from Claw-Eval (Ye et al., 2026) seeds and ClawHub (OpenClaw, 2026) workflows, distill successful MiniMax-M2.5 trajectories, perform agentic training (SFT+RL) on Qwen3-30B-A3B-Thinking, and evaluate across harnesses, including a ReAct-style harness and the ZeroClaw (ZeroClaw Labs, 2026) harness. Orchard-Claw achieves 31.7% *pass*<sup>3</sup> and 59.6% *pass*@3 on Claw-Eval, significantly outperforming comparable-size open-source baselines despite using only 0.2K synthetic tasks. When paired with the stronger ZeroClaw harness at inference time, the same model improves further to 41.0% *pass*<sup>3</sup> and 73.9% *pass*@3.

Collectively, the results from the three agentic modeling recipes support the central claim of this study: the environment layer is not merely an infrastructural component, but the substrate governing the reusability of agentic modeling artifacts. A thin, open, harness-agnostic environment service enables trajectory data, SFT recipes, RL rollouts, and evaluation protocols to transfer across domains, agent harnesses, and pipeline stages. Orchard demonstrates that open-source agentic modeling can be scaled in a manner that is both cost-effective and reproducible, without coupling the environment to any single training stack. We release the full Orchard framework—environment service, training recipes, and trajectory datasets spanning software engineering, GUI navigation, and personal-assistant tool use—to facilitate open research in scalable agentic modeling.

## 2 Orchard Env

Scaling agentic training across domains and tasks places specific demands on the environment layer. We identify three core requirements for an environment service that can serve as a practical foundation for the research community:

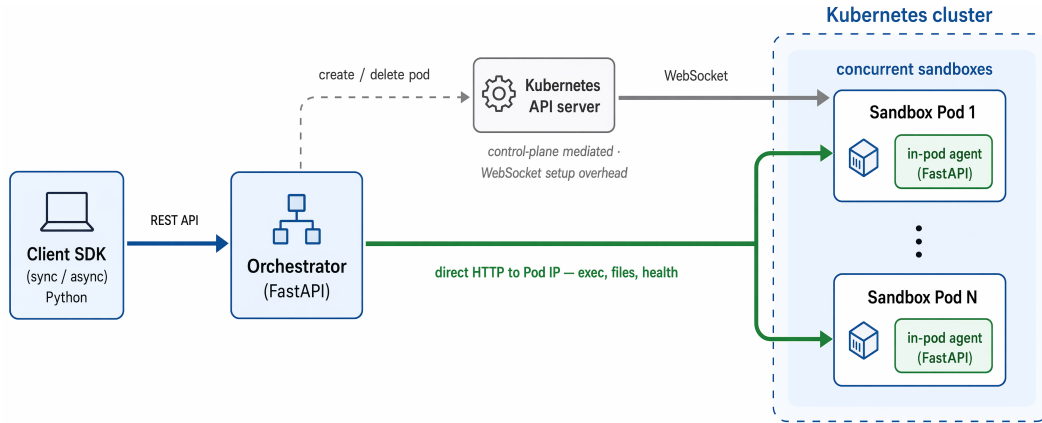


Figure 3: Orchard Env architecture. A Python client SDK (synchronous or asynchronous) issues REST calls to a FastAPI orchestrator, which manages sandbox lifecycle in a Kubernetes cluster. Pod creation and deletion (cold path) go through the Kubernetes API server, while all execution, file, and health requests (hot path) are dispatched directly to each sandbox pod’s in-pod agent via Pod IP, bypassing the API server and avoiding kubectl exec/WebSocket setup overhead.

1. **Thin, standalone service boundary.** Environment management should be isolated as a narrow service—decoupled from agent harness, model serving, and training orchestration—so that any combination of trainer, agent design, and task domain can compose with the same service.
2. **Low-cost image compatibility.** The service should support heterogeneous task environments and arbitrary Docker images at low adaptation cost.
3. **Accessible and cost-practical at scale.** The service should be deployable on any standard cloud infrastructure, making large-scale agentic training affordable and easy to adopt.

This section describes how Orchard Env realizes these requirements, presents its architecture and key design choices, and positions it among existing systems. More details can be found in Appendix A.

## 2.1 Architecture Overview

Orchard Env follows a three-layer architecture, as illustrated in Figure 3: a *client SDK* that provides synchronous and asynchronous Python interfaces, an *orchestrator* that manages sandbox lifecycle and scheduling, and a lightweight *in-pod agent* injected into each sandbox container.

This three-layer separation reflects three deliberate choices. First, the orchestrator and the in-pod agents are deployed and scaled independently: lifecycle decisions (creation, deletion, readiness) flow through the central orchestrator, while per-command execution traffic is dispatched directly to each sandbox’s in-pod agent, isolating control-plane operations from the latency-sensitive hot path. Second, the in-pod agent is *injected* into user-supplied images at runtime rather than baked in at build time, so that arbitrary task images integrate with no per-image modifications. Third, the entire stack runs on standard Kubernetes primitives (Pods, NetworkPolicy, Watch), inheriting open ecosystem tooling, multi-cloud portability, and cost optimizations such as cluster autoscaling and spot instances. We describe each layer in turn.

**Client SDK.** Orchard Env provides both synchronous (`SandboxClient`) and asynchronous (`AsyncSandboxClient`) Python clients. Sandboxes are created from user-specified Docker

images and expose methods for command execution, file upload/download, and patch application. Context managers provide automatic cleanup, and the SDK exposes heartbeat utilities for keeping long-lived sandboxes alive when desired. The SDK also includes configurable retry logic with exponential backoff for transient connection errors and service unavailable errors.

**Orchestrator.** The orchestrator is a FastAPI service deployed as a Kubernetes Deployment with multiple replicas. It exposes a REST API for sandbox lifecycle management and can delegate sandbox metadata tracking to an optional Redis backend across replicas. Key responsibilities include: **Sandbox provisioning:** Translating `POST /sandboxes` requests into Kubernetes Pod specifications, including init container configuration, resource limits, network policies, and readiness probes. **Readiness tracking:** A PodWatcher component maintains a persistent Kubernetes LIST+WATCH stream, caching pod state transitions and waking blocked clients when pods become ready. **Execution scheduling:** An ExecManager routes execution requests to the target sandbox’s in-pod agent via direct HTTP calls to the Pod IP, serializing concurrent requests to the same sandbox via per-sandbox locks. **Lifecycle management:** A background reconciliation loop detects and cleans up orphaned sandboxes (those whose heartbeat has expired or whose backing Pod has been evicted).

**In-Pod Agent.** The in-pod agent<sup>1</sup> is a lightweight FastAPI server that runs inside each sandbox container. It exposes endpoints for command execution (`/exec`), file upload, download, listing, and health checking. Commands are executed as subprocesses with configurable timeouts; on timeout, the entire process tree is killed via process group signal. The agent is reachable only through the sandbox pod’s internal cluster network endpoint, and its health endpoint serves as the Kubernetes readiness probe.

## 2.2 Comparison with Existing Systems

To position Orchard Env relative to existing systems, Table 1 compares environment and training infrastructure along four dimensions derived from the requirements above: whether an open-source server stack exists that researchers can self-host, whether the system is operated primarily as a managed service, whether it exposes a thin standalone environment service, and its relative cost at research scale. Concretely, we treat a system as a *thin env service* when (i) environment management is the system’s primary scope rather than a by-product of agent harness, training orchestration, or LLM serving; (ii) the environment layer presents a stable API—typically a small REST surface for sandbox lifecycle and command execution—that does not require the caller to adopt the system’s trainer, scheduler, or rollout abstractions; and (iii) that API is independent of the choice of agent harness, RL trainer, and inference backend, so the same service can back distillation, RL rollouts, and evaluation interchangeably. We highlight three aspects of Orchard Env’s positioning<sup>2</sup>:

**Thin env service vs. integrated and broad systems.** ProRL Agent (Zhang et al., 2026a) achieves an important decoupling—separating the rollout lifecycle from the RL trainer via an HTTP service—but its environment layer remains coupled with agent harness (via AgentHandler plugins), LLM inference routing, and evaluation logic within the same rollout server. MegaFlow (Zhang et al., 2026b) similarly embeds environment management within a larger training orchestration system. Modal (Modal Labs, 2024) is a different category altogether: it is a general serverless compute platform that offers flexible function and container execution, but it is not specialized as a thin environment service for agentic training, and its hosted control plane and per-second pricing are difficult to amortize across long-running RL training campaigns. ROCK (Wang et al., 2026) provides a broader environment framework with multiple protocols and richer platform components, targeting a wider scope than a thin service boundary. SkyPilot (Kim, 2025) provides open-source multi-cloud compute orchestration and can serve as the underlying infrastructure on which

<sup>1</sup>Here, “agent” refers to the sandbox-side execution service, not the LLM-based agents studied elsewhere in this paper.

<sup>2</sup>The comparison is based on public documentation and repositories as of April 2026.

Table 1: Environment and training systems for agentic training, based on public documentation as of April 2026. *Scope*: the system’s primary design scope. *Self-host.*: an open-source server/control-plane stack exists that researchers can deploy on their own infrastructure. *Mgd. default*: the system’s primary product is offered as a managed/hosted service. *Thin env service*: environment management is exposed as a narrow, standalone service boundary independent of agent harness, training loop, and inference backend (operational definition above). *Rel. cost*: normalized to Daytona for a 2-vCPU, 8-GiB sandbox; “—” indicates no publicly comparable pricing. See Appendix B for methodology.

System	Scope	Self-host.	Mgd. default	Thin env service	Rel. cost
E2B	managed sandbox	✓ <sup>†</sup>	✓	✓	1.0×
Daytona	managed sandbox	✓ <sup>†</sup>	✓	✓	1.0×
Modal	compute platform	✗	✓	✗	1.5×
SkyPilot	compute orch.	✓	✗	✗	—
ROCK	env. framework	✓	✗	✗	—
ProRL Agent	rollout infra.	✓	✗	✗	—
MegaFlow	training orch.	✗	✗	✗	—
<b>Orchard Env</b>	<b>env. service</b>	✓	✗	✓	<b>0.47×</b> *

\*0.10× with spot instances. <sup>†</sup>E2B and Daytona ship limited open-source server components, but their primary product is the hosted control plane and the *Rel. cost* column reflects that managed offering.

Orchard Env is deployed; the two are complementary rather than competing. E2B (E2B, 2024) and Daytona (Daytona, 2025), like Orchard Env, expose environment management as standalone sandbox services, but as managed products with hosted control planes and vendor-determined pricing. Orchard Env’s distinguishing technical choice is *agent injection*: a Kubernetes init container copies a self-contained execution agent into any user-provided Docker image at pod startup, avoiding the need to rebuild task images. This enables Orchard Env to support hundreds of heterogeneous task environments—such as the diverse images required by SWE-bench—without per-image modifications.

**Deployment portability.** Orchard Env targets researcher-controlled infrastructure: any standard Kubernetes environment—managed (AKS, EKS, GKE) or self-hosted—can run the full stack, with direct control over resource allocation, network policies, and autoscaling. This contrasts with HPC-oriented systems like ProRL Agent, which require access to institutional Slurm clusters and Singularity runtimes, limiting adoption to researchers at specific institutions.

**Cost as a consequence of design choices.** Table 2 compares estimated costs for 128 parallel sandboxes (2 vCPU, 8 GiB each) over 240 hours—a representative RL training workload. Because Orchard Env is self-hosted on standard Kubernetes, it naturally benefits from cloud-native cost optimization: ephemeral sandbox nodes can run on spot instances, and cluster autoscaling adjusts capacity to actual demand. This reduces cost to \$673 with spot instances—10× lower than managed alternatives like Daytona and E2B. Even at on-demand rates, Orchard Env (\$3,362) is less than half the cost of Daytona (\$7,078) and E2B (\$7,078). A detailed breakdown is provided in the Appendix B.

### 2.3 System Evaluation

For both agentic data generation and RL training, the most critical systems metric is environment interaction latency—it directly determines rollout throughput and GPU utilization. We evaluate Orchard Env on three axes: (i) execution latency relative to existing services, (ii) reliability under high concurrency, and (iii) functional equivalence to a direct Docker baseline in downstream agent evaluations. Unless noted otherwise, all measurements use a Kubernetes cluster of 8 nodes (each 32 vCPU, 128 GiB RAM) on commodity cloud VMs, with sandbox images pre-pulled on every node and each sandbox provisioned with 2 vCPU and 8 GiB RAM.

Table 2: Estimated cost for 128 parallel sandboxes over 240 hours (30,720 sandbox-hours). Target: 2 vCPU, 8 GiB RAM per sandbox. Costs normalized to Daytona. Prices from official rate cards as of April 2026; see Appendix B.

System	\$/sandbox-hr	Cost (\$)	vs. Daytona
Daytona	0.230	7,078	—
E2B	0.230	7,078	0%
Modal	0.335	10,305	+46%
MegaFlow <sup>†</sup>	0.150 (est.)	4,608 (est.)	−35%
Orchard Env (on-demand)	0.109	3,362	−53%
<b>Orchard Env (spot)</b>	<b>0.022</b>	<b>673</b>	<b>−90%</b>

<sup>†</sup>MegaFlow is not publicly priced; cells marked (*est.*) are estimated from reported infrastructure usage in Zhang et al. (2026b).

Table 3: System evaluation of Orchard Env. **Left:** Average command execution latency across environment services (lower is better; benchmark methodology follows Kim (2025)). **Right:** Agent pass rates (%) on Terminal-Bench 2.0 using Orchard Env vs. a direct Docker baseline, averaged over 3 runs per cell, confirming no regression from the environment service layer.

System	Avg. latency (s)	Model	Docker	Orchard Env
<b>Orchard Env</b>	<b>0.280</b>	GPT-4.1	34.1	<b>35.1</b>
SkyPilot Code Sandbox	0.284	MiniMax-M2.5	52.6	<b>54.4</b>
E2B	0.747	Qwen3-8B-thinking	7.0	<b>8.8</b>
Modal	2.046			

**Execution latency.** We compare average command execution latency across four environment services using the benchmark methodology of SkyPilot Code Sandbox (Kim, 2025), with the same benchmark setup across all platforms.

As shown in Table 3, Orchard Env achieves an average execution latency of 0.28 s, essentially matching SkyPilot Code Sandbox (0.284 s) and significantly outperforming E2B (0.747 s, 2.7× slower) and Modal (2.046 s, 7.3× slower). This validates Orchard Env’s direct Pod-IP communication design: by routing execution requests directly to the in-pod agent and bypassing the Kubernetes API server on the hot path, Orchard Env achieves latency comparable to optimized native runtimes while retaining the flexibility of a Kubernetes-based deployment.

**Reliability under concurrency.** To stress-test Orchard Env at scale, we ran 1,000 sandboxes in parallel through the full lifecycle: create → execute 4 commands → delete.

As shown in Table 4, Orchard Env achieved a **100%** success rate across all 1,000 sessions—no failures on creation, execution, or cleanup—with the entire test completing in 26 seconds end-to-end. Translating these end-to-end numbers, Orchard Env sustains roughly 154 commands per second across the full create → exec → delete lifecycle (4,000 commands across 1,000 sandboxes in 26 s), well above the throughput required by typical agentic distillation and RL workloads at this concurrency. These results confirm that Orchard Env’s architecture—watch-based readiness tracking, per-sandbox locking, and heartbeat-based cleanup—remains reliable at the concurrency levels required for large-scale agentic training.

Table 4: Stress test results for 1,000 parallel sandboxes through the full lifecycle (create → 4× exec → delete).

Metric	Value
Parallel sandboxes	1,000
Commands per sandbox	4
Success rate	<b>100%</b>
End-to-end time	26 s
Avg. create time	11.75 s
Avg. exec latency	0.28 s

**Functional equivalence to Docker.** Beyond infrastructure metrics, we verify that Orchard Env introduces no performance regression in downstream agent evaluations. We compare

Orchard Env against a direct Docker baseline on Terminal-Bench 2.0 (Merrill et al., 2026), using three models of varying capability; reported numbers are averaged over 3 independent runs per (model, backend) pair. As shown in Table 3 (right), Orchard Env matches Docker within run-to-run variance across all three models, with a marginal edge in every case (1–2 points). This confirms that the agent-injection mechanism and Orchard Env’s execution path introduce no observable overhead or interference in agent–environment interactions.

### 3 Orchard-SWE

This section presents Orchard-SWE, our instantiation of the Orchard training recipe for software engineering. We describe the problem setting, trajectory collection pipeline, two-stage training recipe, main results on SWE-bench Verified, and ablations that isolate key design choices.

#### 3.1 Problem Setting

**Task and evaluation.** We target the SWE-bench task formulation (Jimenez et al., 2024): given a GitHub issue description and a snapshot of the repository at the time the issue was filed, the agent must produce a code patch that resolves the issue. A solution is scored as correct if and only if it passes the full gold test suite associated with the ground-truth pull request. We use **SWE-bench Verified** (OpenAI, 2024)—a human-validated subset of 500 instances—as our primary evaluation benchmark. We also report auxiliary evaluations on **SWE-bench Multilingual** (Yang et al., 2025a) and **Terminal-Bench 2.0** (Merrill et al., 2026).

**Agent harness and tool interface.** The agent operates in a multi-turn ReAct-style loop (Yao et al., 2023): at each step, it produces a reasoning trace (Thought) and a tool invocation (Action), then observes the environment response before proceeding. The tool interface includes shell command execution, file viewing and editing, and patch submission. All environment interactions are routed through the Orchard Env service: each task instance runs in an isolated sandbox (2 vCPU, 8 GiB memory) provisioned from the task-specific Docker image, with Orchard Env’s agent-injection mechanism handling image heterogeneity transparently. A distinctive aspect of Orchard-SWE is that we collect trajectories using **two different agent harnesses**—the full-featured *OpenHands* (Wang et al., 2025b) framework and a lightweight *mini-swe-agent* (Yang et al., 2024)—and study how harness design affects both trajectory characteristics and downstream training outcomes (Section 3.6).

#### 3.2 Trajectory Collection and Curation

We construct the Orchard-SWE dataset through large-scale trajectory distillation from strong teacher models, followed by systematic filtering and curation. Table 5 summarizes the composition of the final dataset.

**Task sources.** We draw task instances from three sources: (1) **SWE-rebench** (Badertdinov et al., 2025), a large-scale collection of real-world GitHub issues with executable Docker-based test environments. We use its *filtered* subset, which applies quality and difficulty filters to retain instances that are both solvable and non-trivial, covering over 1,400 Python repositories. (2) **SWE-rebench V2** (Badertdinov et al., 2026), a language-agnostic extension of SWE-rebench that harvests more software engineering tasks. It provides over 32k containerized executable tasks spanning 20 programming languages<sup>3</sup> and more than 3.6k repositories, together with pre-built images. We reserve SWE-rebench V2 entirely for RL training. (3) **Scale-SWE** (Zhao et al., 2026), a complementary task source that constructs 100k task instances from real GitHub pull requests across 5.2k repositories. Each instance is packaged with a Docker image, a gold patch, and automatically generated test scripts, significantly expanding the diversity of repositories and issue types available for trajectory collection.

<sup>3</sup>In our experiments, we primarily use its Python tasks for consistency with the rest of our task pool.

Table 5: Composition of the Orchard-SWE training dataset. The corpus retains both resolved and unresolved trajectories: resolved trajectories provide direct imitation signal, while unresolved trajectories contribute partial-progress signal through credit-assignment SFT.

