ECHOVERSE: DEEP, EVOLVING ENVIRONMENTS FOR TRAINING COMPUTER-USE AGENTS AT SCALE

Anonymous authors
Paper under double-blind review

ABSTRACT

Training computer-use agents requires software application environments that the agents can interact with. Sandbox environments offer a better alternative to real ones, since they can be deterministic and safer to take actions in. However, creating these environments with sufficient depth and breadth of functionalities for training agents is non-trivial. We present Echoverse, a semi-automated pipeline that builds synthetic environments from specifications and target scenarios, with task outcomes verified against references derived from the environments' database. Echoverse works by co-evolving the environment and the model together in a loop. Specifically, for each step, we perform rollouts in our environments and use verifiers to discern between model successes/failures and environment issues. Based on the verifier analysis, we improve the environment and use the rollouts for training the model. By adjusting the environment with feedback from the rollouts, we are able to add functionalities needed to enable more tasks and increase the fidelity to real applications. We use Echoverse to produce a set of 10 environments covering different software use-cases. We show that fine-tuning on rollouts from environments verified only by unit tests and simple checks leads to poor transfer to real environments compared to fine-tuning on our environments (*e.g.,* 48.0% vs 65.0% accuracy). Building on these results, we introduce an RL training recipe that combines database grounded trajectory rewards with dense per-step feedback to address sparse supervision over long action sequences. We show that this recipe improves performance significantly in our synthetic environments (*e.g.,* 64.83% to 77.59% pass@4).

1 INTRODUCTION

Computer-Use Agents (CUAs) put language models in control of applications that manage our messages, schedules, and shared records. Rescheduling a meeting may require finding information in an email thread, updating the right calendar event, and sending revised invitations to its participants. Each step depends on earlier information or state, and success requires carrying through the requested changes. As a result, training CUAs using these attempts requires checking the resulting state and restoring the application before trying again.

Using real software applications for training environments can make this repeated interaction infeasible. In most cases, we do not have access to inspect or control the underlying storage/state of the application, and the state may be non-deterministic and inconsistent between attempts. This also presents safety concerns as exploratory actions from the agent can have negative consequences, such as sending unwanted messages.

Synthetic environments support repeated training through control of the data, isolated rollouts, and state control. Recent pipelines generate such environments and tasks at scale (Zhou, 2026; Zhang et al., 2026; Aggarwal et al., 2026; Wang et al., 2026), expanding the options available for agent training. Their value largely depends on how well they implement functioning workflows that emulate real applications and how well the tasks are grounded to the available functions in the environment. In the meeting example, if the environment lacks proper search functions common to email apps, then agents may learn a bad policy of manually sifting through messages or inputting ineffective queries. Likewise, the task asking for a search is ill-posed with respect to the environment. Such defects can provide the wrong learning signal to the agent being trained or obfuscates the source of issues while developing these agents. We therefore treat the **environment**, its **task corpus**, and its **verifier** as a single *world* whose components must be checked together.

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

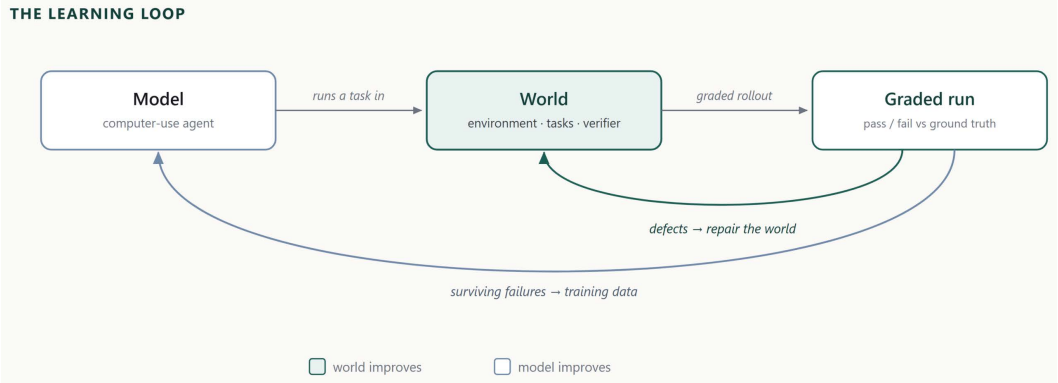


Figure 1: Environment development and model training. Rollout analysis identifies application, task, and verifier defects for repair. Failures on valid tasks guide subsequent training-task selection. SFT demonstrations are collected separately and retained only when they pass outcome verification.