Source	Teacher	Resolved	Unresolved	Total	Harness
Scale-SWE	MiniMax-M2.5	33,527	20,591	54,118	mini-swe-agent
SWE-rebench	MiniMax-M2.5	13,364	10,099	23,463	mini-swe-agent
SWE-rebench	Qwen3.5-397B	15,545	1,846	17,391	mini-swe-agent
SWE-rebench	MiniMax-M2.5	12,213	0	12,213	OpenHands
<b>Total</b>		<b>74,649</b>	<b>32,536</b>	<b>107,185</b>	

**Multi-teacher trajectory generation.** We use multiple teacher models to increase the diversity of successful trajectories while keeping the downstream action space fixed. For each task instance, we sample five rollout trajectories through Orchard Env and retain all trajectories that successfully resolve the task. Our teacher pool includes **Qwen3.5-397B** (Qwen Team, 2026) and **MiniMax-M2.5 230B** (MiniMax, 2026). On SWE-rebench, we collect trajectories from both teachers under the mini-swe-agent and OpenHands harnesses. Empirically, MiniMax-M2.5 achieves a higher task pass rate, while Qwen3.5-397B occasionally emits tool calls that are not defined in the OpenHands tool interface. Based on these observations, we use MiniMax-M2.5 as the sole teacher for Scale-SWE, where rollout efficiency and stability become more important due to the larger number of instances. In all cases, teachers interact through the same sandboxed tool interface used at evaluation time, ensuring that collected trajectories remain faithful to the downstream action space.

**Harness selection.** We collect trajectories with two agent harnesses: **OpenHands** (Wang et al., 2025b) and **mini-swe-agent** (Yang et al., 2024). On SWE-rebench, we use both harnesses so that trajectory collection covers a broader range of interaction styles and tool-use patterns. For the OpenHands runs, we follow its standard SWE-bench tool configuration.<sup>4</sup> For Scale-SWE, we use only mini-swe-agent, a lightweight harness with a minimal tool set (bash execution, file editing, submission), since we did not observe a meaningful performance gap relative to OpenHands on this source and the lighter harness is more practical for large-scale rollout collection. This dual-harness setup also lets us analyze how harness choice affects downstream training outcomes (Section 3.6).

**Filtering and curation.** Unlike most prior work, which retains only successful (resolved) trajectories for SFT, we keep **both resolved and unresolved trajectories** in the training corpus. Resolved trajectories provide standard imitation signal; unresolved trajectories are curated via temporal-difference credit assignment to extract continuous *rise segments*—spans where the trajectory is making progress—which contribute partial-progress supervision (formalized in Section 3.3). We additionally apply the following quality filters: (1) trajectories exceeding 64K tokens are pruned to ensure training stability; (2) trajectories containing tool calls not defined in the harness’s tool interface (primarily observed with Qwen3.5-397B) are discarded; (3) trajectories with syntactically invalid or unparsable actions are removed.

After filtering, the Orchard-SWE dataset comprises **107K trajectories** (74.6K resolved, 32.5K unresolved) spanning **19,287 unique task instances**, with an average of 47.5 interaction turns and approximately 21K tokens per trajectory. We release the full dataset, including both resolved and unresolved trajectories, as an open-source artifact.

### 3.3 Training Recipe

Our training recipe follows a two-stage pipeline: Supervised Fine-Tuning (SFT) on teacher-distilled trajectories, followed by Reinforcement Learning (RL) with environment-grounded rewards. Both stages use Orchard Env as the execution backend.

<sup>4</sup><https://github.com/OpenHands/benchmarks/tree/main/benchmarks/swebench>

### 3.3.1 Stage 1: Supervised Fine-Tuning with credit assignment

We initialize from the base backbone and fine-tune on the curated teacher trajectories. Each training example pairs the issue description and repository context with the full multi-turn interaction trace, serialized as a sequence of observations and actions. Following standard practice for long-horizon agent training, we apply multi-turn masking so that environment observations are excluded from the loss and the model is trained only to predict its reasoning traces and actions.

A distinguishing feature of our SFT stage is the use of **credit-assignment SFT**, which incorporates not only the 74.6K resolved trajectories but also a curated subset of unresolved trajectories where partial progress is identifiable. We instantiate credit assignment as a lightweight LLM-based variant of temporal-difference value estimation, formulated as follows.

**Retrospective value estimation.** For each unresolved trajectory  $\tau = (s_0, a_0, s_1, \dots, s_T)$ , we use the trajectory’s own teacher model as a zero-shot retrospective value function. The teacher is shown the full trajectory together with the gold test outcome and is asked to estimate, at each step  $t$ , the probability that the agent will resolve the issue given the history  $h_t = (s_0, a_0, \dots, s_t)$ :

$$V(s_t) = \mathbb{P}(\text{resolve} \mid h_t, \text{outcome}). \quad (1)$$

The teacher annotates a sparse set of key steps and the remaining values are interpolated, yielding a per-step value curve  $V(s_0), \dots, V(s_T)$ . Because the judgment is retrospective and outcome-conditioned, the value curve calibrates well to actual progress: across our annotated trajectories, the curve is inverted-U in 98.9% of cases, peaking during exploration and decaying near the failed submission. The exact prompt format is shown below.

#### Value-Estimation Prompt

```
<|im_start|>system
You are an expert software engineer performing credit assignment on a coding agent's trajectory.

This trajectory FAILED -- the agent's final patch did not pass the required tests. Your task is to
estimate P(resolve), the probability the trajectory will ultimately succeed, at each step
checkpoint. This is retrospective temporal value estimation.

Reason BACKWARD from the known failure:
1. Read the test outcome to understand exactly what went wrong.
2. Identify the critical mistake -- the step(s) where the agent went wrong.
3. Assign probabilities that REFLECT this knowledge. P should rise during correct exploration but MUST
   DROP at or before the critical mistake and stay low through submission.

Calibration rules:
- Start at a base rate of 0.3-0.5.
- P MUST DROP when the agent makes the critical error.
- P at the final step should be BELOW 0.2 (we know it failed).
- Do NOT give monotonically increasing P.

Output format: a JSON array, one entry per step. Nothing else.
[
  {"step": 0, "p_resolve": 0.40, "reasoning": "initial exploration, base rate"},
  {"step": 1, "p_resolve": 0.50, "reasoning": "found relevant source file"},
  ...
  {"step": K, "p_resolve": 0.25, "reasoning": "edit is incomplete, missed X"},
  ...
  {"step": N, "p_resolve": 0.10, "reasoning": "submitted without fixing the core issue"}
]
<|im_end|>

<|im_start|>user
## Issue Description
[issue text from the original GitHub bug report]

## Outcome (this trajectory FAILED)
Tests that should have been fixed but WERE NOT:
- FAILED: tests/test_module.py::test_specific_case
... and 3 more
Tests that were successfully fixed: 2
Previously passing tests that REGRESSED:
```

```

- REGRESSED: tests/test_other.py::test_unrelated

## Agent Trajectory

[STEP 0]
Thinking: I need to first explore the repository structure...
Action: bash(ls -la)
Result: [directory listing, truncated to 5K chars]

[STEP 1]
Thinking: Let me find the file that defines the buggy class...
Action: bash(grep -rn "class Foo" src/)
Result: src/module.py:42:class Foo: ...

[... additional steps elided ...]

[STEP N-1]
Thinking: I'll submit this patch now.
Action: submit()
Result: Patch submitted.

## Total steps: N
Provide P(resolve) estimates for steps 0 through N-1.
<|im_end|>
    
```

**Rise-segment extraction.** We define the per-step credit as the temporal-difference shift in estimated success probability,

$$c_t = V(s_{t+1}) - V(s_t), \tag{2}$$

and extract *rise segments*: maximal contiguous subsequences  $[t_i, t_j]$  over which the agent makes positive progress, i.e.  $c_t \geq \epsilon$  for all  $t \in [t_i, t_j-1]$  (with a small threshold  $\epsilon$  to filter annotation noise).<sup>5</sup> Rise segments are typically short (median  $\sim 2$  steps before merging with surrounding context) but capture the productive parts of an otherwise unsuccessful trajectory—repository navigation, file localization, and partial root-cause analysis.

**SFT objective.** We train with a standard next-token cross-entropy loss on action tokens, masking environment observations as well as action tokens that fall outside any rise segment:

$$\mathcal{L}_{\text{SFT}} = - \sum_{t \in \mathcal{S}(\tau)} \log \pi_{\theta}(a_t | h_t),$$

where  $\mathcal{S}(\tau)$  is the set of action tokens contributing to the loss for trajectory  $\tau$ . For resolved trajectories,  $\mathcal{S}(\tau)$  contains all action tokens (equivalent to a single segment spanning the entire trace, since the terminal value is 1). For unresolved trajectories,  $\mathcal{S}(\tau)$  is restricted to action tokens inside the extracted rise segments, with the preceding history retained as context. After this construction, the 32,536 unresolved trajectories yield exploration-focused supervision that complements the full solve-and-submit traces from the resolved set.

We train Qwen3-30B-A3B-Thinking (Qwen Team, 2025) with slime (Zhu et al., 2025) for five epochs with a global batch size of 128 and a 64K context window, using a cosine-decayed learning rate from  $10^{-5}$  to  $10^{-6}$ . Although training uses a maximum sequence length of 64K, we extend the context limit to 128K at inference time to accommodate longer repository contexts and interaction histories.

### 3.3.2 Stage 2: Reinforcement Learning with Balanced Adaptive Rollout (BAR)

Starting from the SFT checkpoint, we apply RL to improve the model’s ability to recover from errors and explore alternative solution paths. The reward signal is binary and environment-grounded: a trajectory receives a reward of +1 if the final patch passes the gold test suite in the sandbox, and -1 otherwise. Orchard Env’s fast execution latency (0.28 s per command; Section 2.3) is critical at this stage, as each RL rollout requires dozens of environment interactions, and training throughput scales directly with sandbox responsiveness.

<sup>5</sup>We use  $\epsilon = 0.05$ ; see Appendix for sensitivity analysis.

**System Design and Component Orchestration.** Our RL system builds on slime (Zhu et al., 2025) post-training framework, extending its Ray-based, Megatron-LM-backed training and SGLang-based inference architecture for asynchronous agentic RL. The system is organized as four loosely coupled services that communicate through Ray actor handles and HTTP endpoints, so that each component can be scaled, replaced, or restarted independently:

- **Policy Trainer.** A Megatron-LM-based distributed trainer, sharded with tensor, pipeline, expert, and context parallelism, that owns the trainable parameters and performs the optimization step using advantage-weighted policy gradients.
- **Rollout Inference Service.** An SGLang-based inference service, fronted by a request router, that serves the latest policy snapshot. It supports KV-cache reuse, deterministic sampling seeds, and per-token log-probability extraction.
- **Sandboxed Execution Service.** A sandbox runtime initialized from Orchard Env. Each agent trajectory is bound to an isolated sandbox in which bash commands and unit-test suites can be executed safely through Orchard Env.
- **Agentic Loop Driver.** A per-sample asynchronous coroutine that orchestrates the tool-calling interaction between the inference service and the execution sandbox. At each step, it tokenizes the running message history with the chat template and registered bash tool schema, queries the inference service, parses the structured tool call from the assistant message, executes the call inside the sandbox, and appends the resulting observation as a tool message. The loop terminates when the agent submits a patch, exceeds a step, wall-clock, or token budget, or is aborted due to rare sandbox failures.

The orchestration is asynchronous and pipelined. While the trainer is updating weights for rollout  $k$ , the rollout manager has already dispatched generation for rollout  $k+1$ . A central rollout manager also implements robust handling of partially-failed trajectories so a single sandbox failure cannot crash the optimizer.

Our Agentic Loop Driver is hardened with a multi-layer timeout and retry hierarchy (sandbox creation, LLM inference, observation execution, sandbox shutdown, total reward evaluation) that bounds tail latency without sacrificing end-to-end reliability. When a sandbox crash is attributed to resource exhaustion, CPU and memory allocations are automatically escalated on retry, and small random jitter is injected before sandbox creation to prevent thundering-herd effects when hundreds of concurrent trajectories spin up simultaneously. Loss masking is applied in token space so that gradients flow only through assistant-generated tokens; tool-result tokens are explicitly masked out, which makes multi-turn agentic RL well-defined under a standard LM cross-entropy objective.

### 3.3.3 *Balanced Adaptive Rollout (BAR)*

For challenging agentic tasks such as SWE, the standard fixed- $N$  group rollout used by GRPO (Shao et al., 2024) has two major problems:

- **Wasted compute.** When the policy is competent on a prompt, all  $N$  trajectories tend to succeed; when it is not, all  $N$  tend to fail. In both regimes the resulting group has zero reward variance, contributes a zero advantage to every token, and is silently discarded — yet we have already paid for  $N$  long, environment-bound trajectories.
- **Group-imbalance noise.** When the success rate of a prompt is far from 0.5, even a "non-degenerate" group is dominated by either positives or negatives, and the resulting GRPO advantages are noisy and biased toward whichever class is over-represented.

We address both issues with *Balanced Adaptive Rollout (BAR)*, a progressive, group-aware rollout algorithm. Unlike prior dynamic sampling and difficulty-filtering methods that discard zero-variance prompts (Yu et al., 2025; Le et al., 2025), pre-filter prompts using historical success rates (Bae et al., 2026; Zheng et al., 2025b), or post-hoc down-sample oversized rollout sets (Xu et al., 2025; Shang et al., 2025; Zhang et al., 2026c), our method performs

---

**Algorithm 1** Balanced Adaptive Rollout (BAR) for a single prompt
 

---

**Require:** Prompt  $x$ ; policy  $\pi_\theta$ ; reward function  $R$ ; environment factory  $\mathcal{E}$   
**Require:** Training group size  $N$ ; max budget  $N_{\max}$ ; stride  $s$  (with  $s \mid N_{\max}$ )  
**Require:** Positive-fraction interval  $[\rho_{\min}, \rho_{\max}]$ ; ideal ratio  $\rho^* = (\rho_{\min} + \rho_{\max})/2$   
**Ensure:** A training group  $\mathcal{G} \subset \mathcal{T}$  with  $|\mathcal{G}| = N$

- 1:  $\mathcal{T} \leftarrow \emptyset$  ▷ pool of completed trajectories
- 2: **for**  $t = 0, s, 2s, \dots, N_{\max} - s$  **do**
- 3:    $\mathcal{B}_t \leftarrow \{ \tau_i \sim \pi_\theta(\cdot \mid x; \mathcal{E}) : i = 1, \dots, s \}$  ▷ generate stride in parallel
- 4:    $r_i \leftarrow R(\tau_i)$  for each  $\tau_i \in \mathcal{B}_t$
- 5:    $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{B}_t$
- 6:    $(\mathcal{G}, \text{ok}) \leftarrow \text{TRYASSEMBLE}(\mathcal{T}, N, \rho_{\min}, \rho_{\max}, \rho^*)$
- 7:   **if** ok **then**
- 8:     **return**  $\mathcal{G}$  ▷ early-stop: balanced group found
- 9:   **end if**
- 10: **end for**
- 11:  $(\mathcal{G}, \text{ok}) \leftarrow \text{TRYASSEMBLE}(\mathcal{T}, N, 0, 1, \rho^*)$  ▷ relaxed fallback
- 12: **if** ok **then return**  $\mathcal{G}$
- 13: **end if**
- 14: **return**  $\text{TOPRANKED}(\mathcal{T}, N)$  ▷ best-effort fallback

- 15: **procedure**  $\text{TRYASSEMBLE}(\mathcal{T}, N, \rho_{\min}, \rho_{\max}, \rho^*)$
- 16:   Partition  $\mathcal{T}$  into  $\mathcal{T}_+ = \{ \tau : R(\tau) > 0, \tau \text{ usable} \}$ ,  $\mathcal{T}_- = \{ \tau : R(\tau) \leq 0, \tau \text{ usable} \}$ , and put the rest (aborted / time-exceeded) into a backfill pile.
- 17:   Sort  $\mathcal{T}_+$  and  $\mathcal{T}_-$  by (status, response length) ascending ▷ prefer completed and concise
- 18:   **if**  $|\mathcal{T}_+| + |\mathcal{T}_-| < N$  **then return**  $(\perp, \text{false})$
- 19:   **end if**
- 20:    $n^* \leftarrow \text{round}(\rho^* \cdot N)$ ;  $n_{\min} \leftarrow \lceil \rho_{\min} \cdot N \rceil$ ;  $n_{\max} \leftarrow \lfloor \rho_{\max} \cdot N \rfloor$
- 21:   **for**  $n_+ \in \{n_{\min}, \dots, n_{\max}\}$  sorted by  $|n_+ - n^*|$  **do**
- 22:      $n_- \leftarrow N - n_+$
- 23:     **if**  $n_+ \leq |\mathcal{T}_+|$  **and**  $n_- \leq |\mathcal{T}_-|$  **then**
- 24:       **return**  $(\mathcal{T}_+[1:n_+] \cup \mathcal{T}_-[1:n_-], \text{true})$
- 25:     **end if**
- 26:   **end for**
- 27:   **return**  $(\perp, \text{false})$
- 28: **end procedure**

---

online, per-prompt, stride-based group assembly. It adaptively continues generation only until it can construct a fixed-size training group whose positive-reward fraction lies in a target interval, while accounting for trajectory status, truncation, sandbox failures, and length. This makes the rollout scheduler directly compatible with group-relative estimators in long-horizon agentic environments.

For each prompt we set three quantities: a *training group size*  $N$  (the number of trajectories the optimiser will actually consume), a *maximum budget*  $N_{\max} > N$  (an upper bound on how many trajectories we are willing to generate), and a *stride*  $s$  (the size of an incremental generation batch). We additionally specify a target positive-reward fraction interval  $[\rho_{\min}, \rho_{\max}]$ , with ideal ratio  $\rho^* = (\rho_{\min} + \rho_{\max})/2$ . The algorithm proceeds as follows. We generate  $s$  trajectories, score them with the reward model, and partition the pool of completed trajectories into a *positive* set (trajectories with reward  $> 0$ ) and a *negative* set (otherwise), after first moving aborted or truncated trajectories to a backfill pile. We then attempt to assemble a training group of exactly  $N$  trajectories whose positive fraction lies in  $[\rho_{\min}, \rho_{\max}]$  and that is closest to the ideal ratio  $\rho^*$ . Within each class, trajectories are ranked by terminal status (completed  $>$  truncated  $>$  aborted)<sup>6</sup>, so as to prefer succinct, well-terminated trajectories. If a feasible group exists, we early-stop and return it; otherwise we generate another stride and retry. If after  $N_{\max}$  trajectories no balanced group can be built, we fall back to a relaxed selection (any positive fraction in  $(0, 1)$ ), padding with the best backfill trajectories as needed.

---

<sup>6</sup>Other ranking criteria could also be used here, for example ranking by trajectory length, model likelihood, diversity, or estimated uncertainty, etc.

Table 6: Performance on SWE-rebench v2. Baseline results with <sup>†</sup> are from [Badertdinov et al. \(2026\)](#), where models were evaluated on a 60-task Python subset. Orchard-SWE is evaluated on the full Python subset. All models are evaluated using the mini-swe-agent harness.

Model	pass@1	pass@3
Claude Opus-4.5 <sup>†</sup>	36.11%	36.67%
GLM-4.7 <sup>†</sup>	27.22%	31.67%
MiniMax-M2.1 <sup>†</sup>	26.11%	31.67%
Gemini <sup>†</sup>	25.56%	33.33%
DeepSeek-V3.2 <sup>†</sup>	23.33%	31.67%
GPT-5.2 <sup>†</sup>	20.56%	23.33%
gpt-oss-120b <sup>†</sup>	8.89%	16.67%
Orchard-SWE	22.36%	27.94%

BAR therefore behaves as an anytime, self-pacing rollout schedule: 1) Easy or already-mastered prompts (where the first stride is overwhelmingly positive) trigger no further generation — the prompt is either filtered out or returned with the minimum positives needed to satisfy the lower bound; 2) Hard prompts (where positives are rare) keep generating until either enough positives are discovered or the budget  $N_{\max}$  is exhausted; 3) Well-balanced prompts finish near the first stride and yield maximally informative gradients. The result increases the average information density of every gradient batch. Importantly, BAR composes cleanly with GRPO, GSPO ([Zheng et al., 2025a](#)), and any other group-relative advantage estimator, because the contract it must satisfy is simply “return a list of  $N$  trajectories per prompt”.

**Final Group Filtering.** Because rewards are produced by a noisy, real environment (sandbox creation can time out, containers can be evicted, the LLM can hit its token budget mid-step), some trajectories carry no usable learning signal even within an otherwise valid group. We therefore plug a group-level filter into the rollout loop, evaluated after reward computation and before the group is admitted into the training batch. A dropped group is simply replenished from over-sampled prompts, which decouples the training batch size from the generation batch size and preserves gradient quality.

Filtering and the Balanced Adaptive Rollout (BAR) are designed to work jointly: BAR maximises the probability that a generated group satisfies the filter on the first try, and the filter provides a hard correctness guarantee on whatever BAR returns. Together they implement a form of reward-aware curriculum that is performed online, at every gradient step. Together these components form a fault-tolerant, throughput-optimised pipeline for end-to-end RL on long-horizon, sandbox-grounded agentic tasks, with Balanced Adaptive Rollout turning a fixed-batch rollout into a self-pacing, information-dense one. The final estimated advantage for rollout trajectory  $o_i$  within a group of  $N$  samples is normalized using the group rewards  $A_{i,t} = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_N\})}{\text{std}(\{r_1, r_2, \dots, r_N\})}$ , where  $r_i$  denotes the reward assigned to rollout trajectory  $o_i$ . RL train for at most 150 steps with a global batch size of 128 and a 64K context window, using a cosine-decayed learning rate from  $10^{-6}$ . We apply the rollout batch size 16 with a training group size of  $N = 8$ , maximum budget  $N_{\max} = 16$  and stride  $s = 16$  to encourage parallel rollout. The target positive-reward fraction interval used is  $[\rho_{\min}, \rho_{\max}] = [0.375, 0.625]$ .

**Data Selection.** For RL training, we construct a task pool using all Python subset data of SWE-rebench V2 and Scale-SWE data that were not used during SFT. We first run the initial SFT model on each candidate task with 8 rollouts to get its initial pass rate. Table 6 reports the performance of initial Orchard-SWE SFT checkpoint on SWE-rebench V2 alongside baseline results. Orchard-SWE SFT achieves 22.36% pass@1 and 27.94% pass@3 on the full Python subset. We then retain only tasks with pass rate  $0 < \hat{p} \leq 0.5$ , filtering out tasks that are either too difficult to provide a reliable learning signal or already too easy for the SFT model. This selection is particularly important for SWE-rebench V2, which is highly

Table 7: Resolve rates (%) on SWE-bench Verified. We compare Orchard-SWE against open-source SWE-agent recipes, organized by base-model family. Within each section, rows are sorted by reported resolve rate. Orchard-SWE rows are bolded. For broader context, frontier proprietary systems (Claude Opus 4.5, GPT-5.2, Gemini 3, etc.) reach 71–77% on this benchmark.

System	Base Model	Harness	Resolved (%)
<i>Open-Source Methods: Qwen 2.5 32B Coder Series</i>			
R2EGym-Agent (Jain et al., 2025)	Qwen2.5-32B-Coder-Base	R2E-Gym	34.4
Openhands-LM (Wang et al., 2025b)	Qwen2.5-Coder-32B-Inst.	OpenHands	37.2
Skywork-SWE (Zeng et al., 2025)	Qwen2.5-Coder-32B-Inst.	OpenHands	38.0
SWE-Agent-LM (Yang et al., 2025a)	Qwen2.5-Coder-32B-Inst.	SWE-Agent	40.2
SWE-Mirror-LM (Wang et al., 2025a)	Qwen2.5-Coder-32B-Inst.	MOpenHands	52.2
SWE-Compressor (Liu et al., 2025)	Qwen2.5-32B-Base	OpenHands	57.6
SWE-Master-32B (Song et al., 2026)	Qwen2.5-Coder-32B-Inst.	R2E-Gym	57.8
SWE-Master-32B-RL (Song et al., 2026)	Qwen2.5-Coder-32B-Inst.	R2E-Gym	61.4
<i>Open-Source Methods: Qwen 3 32B Series</i>			
FrogBoss (Sonwane et al., 2025)	Qwen3-32B	R2E-Gym	54.6
SWE-Lego-Qwen3-32B (Tao et al., 2026)	Qwen3-32B	OpenHands	52.6
CoderForge-32B (Ariyak et al., 2026)	Qwen3-32B	OpenHands	59.4
<i>Open-Source Methods: Qwen 2.5 32B Series</i>			
daVinci-Dev-32B (Zeng et al., 2026)	Qwen2.5-32B-Base	SWE-Agent	56.1
OpenSWE-32B (Fu et al., 2026)	Qwen2.5-32B-Base	OpenHands	59.8
OpenSWE-32B (Fu et al., 2026)	Qwen2.5-32B-Base	SWE-Agent	62.4
<i>Open-Source Methods: Qwen 2.5 72B Series</i>			
SWE-Fixer-72B (Xie et al., 2025)	Qwen2.5-72B-Base	Agentless	32.8
daVinci-Dev-72B (Zeng et al., 2026)	Qwen2.5-72B-Base	SWE-Agent	58.5
Kimi-Dev (Yang et al., 2025b)	Qwen2.5-72B-Base	Agentless	60.6
OpenSWE-72B (Fu et al., 2026)	Qwen2.5-72B-Base	OpenHands	65.0
OpenSWE-72B (Fu et al., 2026)	Qwen2.5-72B-Base	SWE-Agent	66.0
<i>Same-size baselines and our model (30B-A3B; ~3B active)</i>			
Qwen3-30B-A3B-Instruct	—	OpenHands	22.0
Qwen3-Coder-30B-A3B-Instruct	—	OpenHands	51.6
GLM-4.7-Flash-30A3B (Team et al., 2025)	—	—	59.2
Scale-SWE-Agent (Zhao et al., 2026)	Qwen3-30B-A3B-Instruct	OpenHands	64.0
<b>Orchard-SWE (SFT)</b>	Qwen3-30B-A3B-Thinking	mini-swe-agent	<b>64.3</b>
<b>Orchard-SWE (SFT)</b>	Qwen3-30B-A3B-Thinking	OpenHands	62.1
<b>Orchard-SWE (SFT+RL)</b>	Qwen3-30B-A3B-Thinking	mini-swe-agent	<b>67.5</b>

challenging: as shown in the table, even state-of-the-art models achieve 20%-40% pass rates on this dataset. After filtering, the final RL training set contains approximately 2k instances. We apply mini-swe-agent as the harness for RL training.

### 3.4 Main Results

Table 7 compares Orchard-SWE with open-source SWE-agent recipes on SWE-bench Verified, organized by base-model family. Orchard-SWE achieves **64.3%** after SFT and **67.5%** with the full SFT+RL recipe, using only  $\sim 3B$  active parameters: its Qwen3-30B-A3B MoE backbone activates 3B of 30B total parameters at inference. At this active-parameter budget, Orchard-SWE is competitive with or exceeds dense open baselines that activate an order of magnitude more parameters: it surpasses every Qwen 2.5 32B and Qwen 3 32B open-source recipe in the table—including OpenSWE-32B (62.4% with SWE-Agent), SWE-Master-32B-RL (61.4%), CoderForge-32B (59.4%), and SWE-Mirror-LM (52.2%)—and surpasses the strongest dense 72B systems (Kimi-Dev 60.6%, OpenSWE-72B 65.0–66.0%).

Table 8: Generalization across harnesses and task distributions. Resolve rate (%) is reported for each system under matched conditions. SWE-V = SWE-bench Verified; SWE-M = SWE-bench Multilingual; T-Bench 2.0 = Terminal-Bench 2.0. \* denotes numbers reported in the original paper; unmarked entries are our own evaluations under matched conditions;  $\times$  indicates the system produced malformed tool calls under that harness, yielding no valid resolve rate.

System	OpenHands	mini-swe-agent		Kimi-CLI (unseen)	
	SWE-V	SWE-V	SWE-M	SWE-V	T-Bench 2.0
Scale-SWE	64.0*	$\times$	$\times$	$\times$	$\times$
OpenSWE-32B	62.4*	54.9	28.7	3.6	0.0
Orchard-SWE	62.1	<b>64.3</b>	<b>51.0</b>	<b>45.0</b>	<b>20.1</b>

**Same-size family lift.** The cleanest apples-to-apples comparison is within the 30B-A3B family. Orchard-SWE improves over Qwen3-30B-A3B-Instruct by a **45.5%** absolute lift on SWE-bench Verified (22.0%  $\rightarrow$  64.3% after SFT,  $\rightarrow$  67.5% after SFT+RL), and also exceeds the code-specialized Qwen3-Coder-30B-A3B-Instruct (51.6%) and the broader-distillation GLM-4.7-Flash-30A3B (59.2%) by wide margins. The closest competitor at comparable scale is Scale-SWE-Agent (64.0%), built on the same backbone family. Orchard-SWE matches it under SFT and outperforms it under SFT+RL. This isolates the effect of the Orchard-SWE recipe itself—multi-teacher distillation, multi-harness collection, credit-assignment SFT, and RL—rather than any advantage from the underlying base model.

The same Orchard-SWE (SFT) checkpoint reaches 64.3% under the mini-swe-agent harness but 62.1% under OpenHands, indicating that single-condition leaderboard numbers are sensitive to harness choice. This sensitivity becomes the central empirical question of Section 3.5: when we evaluate our Orchard-SWE alongside the closest open recipes (Scale-SWE, OpenSWE-32B) across multiple harnesses and tasks, the differences become much more pronounced—Orchard-SWE retains capability on unseen harnesses and out-of-distribution tasks, while other models collapse.

### 3.5 Generalization to Unseen Harnesses and Tasks

On SWE-bench Verified, Scale-SWE (64.0%) and OpenSWE-32B (62.4%) report similar resolve rates. Single-benchmark scores can mask large differences in how well an agent generalizes. We therefore evaluate each model across three harnesses—OpenHands, mini-swe-agent, and Kimi-CLI (Moonshot AI)—and three different tasks: SWE-bench Verified, SWE-bench Multilingual (Yang et al., 2025a), and Terminal-Bench 2.0 (Merrill et al., 2026). Kimi-CLI was not used during training data collection by any of the three systems, making it an unseen harness in this study.

**Harness lock-in is severe in single-harness training.** We find Scale-SWE (Zhao et al., 2026) produces invalid outputs under any harness other than its native one, yielding no measurable resolve rate. OpenSWE-32B (Fu et al., 2026) remains structurally valid but degrades sharply: from 62.4% on its native OpenHands to 54.9% on mini-swe-agent ( $-7.5$  pt) and 3.6% on Kimi-CLI ( $-58.8$  pt). Orchard-SWE, in contrast, holds within a narrow band of 45.0–64.3% across all three harnesses, with the worst-case drop bounded at 19.3 points relative to its own best. The two failure modes observed in Scale-SWE and OpenSWE-32B (catastrophic format failure and degraded resolve rate) have the same root cause - a model trained under a single harness has not learned harness-agnostic SWE skills. This pattern is exactly what our cross-harness ablation predicted in a controlled experiment (Section 3.6, Table 10); here we see the same failure mode play out in two independently developed open recipes.

**Cross-distribution generalization.** On SWE-bench Multilingual under the mini-swe-agent harness, Orchard-SWE drops from 64.3% (Verified) to 51.0% ( $-13.3$  absolute,  $-20.7%$  relative). OpenSWE-32B drops from 54.9% to 28.7% ( $-26.2$  absolute,  $-47.7%$  relative). Orchard’s

relative drop is roughly half, indicating that multi-teacher distillation across SWE-rebench and Scale-SWE provides broader exposure to repositories and issue types than any single source alone.

**Cross-domain transfer to Terminal-Bench 2.0.** Terminal-Bench 2.0 evaluates a broader family of terminal interaction tasks beyond GitHub-issue resolution. Under the Kimi-CLI harness, Orchard-SWE retains a 20.1% resolve rate, while OpenSWE-32B drops to 0.0%. Both systems degrade substantially relative to their SWE-bench Verified scores, but only Orchard-SWE retains a non-trivial level of capability on this out-of-domain benchmark. We hypothesize that broader trajectory diversity during training—multiple teachers, multiple harnesses, and multiple task sources—provides indirect exposure to more varied tool-use and terminal-interaction patterns than narrower training corpora.

**Discussion.** Two largely independent generalization improvements are visible in Table 8: robustness across harnesses and across tasks. Both trace to diversity choices at three layers of the Orchard-SWE recipe. **Data design:** trajectories span two harnesses (mini-swe-agent, OpenHands), two teachers (MiniMax-M2.5, Qwen3.5-397B), and three complementary task sources (SWE-rebench, SWE-rebench V2, Scale-SWE), yielding 107K trajectories that vary along harness, repository structure, and issue type. **Orchard Env:** it makes data-design choices practical at scale and exposes only sandbox lifecycle, command execution, and file I/O—and imposes no assumptions about the harness or tool schema sitting above it—any harness can compose with the same env layer at zero adaptation cost. **Learning design:** credit-assignment SFT extracts partial-progress supervision from the unresolved trajectories that resolved-only recipes discard, broadening the exploration patterns the student sees, while Balanced Adaptive Rollout (BAR) keeps RL gradients informative by enforcing a balanced positive/negative mix within each prompt group across the full difficulty spectrum. However, Orchard-SWE improves but does not solve generalization: the drop to Kimi-CLI Verified is substantial, and entirely-unseen harness or domain conditions still remain a meaningful challenge.

### 3.6 Ablations and Analysis

We ablate key design choices in Orchard-SWE to understand their contribution to the final result. Four questions guide our analysis: (i) how does data scale interact with selection strategy? (ii) how harness-coupled is the trained model? (iii) what does credit-assignment SFT contribute over resolved-only training? (iv) what does reinforcement learning add on top of large-scale SFT?

**Data scale vs. selection strategy.** We first study how SFT performance depends on training data scale and selection strategy. We hold the recipe fixed—same base model, same SFT hyperparameters, mini-swe-agent harness, no RL—and vary only the number of training trajectories ( $N \in \{512, 1024, 2048\}$ ) and the strategy used to choose those trajectories from the resolved-trajectory pool. The selection strategies fall into two families:

1) **Heuristic baselines** (no use of gold-patch information): *Random*: uniform random sampling from the resolved pool. *Diverse repo*: maximize repository diversity by capping per-repo trajectory count. *Concentrated repo*: concentrate samples on the largest-resolved “core” repositories. 2) **Property-based selectors** (use gold-patch characteristics as a signal of issue complexity): *Multi-file*: prefer instances whose gold patch modifies multiple files. *Large diff*: prefer instances with larger gold-patch diffs. *Composite*: composite scoring over multiple gold-patch properties.

Table 9 reports SFT-only resolve rates on SWE-bench Verified. Two patterns dominate the table. First, **data scale dominates selection strategy** at every regime tested. Doubling data twice (512 → 2048) on the worst-performing method (*Diverse repo*, 44.0 → 52.2) yields a +8.2-point gain, larger than the entire 5.5-point spread across all selection strategies at  $N = 512$  and far larger than the 2.0-point spread at  $N = 2048$ . Second, **the spread across selection strategies shrinks monotonically with  $N$** : 5.5 pt at  $N = 512$ , 3.2 pt at  $N = 1024$ ,

Table 9: Effect of data scale and selection strategy on SFT-only resolve rate (%) on SWE-bench Verified, evaluated under the mini-swe-agent harness. All cells use identical hyperparameters; only  $N$  (trajectories) and selection strategy vary. Per-row maximum is bolded.

$N$	Heuristic baselines			Property-based selectors		
	Random	Diverse repo	Concentrated repo	Multi-file	Large diff	Composite
512	45.8	44.0	47.9	47.5	<b>49.5</b>	48.1
1024	50.3	49.0	51.3	51.6	<b>52.2</b>	52.1
2048	52.8	52.2	53.4	<b>54.2</b>	52.9	53.7

Table 10: Cross-harness generalization on SWE-bench Verified. Rows are the harness used to collect training trajectories, columns are the harness used at evaluation. All four cells use identical training data (12K resolved trajectories on SWE-rebench, MiniMax-M2.5 teacher) and the same SFT recipe; only the harness pairing differs. Diagonal entries (matched train/eval harness) are bolded.

Training harness	Evaluation harness	
	mini-swe-agent	OpenHands
mini-swe-agent	<b>57.9</b>	19.0
OpenHands	28.0	<b>53.5</b>

2.0 pt at  $N = 2048$ . At sufficient data scale, the choice of selection strategy matters much less than scale itself.

A few specific behaviors are worth noting. *Large diff* attains the strongest small- $N$  result (49.5 at  $N = 512$ ) but saturates earliest, gaining only +0.7 pt from  $N = 1024$  to  $N = 2048$ , plausibly because the pool of large-diff gold patches is finite and additional samples come from a distribution closer to the overall mean. Counterintuitively, *Concentrated repo* beats *Diverse repo* at small  $N$  by 3.9 points, with the gap shrinking to 1.2 points at  $N = 2048$ : at small data scales, deeper exposure to a few repositories produces more transferable behaviors than thin coverage of many. Property-based selectors edge out heuristic baselines at  $N = 512$  but converge to the baselines by  $N = 2048$ , suggesting that gold-patch heuristics function as a sample-efficient prior that random sampling matches given enough data.

Even the worst (method, scale) cell in Table 9 (44.0% at  $N = 512$ ) lifts the resolve rate by 22 absolute points over the underlying base model (22.0%, Table 7), confirming that even a small dose of high-quality SFT trajectories provides most of the structural lift over the base. However, the entire ablation grid plateaus around 54% under SFT-only at  $N = 2048$ , while the full Orchard-SWE recipe—using the full 107K-trajectory corpus and adding RL—reaches 67.5% on SWE-bench Verified.

What makes this kind of scaling practical is Orchard Env itself. Its *thin, harness-agnostic service boundary* lets the same env layer serve any harness, enabling multi-harness data collection and training at no additional infrastructure cost. *Image-agnostic agent injection* allows arbitrary task images to be added to the corpus without per-image rebuilds. *Low command-execution latency* (0.28 s; Section 2.3) keeps rollout throughput high. And *affordable cost* (an order of magnitude cheaper than managed alternatives; Table 2) makes large scale data collection and RL rollout feasible for academic research groups.

**Cross-harness generalization.** To assess whether harness choice during training affects the trained model’s ability to generalize at evaluation time, we run a controlled comparison: using 12K resolved trajectories on SWE-rebench distilled from MiniMax-M2.5, we vary only the collection harness and train two SFT models with otherwise identical recipes. We then evaluate each model under both harnesses on SWE-bench Verified, yielding results in Table 10.