We build such worlds around three connected design goals. **Depth** requires the controls and internal state dependencies of a target workflow to function end to end to meet the requirements of our task corpus. With these workflows in place, **capability targeting** provides training instances for challenging interactions across particular types of widget, layout, or constraints in which agents are known to struggle. Agent rollouts can then reveal gaps in the *world* that the initial checks missed. **Co-evolution** keeps the *world* (environment, tasks, and verifier) internally consistent as each piece evolves with the others.

To accomplish this, the Echoverse factory builds stateful environments from specifications and target scenarios, deriving task queries and reference solutions from their databases. A verifier compares agent answers and database changes with these references to provide a signal for correctness. Failed rollouts guide repairs to application defects, infeasible tasks, and incorrect verification, which are checked again; failures on valid tasks guide data collection and model training (Figure 1). Using this factory, we produce 10 worlds, spanning different applications, such as email, banking, and machine learning resource management. These worlds are complete with task sets, functioning front-ends as well as back-ends, and task verification setups to support training CUAs.

We first assess our design choices through supervised fine-tuning (SFT) (Sec. 5). On two domains, SFT on trajectories from our environments can transfer to live websites and improve performance. Meanwhile, training on trajectories from environments developed without co-evolution (*i.e.,* fixed task set, test-case driven dev loop) yields no gains or reduces it. We also find that training on trajectories collected from later iterations improves the resulting accuracy (*e.g.,* ECHOSTAY environment raises accuracy from 16.2% to 38.5%).

Our worlds support Reinforcement Learning (RL) with mechanisms for reliable resets and cheap verification. Towards this, we create an RL recipe to effectively train CUAs in our environments using on-policy rollouts. For long horizon trajectories, a terminal score alone gives little guidance about which intermediate actions helped or failed. Our RL recipe therefore combines database-grounded trajectory rewards with dense, model-judged per-step feedback and group-relative updates. The trajectory term assesses task completion, while the step term supplies action-level feedback.

Overall, our contributions are:

1. **Environment construction and co-evolution.** We define task-corpus derived depth and introduce a two-phase pipeline for building stateful applications and feasible tasks. The suite spans ten domains and two capability worlds; rollout analysis guides repairs to applications, tasks, and verifiers alongside policy training (Secs. 3–4).
2. **Evidence for environment design.** Our SFT studies demonstrate benefits of environment depth, targeted widget practice, and repaired training worlds. We also evaluate task and reference co-evolution in RL. Separate scaling studies examine how gains from trajectory volume saturate faster than gains from increasing domain coverage. (Secs. 5 and 6).
3. **An RL training recipe.** We combine task selection, trajectory rewards, and dense per-step feedback with group-relative policy optimization, using database-grounded verification for synthetic environments and multimodal verification for live-web continuation (Sec. 6).

4. **Reproducible evaluation resources.** We release four worlds: ECHOSTAY, ECHOFORGE, and the datepicker and nested-filter worlds, with their applications, seed data, test tasks, and verifiers (Sec. 7).

2 RELATED WORK

Environments and benchmarks. Computer-use benchmarks expose agents to both public websites and controlled applications. WebVoyager, Mind2Web, and Online-Mind2Web draw their tasks from the public web (He et al., 2024; Deng et al., 2023; Xue et al., 2025), while WebArena, VisualWebArena, WorkArena, OSWorld, and AndroidWorld provide controlled settings with task checks (Zhou et al., 2024; Koh et al., 2024; Drouin et al., 2024; Xie et al., 2024; Rawles et al., 2025); BrowserGym brings several environments behind a shared interface (de Chezelles et al., 2025). Recent pipelines expand this coverage by automating application or task generation (Zhou, 2026; Zhang et al., 2026; Aggarwal et al., 2026; Wang et al., 2026). As generated applications become sources of training experience, their workflow completeness and reliability become important properties of the resulting data. Echoverse examines these properties through workflow depth, variation in difficult UI controls, and repairs guided by agent rollouts.