The cross-harness matrix reveals a sharp diagonal–off-diagonal gap. Models evaluated on the same harness used during training reach 53.5–57.9% resolve rate, but performance

collapses to 19.0–28.0% under the mismatched harness. This asymmetry suggests that Open-Hands trajectories, which expose richer tool semantics and more structured observations, transfer slightly better to the simpler mini-swe-agent setting than the reverse. The dominant effect, however, is that the model has not learned harness-agnostic SWE skills: tool-call format, observation structure, and turn-level conventions are tightly coupled to the harness seen during training. This finding aligns with concurrent observations on harness coupling reported in the Qwen3-Next-Coder report (Cao et al., 2026). The implication is that no single-harness training corpus can produce an agent that generalizes well across the harness ecosystem; multi-harness training is necessary.

**Effect of credit-assignment SFT.** We isolate the contribution of credit-assignment SFT through a controlled, scale-matched comparison. Starting from the full resolved pool, we sub-sample 32K resolved trajectories so that the resolved baseline matches the 32,536 unresolved-trajectory rise segments in size, and train two SFT models with otherwise identical recipes: (i) resolved-only (32K trajectories), and (ii) resolved + unresolved with credit-assignment SFT (32K resolved + 32K rise-segment trajectories). On SWE-bench Verified, the resolved-only baseline reaches 59.3%, while adding credit-assignment SFT improves the resolve rate to 61.2%—a gain of +1.9 points. This gain validates that credit-assignment SFT extracts useful supervision from otherwise-discarded unresolved trajectories rather than fitting noise. In the full Orchard-SWE recipe, the same signal compounds with the larger 74.6K-resolved corpus, contributing to the headline 64.3% on SWE-bench Verified.

**Effect of reinforcement learning.** A natural question is how RL’s benefit depends on the strength of the SFT checkpoint it builds on—particularly for *out-of-distribution* generalization, where heavy SFT may leave less surface for RL to preserve cross-distribution capability. We compare RL initialized from two SFT checkpoints that differ by roughly two orders of magnitude in supervision: a *moderate* checkpoint (the Composite/ $N=512$  cell of Table 9, 48.1% on SWE-bench Verified) and a *heavy* checkpoint (the full 107K-trajectory recipe, 64.3%). We evaluate on SWE-bench Verified (in-distribution) and SWE-bench Multilingual (OOD) under mini-swe-agent. The two initial models respond to RL very differently. From the moderate init, RL improves both axes, with the OOD gain *larger* than the in-distribution gain: Verified 48.1%  $\rightarrow$  50.1% (+2.0 pt) and Multilingual 22.0%  $\rightarrow$  28.7% (+6.7 pt). From the heavy init, RL still improves Verified (64.3%  $\rightarrow$  67.5%, +3.2 pt) but Multilingual slightly regresses. We read this as a specialization effect: heavy SFT places the policy on a sharper mode of the training distribution, so on-policy refinement sharpens in-distribution behavior at the cost of OOD transfer; a moderate base retains more behavioral diversity, so the same RL signal acts as broad-coverage refinement rather than narrow optimization.

## 4 Orchard-GUI

This section presents Orchard-GUI, our instantiation of the Orchard training recipe for multi-modal Graphical User Interface (GUI) agents. We describe the problem setting, trajectory collection pipeline, two-stage training recipe, main results on evaluation benchmarks.

### 4.1 Problem Setting

We adopt the standard task formulation for browser-use agents: each task is defined by a starting URL and a natural-language user intent (*e.g.*, “Find a dog bed on Amazon that is washable and has a length of at least 30 inches”). The agent must navigate from the provided `start_url` within a browser interface, interact with live web pages, and complete the task by producing a natural-language final answer (or executing the requested action). Success is evaluated using an LLM-as-a-judge, which scores the trajectory against the user intent based on the final response and the sequence of screenshots. We evaluate on three benchmarks: i) *WebVoyager* (He et al., 2024); ii) *Online-Mind2Web* (Deng et al., 2023); and iii) *DeepShop* (Lyu et al., 2025). We use the same evaluation protocol as FARA (Awadallah et al., 2025) and Molmo-Web (Gupta et al., 2026) for fair comparison.

Table 11: Browser action space: 13 atomic tools grouped by family. Full argument signatures are listed in Appendix C.

Category	Tool	Description
Pointer Mgmt.	click	Mouse-click at a screen pixel; supports single/double click and left/right/middle button.
	hover	Move the cursor to a pixel to reveal tooltips or open dropdowns.
	drag	Drag-and-drop from a start pixel to an end pixel.
Keyboard Mgmt.	write	Clear the focused input and type a string.
	press_keys	Press one or more keys, sequentially or as a hotkey combo.
Page Nav.	scroll	Scroll the page or a sub-element by a fraction of the viewport.
	goto_url	Navigate the current tab to a given URL.
	go_back	Navigate back in the browser history.
	wait	Pause for $N$ seconds to allow the page to settle.
Tab Mgmt.	new_tab	Open a new blank browser tab.
	switch_tab	Switch to the tab with the given 0-based index.
	close_tab	Close the current tab.
Termination	done	End the episode and emit the final user-facing answer.

## 4.2 Generic Tool-Calling Agent Harness

Rather than adopting a bespoke browser-agent harness such as Browser-Use<sup>7</sup>, we intentionally employ a generic multi-turn ReAct-style loop (Yao et al., 2023). This design choice avoids conflating harness-specific effects with differences in data or training recipes, and more importantly, enables a unified paradigm for agentic learning that can generalize across domains and tasks beyond GUI navigation.

Specifically, each episode begins with a system prompt that specifies the agent’s role, high-level operating guidelines, and an action schema defined in the standard OpenAI tools format. Following standard GUI-agent practice, we define a fixed action space of 13 atomic tools using the OpenAI tool-calling interface: `click`, `write`, `press_keys`, `scroll`, `wait`, `drag`, `hover`, `goto_url`, `go_back`, `new_tab`, `switch_tab`, `close_tab`, and the terminal `done(response)`. The `done(response)` action is the only mechanism for terminating an episode and serves as the sole carrier of the final user-facing output. Table 11 provides one-line summaries of each tool, with full argument specifications deferred to Appendix C.

The first user turn provides the task intent along with the initial browser observation, comprising the latest screenshot, viewport dimensions, and a tab summary (URL and title for each open tab) obtained from the task’s `start_url`. At each subsequent step, the model produces a reasoning trace inside `<think>...</think>` followed by one or more `<tool_call>` blocks. Each call is parsed and executed in the Orchard Env sandbox. The resulting tool response (e.g., Succeed: `click` on `<button>` "Continue shopping") is combined with the updated observation and appended to the context as the next user turn, framed as feedback to the preceding action. This loop repeats until the model emits `done` or a predefined step budget is exhausted.

As a single screenshot can expand to thousands of vision tokens, naively concatenating the full screenshot history quickly inflates the context window beyond any reasonable training-time length: a 30-step rollout would saturate even a 64k-token context. Empirically, the actionable information from earlier screenshots is already distilled into the agent’s prior reasoning traces, which themselves remain in context across turns. We therefore retain only the last  $k$  screenshots verbatim to reducing context length substantially. Here we show one example input to VLM for the last turn with  $k = 1$  context image and the corresponding VLM response. See Appendix D for the complete trajectory.

<sup>7</sup><https://github.com/browser-use/browser-use>

## Example Input to VLM

```

<|im_start|>system
You are a GUI agent designed to operate in an iterative loop to automate [... truncated ...]

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "click", "description": "Performs a mouse click at specific
screen coordinates.", "parameters": {...}}}

[... 12 additional tool schemas omitted ...]

</tools>

For each function call, return a json object with function name and arguments within
<tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
<|im_end|>

<|im_start|>user
<user_request>
Find a dog bed on Amazon that is washable and has a length of at least 30 inches.
</user_request>
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/ - Amazon.com
</observation>
<|im_end|>

<|im_start|>assistant
<think>
The current screen shows an Amazon verification page that requires clicking a button to continue shopping.
This is a standard anti-bot measure. I need to click the "Continue shopping" button to proceed to
the main Amazon page where I can search for dog beds.

My plan:
1. Click "Continue shopping" to get past this verification page
2. Search for "dog bed washable 30 inches" on the main Amazon page
3. Apply filters for washable beds with length >= 30 inches
4. Review results and find a suitable product
</think>

<tool_call>
{"name": "click", "arguments": {"point_2d": [501, 213]}}
</tool_call>
<|im_end|>

[... intermediate turns omitted: focus search box -> write query -> submit search -> apply filter ...]

<|im_start|>user
<tool_response>
Succeed: `click` on <span> "EHEYCIGA Washable Large Dog Be..." at (635, 514) executed. Page navigated to
https://www.amazon.com/EHEYCIGA-Washable...
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/... - Amazon.com : EHEYCIGA ...
screenshot:
<|vision_start|><|image_pad|><|vision_end|>
</observation>
</tool_response>
<|im_end|>

<|im_start|>assistant
<think>

```

### Example Response from VLM

```

I've successfully navigated to the product page for the EHEYCIGA Washable Large Dog Bed. This product
meets all the requirements specified in the user request:

1. It's washable -- confirmed by the product name and description ("Washable Large Dog Bed").
2. It has a length of 35 inches -- explicitly stated in the product title as "35x23 Inch" and confirmed
   in the size specifications at the bottom of the page showing 35" x 23" x 3.5".

The product details confirm it's designed for dogs up to 45 lbs and is labeled as "Amazon's Choice" with
good reviews (4.7 stars from 381 reviews). It's also marked as having sustainability features.

I've found a dog bed that satisfies both key requirements:
- Washable (as required)
- At least 30 inches in length (it's 35 inches, which exceeds the minimum)

The task has been successfully completed. I should now signal that the task is complete and provide the
details of the found product as the answer.
</think>

<tool_call>
{"name": "done", "arguments": {"response": "I found a suitable dog bed on Amazon that meets your
requirements: The EHEYCIGA Washable Large Dog Bed (35\" x 23\" x 3.5\") is washable and has a length
of 35 inches, which exceeds your minimum requirement of 30 inches. It's priced at $17.99, has 4.7
stars from 381 reviews, and is labeled as 'Amazon's Choice'. This bed is designed for dogs up to 45
lbs and is currently in stock."}}
</tool_call>
<|im_end|>

```

All interactions are executed within the Orchard Env sandbox: each task runs in an isolated Playwright-controlled Chromium instance (2 vCPU, 8 GiB memory) with task-specific configurations. This isolation not only helps reproducibility for scenarios involving authentication, region-locked content, and rate-limited APIs, but also enables scalable parallel execution, significantly improving training and evaluation throughput.

### 4.3 Trajectory Collection and Curation

We construct the Orchard-GUI dataset through a three-stage pipeline: (i) sourcing and filtering raw task intents into a clean seed pool, (ii) sampling teacher trajectories on those tasks within the Orchard Env, and (iii) judge-based filtering and quality curation to produce the final SFT/RL splits. Table 12 summarizes the composition of the collected dataset and the subsets used for SFT/RL.

**Task sources.** We draw task instances from the task set provided by WebGym (Bai et al., 2026b), which contains 292,092 raw instances in total. To produce a clean, evaluation-safe, and diverse pool of training prompts, we apply a five-step filtering pipeline (Figure 4). The final filtered pool consists of 15,601 unique task intents, which serve as the seed set for sampling teacher trajectories used in SFT and RL. Of these, 2,537 come from PAE-WebVoyager (Zhou et al., 2025), and 13,064 come from InSTA (Trabucco et al., 2025). These tasks span 13,063 unique hosts across six broad domain categories (Figure 5, left). They cover 425/500 (85.0%) of the MOZ Top-500 websites and 57/100 (57.0%) of the SimilarWeb Top-100 websites. Correspondingly, 48.5% (7,566) of tasks fall on a MOZ Top-500 host, and 13.0% (2,030) on a SimilarWeb Top-100 host (Figure 5, right). More detailed information is provided in Appendix E. Note that the tasks used in the RL stage are drawn from the same task pool and processed using the same filtering pipeline, but with a more restrictive similarity-based deduplication threshold of 0.95, yielding a task set of 2,198 tasks (Table 12).

**Trajectory generation.** We use Qwen3-VL-235B-A22B-Thinking (Bai et al., 2025) as the sole teacher for trajectory distillation. For each of the 15,601 filtered seed tasks, we sample 4 independent rollouts through the Orchard Env under the same Tool-Calling Agent Harness described in Section 4.1, yielding a raw pool of 62,395 teacher rollouts (a small fraction of attempts abort due to environment or rollout engine errors). GPT-4.1 serves as the judge during data collection: its verdict on the final done(response) and the agent interaction history, together with the screenshot trail is used as a binary reward. Under this judge,

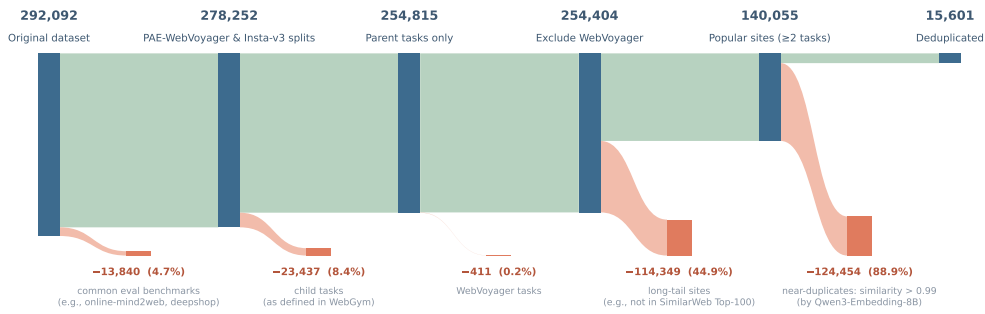


Figure 4: **Task-filtering pipeline.** Starting from 292,092 raw tasks, we sequentially remove evaluation-benchmark overlap, child tasks, WebVoyager intents, long-tail sites, and near-duplicate intents (semantic similarity  $\geq 0.99$  under Qwen3-Embedding-8B), yielding a final pool of 15,601 deduplicated seed tasks on popular websites.

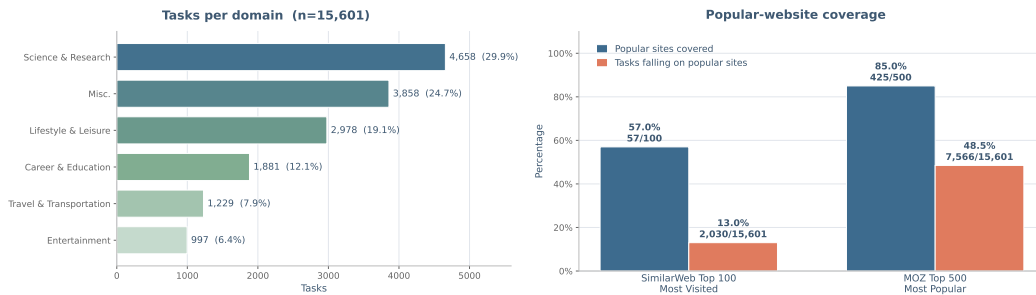


Figure 5: **Composition of the filtered seed task pool.** *Left:* task share by top-level domain (6 categories spanning 15,601 tasks). *Right:* the seed task pool covers 57.0% of SimilarWeb Top-100 Most Visited and 85.0% of MOZ Top-500 Most Popular websites, with 13.0% and 48.5% of tasks landing on those respective lists.

68.4% of tasks have at least one passing rollout, 26.3% pass on all four rollouts, and the remaining 31.6% fail on every rollout (Figure 6, left). A non-trivial share of these failures is environmental rather than agentic: of the 4,934 tasks that fail on every rollout, 41.1% (2,026 tasks; 13.0% of the full pool) are captcha-blocked on all four attempts, leaving roughly 18.6% of the pool that the teacher genuinely cannot solve. Per-website success rates also vary widely (Figure 6, right), with anti-bot-prone hosts (e.g., dictionary.cambridge.org, bing.com) clustered at the low end. Sampling four rollouts per task is intentional: the redundancy provides (i) pass-rate-based difficulty estimates and trajectory diversity for downstream curation, and (ii) a much larger candidate pool than we ultimately train on, letting us study data efficiency by training on small, carefully curated subsets rather than the full collection.

**Filtering and curation.** We first retain only rollouts whose final done(response) is judged SUCCESS by GPT-4.1 (reward = 1.0), and split the survivors by source benchmark, yielding 4,826 successful PAE-WebVoyager rollouts and 26,154 successful InSTA-v3 rollouts. For SFT we deliberately avoid using the full successful pool: oversaturating the student on imitation data before RL tends to drive it into a narrow imitation regime that on-policy gradients struggle to escape, so we instead select a small, carefully curated subset. We further restrict the SFT pool to PAE-WebVoyager: although PAE-WebVoyager contributes only 16% of the seed tasks (2,537 of 15,601), 38.8% of its tasks land on a SimilarWeb Top-100 site, versus just 3.6% for InSTA-v3 (Table 12)—a  $\sim 10\times$  density advantage on popular hosts that more closely reflect everyday user browsing habits.

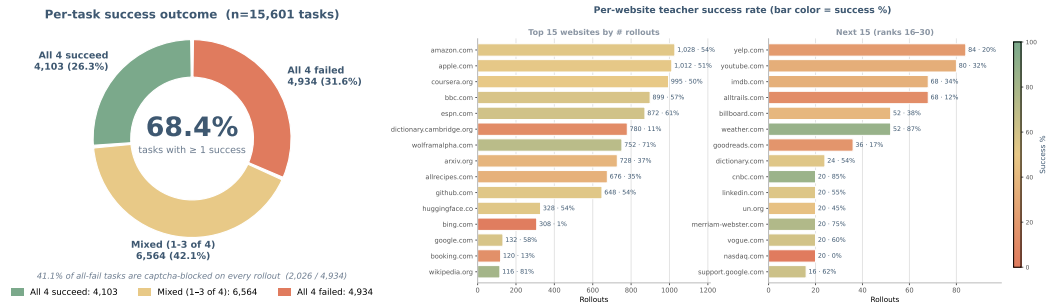


Figure 6: **Per-website and per-task success outcomes.** *Left:* share of tasks by outcome across the 4 rollouts – 26.3% all-pass, 42.1% mixed (1–3 of 4), and 31.6% all-fail. Of the all-fail tasks, 41.1% (2,026 tasks; 13.0% of the full 15,601-task pool) are captcha-blocked on every one of their four rollouts. *Right:* per-website teacher success rate for the top-30 websites by rollout count, split into two columns; bar length = number of rollouts, color = success rate.

Table 12: Composition of the Orchard-GUI training dataset. The *Full Set* is the pool of seed tasks for which we collected rollouts. *SFT Trajectories* are the judge-passing rollouts used for supervised fine-tuning, and *RL Tasks* are seed prompts used to bootstrap rollout-based optimization. Tasks are classified by whether *all* rollouts succeeded (All Succ.), *all* failed (All Failed), or the outcome was mixed. The last column reports the number of tasks (restricted to those with at least one successful rollout) whose start URL falls on a SimilarWeb Top-100 site.

Subset	Source	# Tasks	All Succ.	All Failed	Mixed	Tasks on Top-100
Full Set	PAE-WebVoyager	2,537	587	800	1,150	985
	InSTA-v3	13,064	3,516	4,134	5,414	473
	<i>Total</i>	<b>15,601</b>	<b>4,103</b>	<b>4,934</b>	<b>6,564</b>	<b>1,458</b>
SFT Traj.	PAE-WebVoyager	<b>412</b>	<b>128</b>	<b>0</b>	<b>284</b>	<b>196</b>
RL Tasks	PAE-WebVoyager	734	189	226	319	290
	InSTA-v3	1,464	500	461	503	469
	<i>Total</i>	<b>2,198</b>	<b>689</b>	<b>687</b>	<b>822</b>	<b>759</b>

Within the PAE-WebVoyager success pool we then apply two reductions to balance quality and diversity. (i) *Within-task quality:* for each task we keep a single rollout, namely the *shortest* successful trajectory (fewest turns, with ties broken by total response length), since shorter teacher trajectories tend to be cleaner and contain less recovery noise. (ii) *Across-website diversity:* we cap each website at  $K = 20$  tasks, preventing high-volume hosts (e.g. amazon.com, coursera.org) from dominating the SFT mix. The resulting SFT corpus comprises 412 unique tasks spanning 70 websites; per-source breakdowns and outcome statistics for this subset and the RL pool are reported in Table 12.

#### 4.4 Training Recipe

Our training recipe follows a two-stage pipeline: supervised fine-tuning (SFT) on teacher-distilled trajectories, followed by reinforcement learning (RL) with judge-based rewards. Both stages use the Orchard Env as the execution backend.

**Stage 1: Supervised fine-tuning.** We initialize from Qwen3-VL-4B-Thinking (Bai et al., 2025) and fine-tune on the curated teacher trajectories. From each teacher rollout we generate one training example per assistant turn: the  $t$ -th example carries the chat-template-serialized prefix up through turn  $t$  and supervises only that turn’s assistant response. The serialized prefix follows the Qwen chat template, with a system turn carrying the agent role and the OpenAI-format tool schema; an initial user turn with the task intent and the start\_url observation; and the subsequent alternation of assistant turns (a <think>...</think> rea-

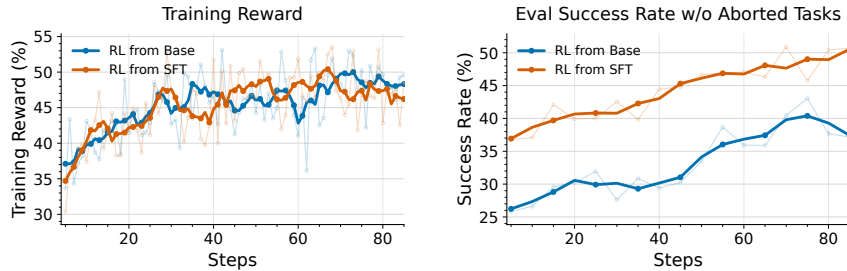


Figure 7: **Orchard-GUI RL Training and Evaluation Curve.** The red curves denote RL training starting from our SFT checkpoint, while the blue curves denote RL training initialized from the base model. Compared with the base model initialization, the SFT-initialized model achieves consistently higher evaluation success rates and more stable reward improvements throughout training.

soning trace followed by one or more `<tool_call>` blocks) and user turns (the tool response wrapping an updated browser observation that includes the latest screenshot). Following standard practice for long-horizon agent training, the loss is computed only on the final (target) assistant turn; the system prompt, the earlier assistant turns retained as in-context history, and every environment observation are masked out. The vision encoder and multi-modal projector are kept frozen and only the language-model weights are updated, which preserves the backbone’s screenshot-grounding capability and concentrates SFT capacity on agent-specific reasoning and action prediction. We train for 3 epochs with peak learning rate  $10^{-5}$  under a cosine schedule with a 10% linear warmup. Each optimizer step uses a per-device batch of 2 with 8-step gradient accumulation, giving a per-worker effective batch of 16 and a global batch of 128 across 8 data-parallel workers.

**Stage 2: Reinforcement learning.** Starting from the SFT checkpoint, we apply RL to improve the model’s ability to recover from errors and explore alternative paths under partial observability. We optimize a multi-turn variant of GRPO (Shao et al., 2024): for each task we sample a group of  $G$  trajectories from parallel browser instances, compute a group-relative advantage from the trajectory-level reward, and broadcast it to every assistant-response token across all turns; observation and environment-feedback tokens are masked out of the loss. The reward combines a deterministic format check with a binary judge: a trajectory receives +1 when every assistant turn parses as a valid `<think>+tool-call` and the final `done(response)` is judged SUCCESS by GPT-4.1 against the screenshot trail and user intent, -1 when the rollout terminates from repeated format failures, and 0 otherwise. We use asymmetric PPO clipping ( $\epsilon_{\text{low}}=0.2$ ,  $\epsilon_{\text{high}}=0.28$ ) without KL or entropy regularization, and intentionally omit the per-trajectory  $1/T_i$  loss normalization so that longer, harder tasks are not down-weighted. To remove uninformative updates we apply DAPO-style trajectory-level dynamic sampling (Yu et al., 2025)—dropping groups whose rewards are all 0 or all +1—and additionally zero the loss mask for judge API failures and captcha-aborted runs via the `remove_sample` mechanism, so that infrastructure noise does not leak into the policy update. We further adopt a step-budget curriculum within RL: we first run RL with the per-episode step budget capped at 15 until performance saturates, then continue training from that checkpoint with the budget raised to 30. The short-horizon phase produces dense reward signal cheaply on tasks the policy can already solve within 15 steps, while the long-horizon phase extends the policy to harder tasks that genuinely require more interaction.

#### 4.5 Main Results

Table 13 compares Orchard-GUI with proprietary VLMs, prior open-source GUI agents, and same-scale baselines on WebVoyager, Online-Mind2Web, and DeepShop. After two-stage training, Orchard-GUI reaches 74.1% / 67.0% / 64.0% on WebVoyager / Online-Mind2Web / DeepShop, for a 68.4% average—the strongest open-source result by a wide margin and

Table 13: GUI-agent success rates (%) across three open-web benchmarks. \* marks numbers reported in FARA (Awadallah et al., 2025); † marks numbers reported in MolmoWeb (Gupta et al., 2026).

System	# Steps	# Tasks	WebVoyager	Online-M2W	DeepShop	Avg
<i>Proprietary Models</i>						
GPT-5 (Axtree) <sup>†</sup>	30	–	70.6	41.9	40.7	51.1
Gemini-3-flash (Axtree) <sup>†</sup>	30	–	74.4	34.8	45.1	51.4
Gemini-3-flash (Axtree) <sup>†</sup>	100	–	85.6	44.8	55.3	61.9
GPT-4o (SoM)*	100	–	65.1	34.6	16.0	38.6
o3 (SoM)*	100	–	79.3	55.4	49.7	61.5
GPT-5 (SoM)*	100	–	90.6	57.7	49.1	65.8
OpenAI computer-use-preview*	100	–	70.9	58.3	24.7	51.3
Gemini computer-use-preview <sup>†</sup>	100	–	88.6	57.3	62.0	69.3
<i>Open-Source Models</i>						
Holo1-7B <sup>†</sup>	30	>15.6k	55.4	–	–	–
UI-TARS-1.5-7B*	100	–	66.4	31.3	11.6	36.4
GLM-4.1V-9B-Thinking*	100	–	66.8	33.9	32.0	44.2
Fara-7B*	100	>123.2k	73.5	34.1	26.2	44.6
MolmoWeb-4B <sup>†</sup>	100	>278.5k	75.2	31.3	35.6	47.4
MolmoWeb-8B <sup>†</sup>	100	>278.5k	<b>78.2</b>	35.3	42.3	51.9
Qwen3-VL-4B-Thinking	30	–	49.0	32.0	33.3	38.1
Qwen3-VL-235B-A22B-Thinking	30	–	63.1	63.7	56.7	61.2
<b>Orchard-GUI-4B-SFT</b>	30	0.4k	60.2	47.0	48.7	52.0
<b>Orchard-GUI-4B (SFT + RL)</b>	30	2.6k	74.1	<b>67.0</b>	<b>64.0</b>	<b>68.4</b>

competitive with the best proprietary system (Gemini computer-use-preview, 69.3% avg) despite a 4B backbone and only 2.6k training tasks. RL contributes most of this gain, lifting the SFT checkpoint by +13.9 / +20.0 / +15.3 absolute points across the three benchmarks (52.0% → 68.4% average).

Four findings stand out. First, on WebVoyager Orchard-GUI is on par with the strongest open-source baselines (74.1% vs. MolmoWeb-4B’s 75.2% and MolmoWeb-8B’s 78.2%) while consuming roughly two orders of magnitude fewer training tasks (2.6k vs. >278.5k). WebVoyager covers only 15 popular sites with relatively short horizons, leaving little room to separate from baselines that have been heavily distilled on this exact distribution.

Second, on Online-Mind2Web and DeepShop Orchard-GUI substantially outperforms every previous open-source model—by +31.7 and +21.7 absolute points over MolmoWeb-8B, the strongest prior open baseline—and also surpasses its own 235B Qwen3-VL teacher by +3.3 / +7.3, demonstrating that environment-grounded RL extracts capability the teacher itself does not exhibit.

Third, the training dynamics in Figure 7 show that RL initialized from the SFT checkpoint consistently achieves higher evaluation success rates and more stable optimization behavior than RL initialized directly from the base model. While both settings obtain comparable training rewards, the SFT-initialized policy converges to substantially stronger generalization performance, ultimately reaching over 50% success on the evaluation set compared with below 40% for base-model initialization. This gap indicates that supervised initialization provides a crucial behavioral prior that stabilizes exploration and enables RL to more effectively translate reward optimization into downstream task success.

Finally, the largest gains appear on Online-Mind2Web, which spans a substantially broader and more diverse website distribution than either WebVoyager (15 fixed sites) or DeepShop (a single shopping vertical). Success on this benchmark therefore requires generalization to previously unseen interfaces rather than adaptation to a narrow site set. The fact that Orchard-GUI improves most strongly in this regime suggests that judge-grounded RL over a relatively small but diverse task pool can generalize across the open web more

effectively than large-scale teacher distillation on narrow distributions, which is ultimately the practically relevant setting for deployable browser agents.

## 5 Orchard-Claw

This section presents Orchard-Claw, our instantiation of the Orchard training recipe for claw-based agents. We describe the problem setting, trajectory collection methods, two-stage training recipe, main results on Claw-Eval (Ye et al., 2026), and ablations that isolate key design choices that impacts performance.

### 5.1 Problem Setting

**Task and evaluation** We target multi-step daily workflow tasks formulated by Claw-Eval (Ye et al., 2026). Given a task instruction such as “Sort my inbox — which emails need a reply, which are notifications, and which are spam?”, the agent need to interact with a diverse set of daily tools such as “gmail\_list\_messages”, “gmail\_get\_message”, etc., to complete the task while being safe and robust. Specifically, after the agent completes the task, the evaluation audits the entire agent trajectory to measure the completion, safety, and robustness of the agent using a combination of automated scripts and LLM-as-a-judge (Zheng et al., 2023; Xiong et al., 2026). These three dimensions are aggregated into a single task score,  $\text{task\_score} = \text{safety} \times (0.8 \cdot \text{completion} + 0.2 \cdot \text{robustness})$ , and a task is counted as a pass if  $\text{task\_score} \geq 0.75$ . We use **Claw-Eval** as our primary evaluation benchmark.

**Agent harness and tool interface** We collect trajectories using two different agent harnesses: a ReAct-style harness defined by the Claw-Eval benchmark, and the ZeroClaw (ZeroClaw Labs, 2026) harness – a faster, more lightweight Rust version of the popular OpenClaw (OpenClaw Team, 2026). All environment and harnesses are implemented in a docker runtime routed through the Orchard Env service: each task run (environment and harness) runs in an isolated sandbox (2 vCPU, 2 GiB memory), provisioned from a python-based image with ClawEval and ZeroClaw pre-installed. During both SFT and RL, we train Orchard-Claw on both harnesses and study whether such end-to-end training helps the model better leverage advanced harnesses such as ZeroClaw and reach higher performance.

### 5.2 Trajectory Collection and Curation

As claw-based agents are relatively new, we conduct a preliminary study using Claude Opus 4.6 (Anthropic, 2026) to synthesize claw-agent tasks as our training set.

**Task sources.** We draw seed tasks from two sources: (1) tasks from Claw-Eval, and (2) workflows from popular skills on ClawHub<sup>8</sup>. From these seeds, we prompt Opus 4.6 via claude-agent-sdk to synthesize new tasks in a four-step loop: (1) propose and filter task ideas; (2) generate the environment, files, tool server, and test script; (3) run MiniMax-M2.5 (MiniMax, 2026) as the solver to produce rollouts; (4) refine the task based on the rollouts to ensure feasibility and instruction clarity. Each task costs 4.9 USD to synthesize on average, yielding 192 tasks in total shared across the Claw-Eval and ZeroClaw harnesses.

**Trajectory generation.** For simplicity, we distill SFT data from a single teacher model. We choose MiniMax-M2.5 for its strong performance. For each synthesized task, we sample five rollouts from MiniMax-M2.5 through Orchard Env under the corresponding harness (ReAct-style or ZeroClaw), and keep only the trajectories that complete the task. To record training samples from complex harnesses such as ZeroClaw, we implement a proxy LLM server that records every LLM call (input and output) during the rollout. An example recorded (input and output) pair from the ZeroClaw harness is shown below. Once a rollout finishes, each recorded (input and output) pair is grouped back as a trajectory for training

<sup>8</sup>We use the official OpenClaw CLI to access skills on <https://clawhub.ai/>

(and also for reward computation during RL). This yields 561 trajectories with 4537 training pairs in total for SFT.

### Example Model Input in the ZeroClaw Harness

```
<|im_start|>system
You are ZeroClaw, an AI assistant with tool access. Use tools to accomplish tasks.
Be concise and helpful. Execute tools directly when needed without excessive explanation.

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "shell", "description": "Execute a shell command and return the
output.\n\nArgs:\n  command: The shell command to execute\n\nReturns:\n  The command output
(stdout and stderr combined)", "parameters": {"properties": {"command": {"type": "string"}},
"required": ["command"], "type": "object"}}}
[... truncated ...]

For each function call, return a json object with function name and arguments within
<tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call><|im_end|>
<|im_start|>user
Diana from operations emailed about the Q2 inventory rotation. Read her email and Tom's follow-up for full
instructions. Check the inventory, identify critical and expiring items, email the relevant
suppliers (look up their contacts first), set up weekly scheduler follow-ups for critical items, and
write the rotation plan summary to /home/user/rotation_plan.txt.

<|im_end|>
<|im_start|>assistant
<think>
The user wants me to:
1. Read Diana's email about Q2 inventory rotation
2. Read Tom's follow-up email
3. Check inventory
4. Identify critical and expiring items
5. Email relevant suppliers (look up contacts first)
6. Set up weekly scheduler follow-ups for critical items
7. Write rotation plan summary to /home/user/rotation_plan.txt

Let me start by listing recent emails to find Diana's email and Tom's follow-up.
</think>

<tool_call>
{"name": "gmail_list_messages", "arguments": {"days_back": 14, "max_results": 20}}
</tool_call><|im_end|>
<|im_start|>user
<tool_response>
{"messages":[{"message_id": "msg_401", "from": "d.kowalski@ourcompany.com", "subject": "Q2 Inventory Rotation -
Need Your Help", "date": "2026-04-10T03:53:24Z", ...
[... truncated ...]
</tool_response><|im_end|>
<|im_start|>assistant
<think>
I found Diana's email (msg_401) and Tom's follow-up (msg_402). Let me get the full content of both.
</think>

<tool_call>
{"name": "gmail_get_message", "arguments": {"message_id": "msg_401"}}
</tool_call>
<tool_call>
{"name": "gmail_get_message", "arguments": {"message_id": "msg_402"}}
</tool_call><|im_end|>
<|im_start|>user
<tool_response>
{"message_id": "msg_401", "from": "d.kowalski@ourcompany.com", "to": "assistant@ourcompany.com", "subject": "Q2
Inventory Rotation - Need Your Help", "body": "Hi,\n\nIt's time for our quarterly inventory rotation
review. I need you to go through our current inventory and identify all items that are CRITICAL
(stock below minimum threshold) or EXPIRING SOON ...
[... truncated ...]
</tool_response>
<tool_response>
{"message_id": "msg_402", "from": "t.nguyen@ourcompany.com", "to": "assistant@ourcompany.com", "subject": "Re: Q2
Inventory Rotation - Budget Note", "body": "Hey,\n\nJust a heads up for the inventory rotation - our
Q2 procurement budget is tight. For any item where the restock cost would exceed $1000 (quantity
```

```

needed to reach minimum x unit cost), flag it as HIGH PRIORITY in the rotation plan so I can
fast-track the PO approval.\n\nAlso, once you've emailed the suppliers, send me a summary email at
t.nguyen@ourcompany.com listing all restock requests you've sent out.\n\nTom Nguyen\nProcurement
Lead", "date": "2026-04-10T05:38:24Z", "is_read": false, "labels": ["inbox", "operations"]}]
</tool_response><|im_end|>
<|im_start|>assistant
<think>

```

### Example Response from LLM

```

Now I understand the task:
1. Check inventory for items that are:
  - CRITICAL: stock below minimum threshold
  - EXPIRING SOON: expiry date within 45 days from April 10, 2026 (so before May 25, 2026)
2. Email suppliers for restocking quotes - group by supplier
3. Set up weekly scheduler follow-ups for critical items
4. Write rotation plan summary to /home/user/rotation_plan.txt
5. Also need to flag items >$1000 as HIGH PRIORITY for Tom

Let me get the inventory list first.
</think>

<tool_call>
{"name": "inventory_list_products", "arguments": {}}
</tool_call>
<|im_end|>

```

## 5.3 Training Recipe

Our training recipe follows a two-stage pipeline: supervised fine-tuning (SFT) on teacher-distilled trajectories, followed by reinforcement learning (RL). Both stages use Orchard Env as the execution backend. We use Qwen3-30B-A3B-Thinking-2507 (Qwen Team, 2025) as the backbone model for our training.

**Stage 1: Supervised fine-tuning.** We initialize from the base backbone and fine-tune on the curated teacher trajectories. Each training sample is an (input prompt, LLM response) pair logged by our proxy LLM server, and following Section 3.3 we mask the input and train only on the response. We run SFT for 1 epoch with a global batch size of 16 and a 64k context window, using a cosine learning rate decayed from  $10^{-5}$  to  $10^{-6}$ , and apply left truncation to sequences exceeding the context window. At inference time, we extend the context to the model’s maximum of 256k.

**Stage 2: Reinforcement learning.** Starting from the SFT checkpoint, we apply RL to teach the model to recover from errors and explore alternative paths to task completion. The reward is binary and environment-grounded: if a rollout passes all test scripts, every (input, output) pair in the rollout receives +1; otherwise, every (input, output) pair receives -1. We optimize with standard GRPO (Shao et al., 2024; Guo et al., 2025) using a batch size of 8 and group size of 8 over 150 training steps. Orchard’s sandbox parallelization is critical at this stage, allowing us to easily run 64 asynchronous rollout sandboxes per step, which substantially improves training throughput. Additionally, during rollout we do not set a maximum step limit but rather a 10-minute wall-clock budget for each task. We find this better accommodates the differing per-step latencies of different harnesses and different tool calls, and better matches real-world usage. Rollouts that exceed the budget are aborted, and all of their turns are excluded from training. In Figure 8 we plot the training and validation success rate and trajectory length over the course of RL training.

## 5.4 Main Results

Table 14 compares Orchard-Claw against large proprietary models and open models of similar size on Claw-Eval (Ye et al., 2026), using its native ReAct-style harness. After our two-stage training, Orchard-Claw reaches 31.7% *pass*<sup>3</sup> and 59.6% *pass*@3, substantially

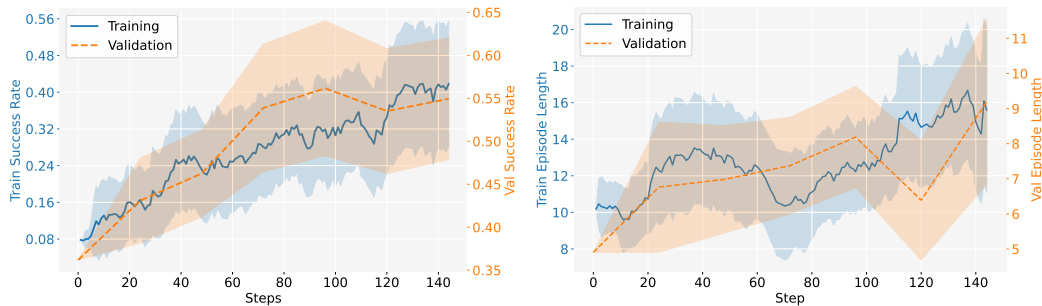


Figure 8: **Orchard-Claw RL training curves.** *Left:* train and validation success rate over RL steps. *Right:* train and validation episode length (number of agent turns per rollout). Validation tasks are sampled from the ClawEval benchmark. Both metrics rise steadily over the course of training, indicating that the agent learns to solve more tasks while also engaging in longer multi-turn interactions.

Table 14: Claw-agent performance on Claw-Eval. We use the general domain (0408) for evaluation. \* marks numbers reported by Ye et al. (2026).

System	#Tasks	ClawEval( $pass^3$ )	ClawEval( $pass@3$ )
<i>SOTA Large Language Models</i>			
Claude Opus 4.6*	–	70.8	80.8
GPT 5.4*	–	60.2	75.8
Gemini 3.1 Pro*	–	55.9	80.8
Qwen3.5 397A17B*	–	57.8	70.8
GLM 5 Turbo*	–	52.8	73.3
MiniMax M2.7*	–	49.7	72.0
MiniMax M2.5	–	47.2	65.2
Kimi K2.5*	–	36.6	67.1
<i>Similar-size baselines and our model (30B-A3B; ~3B active)</i>			
Nemotron-3-nano-30b-a3b	–	26.1	57.8
Qwen3-30B-A3B-Thinking	–	14.3	39.8
Qwen3-Coder-30B-A3B-Instruct	–	30.4	49.7
<b>Orchard-Claw (SFT)</b>	0.2k	22.4	50.3
<b>Orchard-Claw (SFT + RL)</b>	0.2k	<b>31.7</b>	<b>59.6</b>

outperforming its backbone and also surpassing code- and tool-call specialized models such as Qwen3-Coder-30B-A3B-Instruct (Cao et al., 2026) and Nemotron-3-nano-30b-a3b (NVIDIA, 2025), despite being trained on only 0.2k synthetic tasks. RL contributes most of this gain, adding 9.3 absolute points on  $pass^3$  and 9.3 absolute points on  $pass@3$  over the SFT checkpoint. This suggests that even with limited synthetic data, RL is highly effective at refining agent behavior beyond what teacher distillation alone can offer.

In Table 15 we further evaluate Orchard-Claw under both its native ReAct-style harness and the more advanced ZeroClaw harness. Pairing Orchard-Claw (SFT+RL) with ZeroClaw lifts performance to 41.0%  $pass^3$  and 73.9%  $pass@3$ , a +9.3 and +14.3 absolute improvement over the same model run under the ReAct-style harness. This gain is also the largest among all models in the comparison, including baselines such as Qwen3-Coder-30B-A3B-Instruct that benefit much less or even regress when switching to ZeroClaw. We attribute this to our end-to-end training using the target harnesses during rollout, enabled by Orchard Env. By exposing the agent to the harnesses during training, the agent learns to take advantage of the features — including but not limited to subagents, auto-compact, and more — that stronger harnesses offer at inference time.

Table 15: Cross-harness evaluation. ReAct\* is the ReAct-style loop from the ClawEval benchmark. ZeroClaw is a lightweight, Rust version of the popular OpenClaw Harness.

Model	ClawEval( <i>pass</i> <sup>3</sup> )		ClawEval( <i>pass</i> @3)	
	ReAct*	ZeroClaw	ReAct*	ZeroClaw
Qwen3-30B-A3B-Thinking	14.3	20.5 (+6.2)	39.8	44.7 (+4.9)
Qwen3-Coder-30B-A3B-Instruct	30.4	29.8 (-0.6)	49.7	54.7 (+5.0)
<b>Orchard-Claw (SFT)</b>	22.4	25.5 (+3.1)	50.3	62.1 (+11.8)
<b>Orchard-Claw (SFT+RL)</b>	<b>31.7</b>	<b>41.0 (+9.3)</b>	<b>59.6</b>	<b>73.9 (+14.3)</b>

## 6 Related Work

**Interactive environment orchestration for agentic training.** Unlike traditional model training, the cornerstone of agentic training is interactive environment orchestration. It requires agents to execute actions, process feedback, and iterate through multi-turn trajectories within isolated sandboxes. To address the specialized demand on the underlying infrastructure layer, there are two distinct design paradigms have emerged: integrated training stacks and decoupled environment services.

In the integrated paradigm, the execution environment is a sub-component embedded within a larger training or orchestration system. This allows the co-design of the environment layer with specific training frameworks or agent harnesses, tailoring the infrastructure for a particular task pipeline. MegaFlow (Zhang et al., 2026b) decomposes agentic training into three co-designed services (Model, Agent, Environment), having coordinated tens of thousands of concurrent agent tasks. While it recognizes that the environment service should be independently scalable, the three services are co-designed for the Qwen training pipeline and are not designed to be composed with arbitrary external trainers or third-party harnesses. ProRL Agent (Zhang et al., 2026a) achieves an important partial step toward decoupling: it separates rollout generation from the trainer via an HTTP service. However, its environment layer remains bound to agent scaffolding through AgentHandler plugins, so the harnesses cannot be swapped without modifying the environment configuration. In contrast, Orchard Env’s REST API is explicitly decoupled from the training loop, the specific task and the agent harness. This modularity makes it uniquely amenable for open-source development and heterogeneous research environments. By abstracting the environment into a standalone service, our environment orchestration supports the entire agentic development life cycles: trajectory distillation, on-policy rollouts and evaluation.

The second paradigm is decoupled environment services, where the execution environment is exposed as a thin, independent service with a minimal API surface, reusable across different training frameworks, agent scaffolds, and task domains. Commercial platforms such as E2B (E2B, 2024) Daytona (Daytona, 2025) and Modal (Modal Labs, 2024) exemplify this approach for developer-facing use cases: they expose REST APIs or SDKs for sandbox lifecycle and code execution. However, these platforms generally lack the fine-grained environment controls required for scalable RL training, e.g., tunable resource limits, heartbeat-based lifecycle management tied to training state, per-sandbox network isolation policies. Furthermore, the operational costs of these proprietary services are typically significantly higher than those of Orchard Env, which utilizes a Kubernetes-native design to maximize resource efficiency and minimize overhead. Consequently, Orchard Env serves as a high-performance, cost-effective foundation specifically engineered for open-source research and agentic development.

**Software engineering agents.** Automated software engineering has converged on a canonical task formulation: given a real GitHub issue and repository snapshot, produce a patch that passes the associated test suite. SWE-bench (Jimenez et al., 2024) and its human-validated SWE-bench Verified subset (OpenAI, 2024) operationalize this formulation and serve as the primary evaluation benchmark for Orchard-SWE. Within this landscape, work has pursued two complementary directions: designing better agent scaffolds and scaling training data. On the scaffold side, SWE-agent (Yang et al., 2024) introduces a specialized

agent-computer interface enabling structured file viewing, editing, and codebase search. Its lightweight derivative, mini-swe-agent, serves as one of Orchard-SWE’s two training harnesses. Additionally, OpenHands (Wang et al., 2025b) provides a full-featured multi-agent platform that is used as the second harness for Orchard-SWE. On the data-scaling side, SWE-smith (Yang et al., 2025a) generates new training instances from arbitrary repositories via automated task synthesis, scaling task diversity, BugPilot (Sonwane et al., 2025) generates “unintentional” bugs by instructing agents to implement new features in a repository. Orchard-SWE is orthogonal: rather than synthesizing new tasks, we scale trajectory quality through multi-teacher distillation from frontier models and partial-credit supervision on failed traces, drawing from real Github issues (Badertdinov et al., 2025; 2026; Zhao et al., 2026) as task sources.

**GUI and browser navigation agents.** GUI agent research is organized around a set of complementary benchmarks that together span the structural diversity of real-world web and desktop tasks. Mind2Web (Deng et al., 2023) introduces the first large-scale dataset of human-annotated cross-website tasks across 137 sites, establishing the dominant web-navigation evaluation suite. OSWorld (Xie et al., 2024) extends evaluation to full desktop environments with 369 real computer tasks across multi-application workflows, requiring agents to operate over screenshots with no DOM access. WebVoyager (He et al., 2024) establishes an end-to-end web task benchmark using live websites with GPT-4V, serving as both a benchmark and a prompting-only baseline. More recently, Online-Mind2Web revisits the Mind2Web task space in a fully live setting, removing the static snapshot shortcut, and DeepShop (Lyu et al., 2025) introduces a transactional e-commerce benchmark requiring multi-step reasoning under shopping constraints — both serving as held-out evaluation targets for Orchard-GUI. We evaluate Orchard-GUI on WebVoyager, Online-Mind2Web, and DeepShop — selected specifically because they represent structurally distinct task types, use live environments rather than static snapshots, and have no overlap in their action spaces or reward signals, making them a demanding testbed for a single unified model without benchmark-specific tuning.

The methodological progression in this field moves from these prompting-only baselines toward sophisticated training-heavy paradigms. Early systems like WebVoyager (He et al., 2024) establish strong prompting baselines but leave substantial headroom for trained models. The subsequent dominant paradigm focuses on supervised fine-tuning (SFT) on human or model-generated demonstrations. Fara (Awadallah et al., 2025) introduces FaraGen, a scalable pipeline that proposes multi-step web tasks and filters successes via automatic verifiers to produce low-cost SFT data, yielding a screenshot-only 7B agent competitive with frontier models. More recently, MolmoWeb (Gupta et al., 2026) assembles MolmoWebMix—a large curated blend of synthetic trajectories, human demonstrations, and atomic web-skill data—to train a fully open agent that achieves state-of-the-art among open-weight models across WebVoyager, Online-Mind2Web, and DeepShop. A newer wave of research integrates reinforcement learning (RL) to enhance reasoning and out-of-distribution (OOD) performance. UI-TARS (Qin et al., 2025) pioneers an iterative RL data flywheel for GUI agents. Recent open-source methods (Luo et al., 2025; Lu et al., 2025) focus more on improving grounding and reasoning quality through RL. Despite these advances, cross-benchmark generalization across both online and offline environments under a unified training setup remains rare. Orchard-GUI trained with Orchard Env as a harness-agnostic execution backend demonstrates that a single SFT+RL recipe applied on a 4B model achieves cross-domain generalization. This provides concrete evidence for the scalability and reusability of the proposed open development system.

**Generalist long-running autonomous agents (Claw-agent).** Claw-style agents represent a shift from episodic domain tools (SWE/GUI) toward persistent, general-purpose partners. While domain agents optimize for specific environments with resetting memory, Claw-agents maintain persistent state and identity via structured artifacts. They leverage dynamic skill libraries (ClawHub), execute multi-step workflows across heterogeneous APIs, and use proactive heartbeats to sustain an ambient presence. This architectural divergence targets conversational alignment over open-ended horizons, making cross-harness generalization a central challenge as the agent’s tool surface continually expands.

Several recent benchmarks (Ye et al., 2026; Li et al., 2026; Bai et al., 2026a) establish the evaluation standards for Claws-type agents. Claw-Eval (Ye et al., 2026) provides high-quality, human-curated scenarios that rigorously assess long-term planning and tool-calling stability. In contrast, ClawGym (Bai et al., 2026a) relies on automated data synthesis over mock workspaces to create a scalable data pipeline for training and evaluation. Recent training innovations focus on efficiency and rapid adaptation. MetaClaw (Xia et al., 2026) enables continual skill synthesis from failure trajectories and idle-period RL updates, showing gains on Claw benchmarks. Furthermore, OpenClaw-RL (OpenClaw Team, 2026) treats live deployment signals, such as user feedback, as a continuous training source via Hindsight-Guided On-Policy Distillation. Using Orchard-Env, we instantiate an end-to-end training pipeline, Orchard-Claw, to achieve continual improvement on Claw-Eval through a unified, harness-agnostic execution backend.

## 7 Conclusion

This paper presented **Orchard**, an open-source framework for scalable agentic modeling built around a thin, Kubernetes-native, harness-agnostic environment service. By decoupling sandbox management from agent harnesses, trainers, and task domains, Orchard Env makes trajectory collection, SFT, RL rollouts, and evaluation more reusable, reproducible, and cost-effective. Across software engineering, GUI navigation, and personal-assistant workflows, Orchard demonstrates that a shared environment layer can support diverse agents and training recipes. Orchard-SWE, Orchard-GUI, and Orchard-Claw achieve strong results while showing improved transfer across harnesses, domains, and pipeline stages. Overall, Orchard shows that scalable agentic progress depends on both infrastructure and training design. By making the environment service, training recipes, and data collection reusable across domains and harnesses, Orchard lowers the barrier to open, reproducible, and capability-focused research in agentic AI.

## References

- Anthropic. Introducing claude opus 4.6. <https://www.anthropic.com/news/claude-opus-4-6>, 2026. Accessed: 2026-05-03.
- Alpay Ariyak, Junda Zhang, Junxiong Wang, Shang Zhu, Federico Bianchi, Sanjana Srivastava, Ashwinee Panda, Siddhant Bharti, Chenfeng Xu, John Heo, Xiaoxia Shirley Wu, James Zou, Percy Liang, Leon Song, Ce Zhang, Ben Athiwaratkun, Zhongzhu Zhou, and Qingyang Wu. CoderForge-Preview: SOTA open dataset for training efficient agents, February 2026. URL <https://www.together.ai/blog/coderforge-preview>. Project core leads: Alpay Ariyak; Zhongzhu Zhou; Qingyang Wu.
- 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, 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. *arXiv preprint arXiv:2505.20411*, 2025.
- Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. Swe-rebench v2: Language-agnostic swe task collection at scale. *arXiv preprint arXiv:2602.23866*, 2026.
- Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, Jeongyeon Nam, and Donghyun Kwak. Online difficulty filtering for reasoning oriented reinforcement learning. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 700–719, Rabat, Morocco, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.eacl-long.30. URL <https://aclanthology.org/2026.eacl-long.30/>.

- Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, Bryan Dai, Jian Yang, and Wayne Xin Zhao. Clawgym: A scalable framework for building effective claw agents, 2026a. URL <https://arxiv.org/abs/2604.26904>.
- Hao Bai, Alexey Taymanov, Tong Zhang, Aviral Kumar, and Spencer Whitehead. Webgym: Scaling training environments for visual web agents with realistic tasks. *arXiv preprint arXiv:2601.02439*, 2026b.
- Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibao Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.
- Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, Zeyao Ma, Kashun Shum, Xuwu Wang, Jinxi Wei, Jiayi Yang, Jiajun Zhang, Lei Zhang, Zongmeng Zhang, Wenting Zhao, and Fan Zhou. Qwen3-Coder-Next technical report. *arXiv preprint arXiv:2603.00729*, 2026.
- Daytona. Daytona: Secure and elastic infrastructure for running AI-generated code. <https://www.daytona.io>, 2025. GitHub repository: <https://github.com/daytonaio/daytona>.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114. Curran Associates, Inc., 2023. URL [https://proceedings.neurips.cc/paper\\_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets\\_and\\_Benchmarks.pdf](https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf).
- E2B. E2B: Open-source secure sandboxes for AI code execution. <https://e2b.dev>, 2024. GitHub repository: <https://github.com/e2b-dev/E2B>.
- Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. davincienv: Open swe environment synthesis at scale, 2026. URL <https://arxiv.org/abs/2603.13023>.
- 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, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, 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.
- 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. *arXiv preprint arXiv:2401.13919*, 2024.
- Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchun Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tiejong Zeng, Liang Wang, Jiwei Li, Yuchen Eleanor Jiang, Wangchunshu Zhou, Guoyin Wang, Keting Yin, Zhou Zhao, Hongxia Yang, Fan Wu, Shengyu Zhang, and Fei Wu. OS agents: A survey on MLLM-based agents for computer, phone and browser use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7436–7465, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.369. URL <https://aclanthology.org/2025.acl-long.369/>.
- Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.
- Alex Kim. Self-host open-source LLM agent sandbox on your own cloud. SkyPilot Blog, <https://blog.skypilot.co/skypilot-llm-sandbox/>, August 2025. SkyPilot Code Sandbox; GitHub: <https://github.com/alex00kim/skypilot-code-sandbox>.
- Thanh-Long V. Le, Myeongho Jeon, Kim Vu, Viet Lai, and Eunho Yang. No prompt left behind: Exploiting zero-variance prompts in llm reinforcement learning via entropy-guided advantage shaping. *arXiv preprint arXiv:2509.21880*, 2025. URL <https://arxiv.org/abs/2509.21880>.
- Chenxin Li, Zhengyang Tang, Mingxin Huang, Yunlong Lin, Shijue Huang, Shengyuan Liu, Bowen Ye, Rang Li, Lei Li, Benyou Wang, and Yixuan Yuan. Claw-eval-live: A live agent benchmark for evolving real-world workflows, 2026. URL <https://arxiv.org/abs/2604.28139>.
- Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. Context as a tool: Context management for long-horizon swe-agents. *arXiv preprint arXiv:2512.22087*, 2025.
- Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanqing Xiong, and Hongsheng Li. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning, 2025. URL <https://arxiv.org/abs/2503.21620>.
- Run Luo, Lu Wang, Wanwei He, Longze Chen, Jiaming Li, and Xiaobo Xia. Gui-r1 : A generalist r1-style vision-language action model for gui agents, 2025. URL <https://arxiv.org/abs/2504.10458>.
- Yougang Lyu, Xiaoyu Zhang, Lingyong Yan, Maarten de Rijke, Zhaochun Ren, and Xiuying Chen. Deepshop: A benchmark for deep research shopping agents. *arXiv preprint arXiv:2506.02839*, 2025.
- 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: Built for real-world productivity. <https://www.minimax.io/news/minimax-m25>, February 2026. HuggingFace: <https://huggingface.co/MiniMaxAI/MiniMax-M2.5>.
- Modal Labs. Modal: High-performance AI infrastructure. <https://modal.com>, 2024.
- Moonshot AI. Kimi Code CLI. URL <https://github.com/MoonshotAI/kimi-cli>. AI agent command-line tool for software development and terminal operations. Accessed 2026-05-06.
- Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiaoyong Wei, Shanru Lin, Hui Liu, Philip S. Yu, and Qing Li. A survey of WebAgents: Towards next-generation AI agents for web automation with large foundation models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 6140–6150. Association for Computing Machinery, 2025. doi: 10.1145/3711896.3736555.
- NVIDIA. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for Agentic reasoning, 2025. URL <https://research.nvidia.com/labs/nemotron/files/NVIDIA-Nemotron-3-Nano-Technical-Report.pdf>. Technical report.
- OpenAI. Introducing SWE-bench Verified. <https://openai.com/index/introducing-swe-bench-verified/>, August 2024. Human-validated subset of 500 instances from SWE-bench, released August 13, 2024.
- OpenClaw. ClawHub: Skill directory for openclaw, 2026. URL <https://github.com/openclaw/clawhub>.
- OpenClaw Team. Openclaw, 2026. URL <https://github.com/openclaw/openclaw>.
- 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. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.
- Qwen Team. Qwen3.5: Towards native multimodal agents. <https://qwen.ai/blog?id=qwen3.5>, February 2026. Open-weights release of Qwen3.5-397B-A17B; HuggingFace: <https://huggingface.co/Qwen/Qwen3.5-397B-A17B>.
- Ning Shang, Yifei Liu, Yi Zhu, Li Lyna Zhang, Weijiang Xu, Xinyu Guan, Buze Zhang, Bingcheng Dong, Xudong Zhou, Bowen Zhang, Ying Xin, Ziming Miao, Scarlett Li, Fan Yang, and Mao Yang. rStar2-Agent: Agentic reasoning technical report. *arXiv preprint arXiv:2508.20722*, 2025. URL <https://arxiv.org/abs/2508.20722>.

- 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>.
- Huatong Song, Lisheng Huang, Shuang Sun, Jinhao Jiang, Ran Le, Daixuan Cheng, Guoxin Chen, Yiwen Hu, Zongchao Chen, Wayne Xin Zhao, Yang Song, Tao Zhang, and Ji-Rong Wen. SWE-Master: Unleashing the potential of software engineering agents via post-training, 2026.
- Atharv Sonwane, Isadora White, Hyunji Lee, Matheus Pereira, Lucas Caccia, Minseon Kim, Zhengyan Shi, Chinmay Singh, Alessandro Sordoni, Marc-Alexandre Côté, and Xingdi Yuan. Bugpilot: Complex bug generation for efficient learning of swe skills. *arXiv preprint arXiv:2510.19898*, 2025.
- Chaofan Tao, Jierun Chen, Yuxin Jiang, Kaiqi Kou, Shaowei Wang, Ruoyu Wang, Xiaohui Li, Sidi Yang, Yiming Du, Jianbo Dai, Zhiming Mao, Xinyu Wang, Lifeng Shang, and Haoli Bai. SWE-Lego: Pushing the limits of supervised fine-tuning for software issue resolving, 2026.
- GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Siboyi, Tianshu Yu, Wei Tian, Weihang Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL <https://arxiv.org/abs/2508.06471>.
- Brandon Trabucco, Gunnar Sigurdsson, Robinson Piramuthu, and Ruslan Salakhutdinov. Insta: Towards internet-scale training for agents, 2025.
- Junhao Wang, Daoguang Zan, Shulin Xin, Siyao Liu, Yurong Wu, and Kai Shen. Swe-mirror: Scaling issue-resolving datasets by mirroring issues across repositories. *arXiv preprint arXiv:2509.08724*, 2025a.
- Weixun Wang et al. Let it flow: Agentic crafting on rock and roll, building the ROME model within an open agentic learning ecosystem, 2026. URL <https://arxiv.org/abs/2512.24873>. Introduces the ROCK (Reinforcement Open Construction Kit) sandbox environment manager as part of the ALE ecosystem; GitHub: <https://github.com/alibaba/ROCK>.
- 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. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025b. URL <https://openreview.net/forum?id=0Jd3ayDDoF>.
- Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Metaclaw: Just talk – an agent that meta-learns and evolves in the wild, 2026. URL <https://arxiv.org/abs/2603.17187>.
- Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. SWE-Fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1123–1139, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.62. URL <https://aclanthology.org/2025.findings-acl.62/>.
- 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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094, 2024.
- Tianyi Xiong, Yi Ge, Ming Li, Zuolong Zhang, Pranav Kulkarni, Kaishen Wang, Qi He, Zeying Zhu, Chenxi Liu, Ruibo Chen, Tong Zheng, Yanshuo Chen, Xiyao Wang, Renrui Zhang, Wenhui Chen, and Heng Huang. Multi-crit: Benchmarking multimodal judges on pluralistic criteria-following, 2026. URL <https://arxiv.org/abs/2511.21662>.
- Yixuan Even Xu, Yash Savani, Fei Fang, and Zico Kolter. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning. *arXiv preprint arXiv:2504.13818*, 2025. URL <https://arxiv.org/abs/2504.13818>.
- 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>.
- John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025a. URL <https://arxiv.org/abs/2504.21798>.
- Zonghan Yang, Shengjie Wang, Kelin Fu, Wenyang He, Weimin Xiong, Yibo Liu, Yibo Miao, Bofei Gao, Yejie Wang, Yingwei Ma, Yanhao Li, Yue Liu, Zhenxing Hu, Kaitai Zhang, Shuyi Wang, Huarong Chen, Flood Sung, Yang Liu, Yang Gao, Zhilin Yang, and Tianyu Liu. Kimi-Dev: Agentless training as skill prior for SWE-agents, 2025b.
- 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.
- 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: Toward trustworthy evaluation of autonomous agents, 2026. URL <https://arxiv.org/abs/2604.06132>.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, 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, Jiaye Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, 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. *arXiv preprint arXiv:2503.14476*, 2025.

- Ji Zeng, Dayuan Fu, Tiantian Mi, Yumin Zhuang, Yaxing Huang, Xuefeng Li, Lyumanshan Ye, Muhang Xie, Qishuo Hua, Zhen Huang, Mohan Jiang, Hanning Wang, Jifan Lin, Yang Xiao, Jie Sun, Yunze Wu, and Pengfei Liu. daVinci-Dev: Agent-native mid-training for software engineering, 2026.
- Liang Zeng, Yongcong Li, Yuzhen Xiao, Changshi Li, Chris Yuhao Liu, Rui Yan, Tianwen Wei, Jujie He, Xuchen Song, Yang Liu, and Yahui Zhou. Skywork-SWE: Unveiling data scaling laws for software engineering in LLMs, 2025.
- ZeroClaw Labs. Zeroclaw, 2026. URL <https://github.com/zeroclaw-labs/zeroclaw>.
- Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained GUI agents: A survey. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL <https://openreview.net/forum?id=xChvYjvXTp>.
- Hao Zhang, Mingjie Liu, Shaokun Zhang, Songyang Han, Jian Hu, Zhenghui Jin, Yuchi Zhang, Shizhe Diao, Ximing Lu, Binfeng Xu, Zhiding Yu, Jan Kautz, and Yi Dong. ProRL Agent: Rollout-as-a-service for RL training of multi-turn LLM agents, 2026a. URL <https://arxiv.org/abs/2603.18815>.
- Lei Zhang, Mouxiang Chen, Ruisheng Cao, Jiawei Chen, Fan Zhou, Yiheng Xu, Jiayi Yang, Liang Chen, Changwei Luo, Kai Zhang, Fan Yan, KaShun Shum, Jiajun Zhang, Zeyu Cui, Hu Feng, Junyang Lin, Binyuan Hui, and Min Yang. MegaFlow: Large-scale distributed orchestration system for the agentic era, 2026b. URL <https://arxiv.org/abs/2601.07526>.
- Zhi Zhang, Zhen Han, Costas Mavromatis, Qi Zhu, Yunyi Zhang, Sheng Guan, Dingmin Wang, Xiong Zhou, Shuai Wang, Soji Adeshina, Vassilis Ioannidis, and Huzefa Rangwala. Train less, learn more: Adaptive efficient rollout optimization for group-based reinforcement learning. *arXiv preprint arXiv:2602.14338*, 2026c. URL <https://arxiv.org/abs/2602.14338>.
- Jiale Zhao, Guoxin Chen, Fanzhe Meng, Minghao Li, Jie Chen, Hui Xu, Yongshuai Sun, Wayne Xin Zhao, Ruihua Song, Yuan Zhang, Peng Wang, Cheng Chen, Jirong Wen, and Kai Jia. Immersion in the GitHub universe: Scaling coding agents to mastery. *arXiv preprint arXiv:2602.09892*, 2026.
- Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025a.
- Haizhong Zheng, Yang Zhou, Brian R. Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. *arXiv preprint arXiv:2506.02177*, 2025b. URL <https://arxiv.org/abs/2506.02177>.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL <https://arxiv.org/abs/2306.05685>.
- 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. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=oKn9c6ytLx>.
- 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/THUHM/slime>, 2025. GitHub repository. Corresponding author: Xin Lv.

## A Orchard Env Design Details

The design of Orchard Env is guided by a central principle: the environment layer should be *thin enough to be reusable* across training recipes, agent harnesses, and model backends, while providing the isolation and lifecycle management needed for large-scale agentic training. We highlight five design choices that realize this principle and, together, satisfy requirements R1–R3 stated in the section opening. Two are *distinguishing technical choices* that set Orchard Env apart from existing environment services: *agent injection* addresses image heterogeneity at near-zero adaptation cost (R2), and *direct Pod-IP communication* keeps the service thin and removes the Kubernetes control plane from the hot path (R1). The remaining three—*network isolation*, *asynchronous lifecycle with heartbeat-based cleanup*, and *watch-based readiness*—are *operational properties* that make the service production-grade at the concurrency and reliability levels required by large-scale agentic training; together with the Kubernetes-native deployment they support R3 (quantified in §2.2). The five paragraphs below present the distinguishing choices first, then the operational properties.

**Agent Injection via Init Containers.** A central challenge in building environment services for agentic training is *image heterogeneity*: different tasks require different base images (e.g., specific Python versions, system libraries, or language toolchains), and modifying each image to include an execution agent is impractical at scale. Orchard Env addresses this through a Kubernetes init container that copies a self-contained Python runtime and agent server into a shared `emptyDir` volume before the main container starts. The main container then launches the agent from the shared volume via `/opt/sandbox-agent/start.sh`. This design avoids baking Python or the agent into each task image; in practice, Orchard Env targets Linux container images and, by default, launches the injected agent through `sh -c`.

**Direct Pod-IP Communication.** After a sandbox is provisioned, all execution and file operation requests are routed directly to the pod IP, bypassing the Kubernetes API server entirely. This avoids the control-plane mediation and WebSocket setup overhead of the Kubernetes exec API. Direct communication reduces per-command round-trip overhead and removes the API server as a throughput bottleneck under high concurrency.

**Network Isolation.** Orchard Env enforces network isolation through Kubernetes NetworkPolicy resources. A namespace-wide default-deny egress policy prevents sandbox containers from initiating outbound connections. When a sandbox requires network access (e.g., for package installation), the orchestrator creates a per-sandbox NetworkPolicy that selectively allows egress, which is cleaned up with the sandbox. This provides defense-in-depth: even if a user-supplied command attempts to exfiltrate data, it is blocked at the network layer.

**Asynchronous Lifecycle with Heartbeat-Based Cleanup.** Sandbox creation is asynchronous: the API returns immediately after pod creation, and clients poll or block on a `/wait` endpoint until readiness. This decouples API responsiveness from Kubernetes scheduling latency. Long-running sandboxes can be kept alive by periodic heartbeat messages from the client SDK. A background cleanup loop in the orchestrator detects sandboxes whose heartbeat has expired and deletes them, preventing resource leakage from crashed or abandoned clients.

**Watch-Based Readiness.** Rather than polling the Kubernetes API for pod status, the orchestrator maintains a persistent LIST+WATCH stream that tracks all sandbox pod state transitions in real time. State changes are cached in memory and waiters are notified via `asyncio.Event`, avoiding repeated polling of the Kubernetes API.

## B Cost Analysis Details

This appendix provides the full methodology and discussion for the cost comparison in Table 2.

**Scenario.** We estimate the cost of running 128 parallel sandbox environments for 240 hours, with each sandbox configured at 2 vCPUs and 8 GiB RAM—matching a typical SWE-bench task environment.

**Orchard setup.** Orchard is deployed on 17 Azure Standard\_D16ads\_v5 instances (16 vCPU, 64 GiB RAM each): 16 nodes host 8 sandboxes each (128 total), and 1 node runs the orchestrator. Sandbox nodes use *spot instances* (preemptible VMs at ~80% discount, ~\$0.165/hr vs. \$0.824/hr on-demand), which are well-suited for ephemeral sandbox workloads that can tolerate occasional preemption. The orchestrator node uses standard pay-as-you-go pricing for stability.

**Managed service pricing.** For managed services, we use their official per-second or per-hour pricing for a 2-vCPU, 8-GiB sandbox, including both compute and memory charges:

- **E2B:** vCPU charge of  $\$0.000014/\text{vCPU}/\text{s} \times 2 = \$0.000028/\text{s}$ , plus RAM charge of  $\$0.0000045/\text{GiB}/\text{s} \times 8 = \$0.000036/\text{s}$ . Total:  $\$0.000064/\text{s} = \$0.2304/\text{hr}$  per sandbox.
- **Daytona:** vCPU charge of  $\$0.0504/\text{vCPU}/\text{hr} \times 2 = \$0.1008/\text{hr}$ , plus RAM charge of  $\$0.0162/\text{GiB}/\text{hr} \times 8 = \$0.1296/\text{hr}$ . Total:  $\$0.2304/\text{hr}$  per sandbox.
- **Modal:** CPU charge of  $\$0.00003942/\text{physical-core}/\text{s} \times 1 \text{ core} (= 2 \text{ vCPU}) = \$0.00003942/\text{s}$ , plus RAM charge of  $\$0.00000672/\text{GiB}/\text{s} \times 8 = \$0.00005376/\text{s}$ . Total:  $\$0.00009318/\text{s} = \$0.3354/\text{hr}$  per sandbox. Modal Sandbox pricing is non-preemptible by default.
- **MegaFlow:** Estimated based on Alibaba Cloud ecs.c8a.2xlarge instances (8 vCPU, 16 GiB, ~\$0.15/hr), one task per instance as described in the original paper. The per-sandbox resource allocation exceeds our 2-vCPU target.

### Key observations.

- **Spot-instance economics.** Orchard’s self-hosted design enables the use of cloud spot instances—preemptible VMs offered at steep discounts (~\$0.165/hr vs. \$0.824/hr on-demand for D16ads\_v5) in exchange for the possibility of short-notice eviction. Because sandbox containers are ephemeral and can be recreated on eviction, spot pricing is a natural fit for the sandbox node pool. Managed services cannot pass through spot pricing because they control the underlying infrastructure.
- **VM-level multiplexing.** Because Orchard packs multiple sandboxes onto each VM (8 per D16ads\_v5 node), the per-sandbox cost benefits from shared overhead. MegaFlow’s one-task-per-instance model, by contrast, allocates a full VM to each sandbox, leading to higher per-sandbox cost even with comparable cloud pricing.
- **On-demand comparison.** Even without spot instances, Orchard’s on-demand cost (\$3,362) is roughly half the cost of E2B and Daytona (\$7,078 each) and one-third of Modal (\$10,305). The key advantage is that Orchard gives researchers full control over the cluster—they can tune node pools, autoscaling, network policies, and resource limits without depending on a vendor’s control plane.

These cost differences compound over the course of a research project. Generating 160K rollout trajectories, running ablation studies, and iterating on training recipes can easily require thousands of hours of environment interaction. Orchard’s self-hosted, spot-friendly design makes such workloads practical for academic research budgets.

## C Orchard-GUI Tool List

This appendix reproduces the full OpenAI tool-call *JSON Schema* for all 13 atomic tools used by Orchard-GUI. At each step the agent emits one or multiples tool calls with each inside a `<tool_call>...</tool_call>` block. For readability, the schemas are grouped by family, one styled box per family.

### Pointer Actions: click / hover / drag

```
{
  "type": "function",
  "function": {
    "name": "click",
    "description": "Performs a mouse click at specific screen coordinates.",
    "parameters": {
      "type": "object",
      "properties": {
        "point_2d": {
          "type": "array",
          "items": {"type": "number"},
          "minItems": 2, "maxItems": 2,
          "description": "The [x, y] pixel coordinates relative to the viewport's top-left
corner."
        },
        "clicks": {
          "type": "integer",
          "description": "The number of consecutive clicks to perform (e.g., 1 for a single
click, 2 for a double click).",
          "default": 1
        },
        "button": {
          "type": "string",
          "enum": ["left", "right", "middle"],
          "description": "The mouse button to trigger.",
          "default": "left"
        }
      },
      "required": ["point_2d"]
    }
  }
},
{
  "type": "function",
  "function": {
    "name": "hover",
    "description": "Move the mouse cursor to a specific coordinate. Use this to trigger hover states,
reveal tooltips, or open dropdown menus.",
    "parameters": {
      "type": "object",
      "properties": {
        "point_2d": {
          "type": "array",
          "items": {"type": "integer"},
          "minItems": 2, "maxItems": 2,
          "description": "The [x, y] pixel coordinates to move the cursor to."
        }
      },
      "required": ["point_2d"]
    }
  }
},
{
  "type": "function",
  "function": {
    "name": "drag",
    "description": "Perform a drag-and-drop operation from a starting screen coordinate to an ending
coordinate.",
    "parameters": {
      "type": "object",
      "properties": {
        "start_point_2d": {
          "type": "array",
          "items": {"type": "number"},
          "minItems": 2, "maxItems": 2,
          "description": "The [x, y] coordinates to start dragging from."
        }
      },
    }
  }
}
```

```

        "end_point_2d": {
            "type": "array",
            "items": {"type": "number"},
            "minItems": 2, "maxItems": 2,
            "description": "The [x, y] coordinates to drop at."
        },
        "steps": {
            "type": "integer",
            "description": "The number of intermediate mouse steps to take. Higher values make the
drag slower and more human-like. Use 10-20 for standard drags.",
            "default": 10
        }
    },
    "required": ["start_point_2d", "end_point_2d"]
}
}
}

```

### Keyboard Actions: write / press\_keys

```

{
  "type": "function",
  "function": {
    "name": "write",
    "description": "Type text using the keyboard. This will first clear any existing content in the
focused input field (using Ctrl+A and Backspace), then type the new text. Make sure to click on an
input field before using this action.",
    "parameters": {
      "type": "object",
      "properties": {
        "message": {
          "type": "string",
          "description": "The text content to type into the focused input field"
        }
      }
    },
    "required": ["message"]
  }
},
{
  "type": "function",
  "function": {
    "name": "press_keys",
    "description": "Press specific keys or key combinations on the keyboard. Use this for navigation,
shortcuts, or discrete key presses.",
    "parameters": {
      "type": "object",
      "properties": {
        "keys": {
          "type": "array",
          "items": {"type": "string"},
          "description": "The list of specific keys to press.\nACCEPTED KEYS:\n- Modifiers:
\\\"Alt\\\", \\\"Control\\\", \\\"Meta\\\", \\\"Shift\\\"\n- Actions: \\\"Enter\\\", \\\"Tab\\\", \\\"Space\\\", \\\"Backspace\\\",
\\\"Delete\\\", \\\"Escape\\\"\n- Navigation: \\\"ArrowUp\\\", \\\"ArrowDown\\\", \\\"ArrowLeft\\\", \\\"ArrowRight\\\",
\\\"Home\\\", \\\"End\\\", \\\"PageUp\\\", \\\"PageDown\\\"\n- Standard: \\\"a\\\"-\\\"z\\\", \\\"0\\\"-\\\"9\\\", \\\"F1\\\"-\\\"F12\\\""
        },
        "is_hotkey": {
          "type": "boolean",
          "description": "If true, all keys in the list are pressed simultaneously (e.g.,
[\\\"Control\\\", \\\"c\\\"]). If false, keys are pressed one by one in sequence.",
          "default": false
        }
      },
      "presses": {
        "type": "integer",
        "description": "Number of times to repeat the press action.",
        "default": 1
      }
    },
    "required": ["keys"]
  }
}
}

```

## Page-Navigation Actions: scroll / goto\_url / go\_back / wait

```

{
  "type": "function",
  "function": {
    "name": "scroll",
    "description": "Scroll the page or a specific scrollable element in the given direction. If point_2d is provided, the scroll occurs on the element at that coordinate (useful for scrollable sub-containers like sidebars, modals, or chat panels). Otherwise, scrolls the main page.",
    "parameters": {
      "type": "object",
      "properties": {
        "direction": {
          "type": "string",
          "enum": ["up", "down", "left", "right"],
          "description": "Direction to scroll.",
          "default": "down"
        },
        "amount": {
          "type": "number",
          "minimum": 0.0, "maximum": 1.0,
          "description": "Scroll amount as a percentage of the viewport height (0.0 to 1.0). For example, 0.5 means scroll by half a page.",
          "default": 0.5
        },
        "point_2d": {
          "type": "array",
          "items": {"type": "number"},
          "minItems": 2, "maxItems": 2,
          "description": "Optional [x, y] pixel coordinates of the element to scroll within. If omitted, scrolls the main page."
        }
      }
    }
  }
},
{
  "type": "function",
  "function": {
    "name": "goto_url",
    "description": "Navigate the current open tab to a specific URL. Use this to open a new webpage directly in the current tab instead of clicking through links or typing in the address bar.",
    "parameters": {
      "type": "object",
      "properties": {
        "url": {
          "type": "string",
          "description": "The full URL to navigate to (e.g., \"https://www.example.com\")."
        }
      },
      "required": ["url"]
    }
  }
},
{
  "type": "function",
  "function": {
    "name": "go_back",
    "description": "Navigate back to the previous page in the browser history. Equivalent to clicking the browser's back button.",
    "parameters": {"type": "object", "properties": {}}
  }
},
{
  "type": "function",
  "function": {
    "name": "wait",
    "description": "Wait for a specified number of seconds before continuing. Use this when you need to allow time for page content to load or for animations to complete.",
    "parameters": {
      "type": "object",
      "properties": {
        "seconds": {
          "type": "integer",
          "description": "Number of seconds to wait (integer value)",
          "default": 3
        }
      }
    }
  }
}

```

```

    }
  }
}

```

### Tab-Management Actions: new\_tab / switch\_tab / close\_tab

```

{
  "type": "function",
  "function": {
    "name": "new_tab",
    "description": "Open a new blank browser tab. Use goto_url after switching to the new tab to navigate to a specific URL.",
    "parameters": {"type": "object", "properties": {}}
  }
},
{
  "type": "function",
  "function": {
    "name": "switch_tab",
    "description": "Switch to a different browser tab by its index (0-based). Use this to navigate between multiple open tabs.",
    "parameters": {
      "type": "object",
      "properties": {
        "tab_index": {
          "type": "integer",
          "description": "The 0-based index of the tab to switch to."
        }
      },
      "required": ["tab_index"]
    }
  }
},
{
  "type": "function",
  "function": {
    "name": "close_tab",
    "description": "Close the current browser tab. If there are multiple tabs open, the browser will switch to the nearest remaining tab.",
    "parameters": {"type": "object", "properties": {}}
  }
}
}

```

### Termination Action: done

```

{
  "type": "function",
  "function": {
    "name": "done",
    "description": "Signal task completion and provide the final answer. Call this function when you have successfully completed the task, when you have determined that the task cannot be completed, or when you encounter a CAPTCHA or anti-bot verification page that blocks further progress. Provide a clear and concise response summarizing what was accomplished or explaining why the task could not be completed (e.g., blocked by CAPTCHA).",
    "parameters": {
      "type": "object",
      "properties": {
        "response": {
          "type": "string",
          "description": "The final answer, result, summary of the task, or reason for task failure. Be specific and include relevant details such as: the information found, actions completed, or reason for task failure."
        }
      },
      "required": ["response"]
    }
  }
}
}

```

## D Example GUI Agent Trajectory

This appendix shows a representative multi-turn rollout produced by Orchard-GUI on a WebVoyager-style task. The first box gives the prompt fed to the model at the final step (system prompt + accumulated trajectory through prior steps). The second box shows the model's response at that step (its final reasoning followed by the terminating done call). The full trajectory of seven *think* → *tool\_call* → *tool\_response* cycles is shown verbatim. Only the JSON tool schema inside the system prompt is abbreviated with [ . . . ] (the full schema appears in Appendix C).

### Inputs to LLM – full trajectory (last turn, 1 context image)

```
<|im_start|>system
You are a GUI agent designed to operate in an iterative loop to automate browser tasks.

# GUI Agent Policy

As an autonomous GUI agent operating on the Web Browser platform, your primary function is to analyze
screen captures and perform appropriate UI actions to complete assigned tasks.

## Core Responsibilities

You can perform web browser interactions including:
- Mouse interactions - click, double-click, right-click, hover, drag, and scroll (page or element)
- Keyboard interactions - type text, press keys, and execute keyboard shortcuts
- Navigation - go to a URL, go back in browser history
- Tab management - open, switch, and close browser tabs
- Task completion - provide responses to queries and terminate tasks with status
- Waiting - allow time for UI changes to occur

## Input Information

At each step, you will receive the following information:

1. Action History: Your interaction history showing all previous actions taken to accomplish the
current task. This helps you track progress and avoid repeating actions.
2. User Request: The primary objective that clearly specifies the task you need to complete. This is
your main goal.
3. Observation: Current state information about the web page, including:
- Tab Info: The currently active tab index and a list of all open tabs with their index, URL, and
page title
- Screenshot: Visual representation of the current page state
- A11y Tree (optional): Accessibility tree containing interactive elements with their IDs, types,
labels, and positions

## Output Requirements

- Your output must include two tags: one <think> and one or more <tool_call> blocks.
- Always start your response with <think>.

## Guidelines

- Reasoning process: In your <think> block, you should analyze the current state (e.g., what do you
see on the screen), reflect on your previous actions (e.g., did they produce the expected result, or
did something go wrong), assess progress toward the goal, plan your next steps, and validate that
your planned actions are safe and correct. If you notice you've been repeating the same action
without progress, consider an alternative approach.
- Valid and executable tool calls only: All tool calls must exist within the defined tool set, and
must be valid, executable tool calls.
- Use multiple tool calls when appropriate: If a task step naturally involves a short chain of actions
on the current page (e.g., "click -> write -> press Enter" or "new_tab -> goto_url"), emit them all
in one response with multiple <tool_call> blocks -- one per action, each on its own line.
- Sequential execution: When using multiple <tool_call> blocks, they are executed in order from top
to bottom. Ensure the sequence is logically correct -- later actions may depend on earlier ones
completing successfully.

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "click", "description": "Performs a mouse click at specific
screen coordinates.", "parameters": {...}}}
```

```

{"type": "function", "function": {"name": "write", "description": "Type text using the keyboard. ...",
  "parameters": {...}}

[... 11 additional tool schemas omitted; see Appendix~\ref{app:browser:tools} for the full list ...]

</tools>

For each function call, return a json object with function name and arguments within
  <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
<|im_end|>

<|im_start|>user
<user_request>
Find a dog bed on Amazon that is washable and has a length of at least 30 inches.
</user_request>
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/ - Amazon.com
</observation>
<|im_end|>

<|im_start|>assistant
<think>
The current screen shows an Amazon verification page that requires clicking a button to continue shopping.
  This is a standard anti-bot measure. I need to click the "Continue shopping" button to proceed to
  the main Amazon page where I can search for dog beds.

My plan:
1. Click the "Continue shopping" button to get past this verification page
2. Once on the main Amazon page, search for "dog bed washable 30 inches"
3. Apply filters for washable beds with length >= 30 inches
4. Review results and find a suitable product

The next logical action is to click the "Continue shopping" button to proceed with the task. This is
  necessary before I can access the search functionality of Amazon.
</think>

<tool_call>
{"name": "click", "arguments": {"point_2d": [501, 213]}}
</tool_call>
<|im_end|>

<|im_start|>user
<tool_response>
Succeded: `click` on <button> type=submit "Continue shopping" at (641, 213) executed. Note: no visible
  navigation or new tab detected.
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/ - Amazon.com. Spend less. Smile more.
</observation>
</tool_response>
<|im_end|>

<|im_start|>assistant
<think>
The previous action successfully got me past the Amazon verification page, and now I'm on the main Amazon
  homepage. I can see the search bar at the top of the page which I need to use to search for dog beds.

My plan remains:
1. Search for "dog bed washable 30 inches" in the Amazon search bar
2. Apply filters for washable beds with length >= 30 inches
3. Review results and find a suitable product

The next logical step is to click on the search bar so I can type my search query. The search bar is
  clearly visible at the top of the page with the placeholder text "Search Amazon". Once I click on
  it, I'll be able to type my search terms for dog beds that are washable and at least 30 inches long.

I should click on the search bar to focus it before typing my search query.
</think>

<tool_call>
{"name": "click", "arguments": {"point_2d": [385, 31]}}

```

```

</tool_call>
<|im_end|>

<|im_start|>user
<tool_response>
Succeed: `click` on <input> type=text role=searchbox "Search Amazon" at (493, 31) executed. Note: no
  visible navigation or new tab detected.
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/ - Amazon.com. Spend less. Smile more.
</observation>
</tool_response>
<|im_end|>

<|im_start|>assistant
<think>
The previous action successfully clicked on the Amazon search bar, and I can see it's now focused and
  ready for text input. The cursor is blinking in the search field, indicating it's ready to receive
  my search query.

My plan is still:
1. Search for "dog bed washable 30 inches" in the Amazon search bar
2. Apply filters for washable beds with length >= 30 inches
3. Review results and find a suitable product

Now that the search bar is focused, the next logical step is to type my search query. I should enter "dog
  bed washable 30 inches" to find products that match all the criteria specified in the user request.
  This search term includes all the key requirements:
- It's for a dog bed
- It specifies "washable" as a required feature
- It includes "30 inches" to help find beds of the appropriate size

After typing this search query, I'll need to press Enter to execute the search, which will take me to the
  search results page where I can further refine the results using filters.
</think>

<tool_call>
{"name": "write", "arguments": {"message": "dog bed washable 30 inches"}}
</tool_call>
<|im_end|>

<|im_start|>user
<tool_response>
Succeed: `write` typed "dog bed washable 30 inches" into <input> type=text role=searchbox "Search Amazon".
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/ - Amazon.com. Spend less. Smile more.
</observation>
</tool_response>
<|im_end|>

<|im_start|>assistant
<think>
The previous action successfully typed "dog bed washable 30 inches" into the Amazon search bar, and I can
  see the search suggestions dropdown has appeared with various related search terms. The search query
  is ready to be submitted.

My plan is still:
1. Execute the search for "dog bed washable 30 inches"
2. Apply filters for washable beds with length >= 30 inches
3. Review results and find a suitable product

Now that I've entered the search query, the next logical step is to submit the search. I can do this by
  clicking the search button (the magnifying glass icon) next to the search bar. This will take me to
  the search results page where I can see all the dog beds that match my criteria.

The search button is visible as an orange button with a magnifying glass icon at the right end of the
  search bar. Clicking this button will execute the search and show me the results for dog beds that
  are washable and at least 30 inches long.

After the search results load, I'll need to check if the results match the criteria and potentially apply
  additional filters to narrow down the options to find the best match for the user's requirements.
</think>

<tool_call>

```

```

{"name": "click", "arguments": {"point_2d": [649, 31]}}
</tool_call>
<|im_end|>

<|im_start|>user
<tool_response>
Succeed: `click` on <input> type=submit at (831, 31) executed. Page navigated to
  https://www.amazon.com/s?k=dog+bed+washable+30+inches&crd=3M6EI7976XXX6&srefix=dog+bed+...
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active):
  https://www.amazon.com/s?k=dog+bed+washable+30+inches&crd=3M6EI7976XXX6&srefix=dog+bed+... -
  Amazon.com : dog bed washable 30 inches
</observation>
</tool_response>
<|im_end|>

<|im_start|>assistant
<think>
The search has been successfully executed and I'm now on the Amazon search results page for "dog bed
washable 30 inches". The page shows over 10,000 results, which is quite a lot to manually review. I
can see several dog beds in the results, and I need to find one that meets the specific criteria:
washable and at least 30 inches in length.

Looking at the left sidebar, I can see there's a "Washable" filter option under "Special Features" that I
should apply to narrow down the results. This will help ensure that all the results shown are for
washable dog beds, which is one of the key requirements.

I also notice that the first product listed (Waterproof Dog Bed Medium Size Dog) mentions "30 inch" in its
description, which appears to meet the length requirement. However, to be thorough, I should apply
the washable filter first to ensure all results meet that criteria, and then I can look for beds
that are at least 30 inches long.

My updated plan:
1. Apply the "Washable" filter by clicking the checkbox in the left sidebar
2. After filtering, review the results to confirm they meet the length requirement (at least 30 inches)
3. Select a suitable product that meets both criteria

The next logical action is to click the "Washable" checkbox under Special Features to filter the results
to only show washable dog beds. This will help narrow down the options and make it easier to find
products that meet the user's requirements.
</think>

<tool_call>
{"name": "click", "arguments": {"point_2d": [27, 509]}}
</tool_call>
<|im_end|>

<|im_start|>user
<tool_response>
Succeed: `click` on <i> at (35, 509) executed. Page navigated to
  https://www.amazon.com/s?k=dog+bed+washable+30+inches&rh=p_n_g-1003349998111...
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/s?k=dog+bed+washable+30+inches&rh=p_n_g-1003349998111...
</observation>
</tool_response>
<|im_end|>

<|im_start|>assistant
<think>
The washable filter has been successfully applied, and now I'm seeing 464 results for dog beds that are
washable. The page shows several options that appear to meet the criteria, but I need to verify that
they have a length of at least 30 inches.

Looking at the first product, the FURTIME Washable Dog Bed, it mentions "Large Sized Dogs" but doesn't
explicitly state the dimensions in the visible text. The second product, EHEYCIGA Washable Large Dog
Bed, clearly states "35x23 Inch" which means it has a length of 35 inches - meeting the requirement
of at least 30 inches.

I should check the product details to confirm the dimensions and washability. The EHEYCIGA bed is a good
candidate since it explicitly states 35 inches in length, which exceeds the 30-inch requirement, and
it's already filtered as washable.

```

```

My next step should be to click on this product to view more details and confirm it meets all the
requirements. The EHEYCIGA Washable Large Dog Bed (35x23 Inch) appears to be a strong candidate as
it:
1. Is washable (confirmed by the filter)
2. Has a length of 35 inches (exceeding the 30-inch requirement)

I'll click on this product to verify the details and see if it's a suitable match for the user's request.
</think>

<tool_call>
{"name": "click", "arguments": {"point_2d": [496, 514]}}
</tool_call>
</im_end>

<|im_start|>user
<tool_response>
Succeed: `click` on <span> "EHEYCIGA Washable Large Dog Be..." at (635, 514) executed. Page navigated to
https://www.amazon.com/EHEYCIGA-Washable-Reversible-Sleeping-Mattress/dp/B0FG7XNSYH...
<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.amazon.com/EHEYCIGA-Washable-Reversible-Sleeping-Mattress/dp/B0FG7XNSYH/...
- Amazon.com : EHEYCIGA Washable Large Dog Bed for C
screenshot:
<|vision_start|><|image_pad|><|vision_end|>
</observation>
</tool_response>
</im_end|>

<|im_start|>assistant
<think>

```

### Example response from LLM at the last turn

```

I've successfully navigated to the product page for the EHEYCIGA Washable Large Dog Bed. This product
meets all the requirements specified in the user request:

1. It's washable - confirmed by the product name and description ("Washable Large Dog Bed")
2. It has a length of 35 inches - explicitly stated in the product title as "35x23 Inch" and confirmed in
the size specifications at the bottom of the page showing "35" x 23" x 3.5"

The product details confirm it's designed for dogs up to 45 lbs and is labeled as "Amazon's Choice" with
good reviews (4.7 stars from 381 reviews). It's also marked as having sustainability features.

I've found a dog bed that satisfies both key requirements:
- Washable (as required)
- At least 30 inches in length (it's 35 inches, which exceeds the minimum requirement)

The task has been successfully completed. I've found a suitable product that meets all the criteria
specified in the user request. There's no need for further actions as I've identified a product that
matches the requirements.

I should now signal that the task is complete and provide the details of the found product as the answer.
</think>

<tool_call>
{"name": "done", "arguments": {"response": "I found a suitable dog bed on Amazon that meets your
requirements: The EHEYCIGA Washable Large Dog Bed (35\" x 23\" x 3.5\") is washable and has a length
of 35 inches, which exceeds your minimum requirement of 30 inches. It's priced at $17.99, has 4.7
stars from 381 reviews, and is labeled as 'Amazon's Choice'. This bed is designed for dogs up to 45
lbs and is currently in stock."}}
</tool_call>
</im_end|>

```

## E Orchard-GUI Task Filtering Pipeline

We draw task instances from the task set organized by WebGym (Bai et al., 2026b), which consists of 292,092 raw task instances in total. To produce a clean, evaluation-safe, and diverse pool of training prompts, we apply a five-stage filtering pipeline:

1. *Remove common evaluation benchmarks.* We strip out splits that overlap with our held-out benchmarks (e.g., Online-Mind2Web (Deng et al., 2023) and DeepShop (Lyu et al., 2025)) to prevent train/test contamination, retaining only the two complementary PAE-WebVoyager (Zhou et al., 2025) and InSTA-v3 (Trabucco et al., 2025) splits (-13,840, 4.7% → **278,252**). The former consists of automatically proposed web-navigation tasks generated by a context-aware task proposer, grounded in the websites covered by the WebVoyager (He et al., 2024) benchmark, while the later contains tasks automatically synthesized by an LLM over a large and diverse set of websites. Each task is grounded in a specific domain and phrased as a realistic user goal (e.g., finding information, retrieving attributes, or completing simple workflows), with an emphasis on feasibility and safety.
2. *Keep parent tasks only.* WebGym additionally provides child tasks decomposed from each parent intent. Since child tasks share substantial structure with their parents, we retain only the parents to avoid intra-family redundancy (-23,437, 8.4% → **254,815**).
3. *Exclude WebVoyager tasks.* We further drop any task whose intent appears in the original WebVoyager benchmark, eliminating residual contamination at the prompt level (-411, 0.2% → **254,404**).
4. *Restrict to popular websites.* Long-tail websites are noisier (more captchas, anti-bot blocks, broken pages) and less representative of realistic browsing. We keep only tasks whose target site falls within the SimilarWeb Top-100 list and the MOZ Top 500 Most Popular Websites, and where the same site has at least two tasks, ensuring sufficient per-site coverage for the downstream agent (-114,349, 44.9% → **140,055**).
5. *Semantic deduplication.* The remaining pool is dominated by near-duplicate intents (e.g., paraphrases of the same shopping or search query across thousands of products). We embed each task intent with Qwen/Qwen3-Embedding-8B and greedily remove tasks whose cosine similarity to a previously kept task exceeds 0.99 (-124,454, 88.9% → **15,601**).

The final filtered pool of **15,601** unique task intents serves as the seed set from which we sample teacher trajectories for SFT and RL prompts.