Reinforcement learning for computer-use agents. RL uses repeated interaction to improve the policy itself, a direction explored for browser control before current vision-language agents (Shi et al., 2017). For complete tasks, this requires choosing what the agent attempts and judging the resulting outcomes. DigiRL combines offline-to-online Android learning with a VLM evaluator (Bai et al., 2024). On the web, WebRL couples an evolving task curriculum with an outcome reward model (Qi et al., 2025), whereas WebAgent-R1 uses binary task-completion feedback for multi-turn interaction (Wei et al., 2025). ZeroGUI automates both task generation and reward estimation (Yang et al., 2025), and MobileGUI-RL combines a task curriculum with trajectory-aware advantages and rewards for success and efficiency (Shi et al., 2025).

Scaling this interaction introduces demands on both rollout infrastructure and the supply of training tasks. ComputerRL, UI-TARS-2, and DART address multi-turn optimization and rollout systems (Lai et al., 2025; Wang et al., 2025a; Li et al., 2025). Verifiable synthetic tasks support UltraCUA's combination of SFT and online RL in a hybrid action space of GUI operations and high-level tools (Yang et al., 2026b). SCALECUA and EvoCUA-1.5 integrate verifiable task generation and selection with online RL (Lv et al., 2026; Huang et al., 2026). These efforts also span different application settings: Qwen-CUA scales verifiable trajectory training across software domains (Lu et al., 2026), while BlueLM-GUI trains on real mobile devices (Ye et al., 2026).

The granularity of supervision shapes what an agent can learn from each attempt. UI-R1, GUI-R1, InfiGUI-G1, and AgentCPM-GUI study action prediction or grounding with rule-based rewards (Lu et al., 2025; Luo et al., 2025; Liu et al., 2025b; Zhang et al., 2025). InfiGUI-R1 includes subgoal and recovery supervision (Liu et al., 2025a), while GUI-Libra addresses partial verifiability in offline step-wise RL (Yang et al., 2026a). For multi-turn interaction, PRO-CUA optimizes candidate actions using a process reward model (He et al., 2026), and VAGEN supplies turn-level world-modeling rewards (Wang et al., 2025b). Echoverse builds on this work by combining database-referenced outcome checks with per-step feedback for complete executions in repairable synthetic applications. Appendix A compares the closest training settings; differences in task sets prevent direct comparison of cross-paper scores.

Verification. The reliability of these rewards depends on the evidence available to the verifier. WebArena and τ-bench use application state to evaluate outcomes (Zhou et al., 2024; Yao et al., 2024). Without direct state access, trajectory judges estimate success from observations and actions (Rosset et al., 2026). OSReward studies the reliability of such judgments (Sun et al., 2026), while autonomous-evaluation RL explicitly models evaluator noise (Sumyk & Kosovan, 2026). Evidence collection offers another way to support verification: SeekJudge searches trajectory evidence (Wan et al., 2026), and Interactive Reward Agent gathers evidence from the post-execution environment (Shi et al., 2026). Echoverse uses database-derived task references and before/after database state to check the requested outcomes. This access grounds the comparison in stored results, although the semantic judgment remains model-based.

3 PROBLEM SETTING: ENVIRONMENTS, TASKS AND GROUNDED GRADING

An Echoverse *world* brings together the environment the agent acts on, the tasks it must complete, and the verifier that evaluates its work. The agent interacts with the environment through the browser to retrieve information, change stored records, or both. Each task pairs a user goal with a reference derived from the environment's database, so the result of an attempt can be checked against a specified outcome.

We define this *world* as $W = (E, \mathcal{T}, V)$, comprising an environment E, task corpus $\mathcal{T}$, and verifier V. The environment $E = (\mathcal{S}, \mathcal{A}, T, \mathcal{O}, D_0)$ has database state $\mathcal{S}$, browser actions $\mathcal{A}$, transition function T, observations $\mathcal{O}$, and seeded database D_0, where persistent task outcomes are recorded. Each task $\tau = (g, r, \kappa)$ specifies a goal g, a database-derived reference r, and a kind $\kappa \in \{\text{READ}, \text{WRITE}, \text{READ_WRITE}\}$.

3.1 GROUNDED GRADING

During task generation, SQL queries over the seeded database D_0 produce the reference used for grading: an expected answer for a read task or an intended state change for a write task. Once a rollout ends, the verifier compares the agent's answer with the read reference or checks the `sqldiff` between D_0 and the final database D_T against the write reference. Each comparison returns a binary score. A read–write task receives the minimum of the two scores, so both checks must pass.

An LLM judge performs the semantic comparison from the task goal, reference, and observed answer or database diff. It receives no screenshots or action history. Its reliability depends on the database-derived reference, the judge prompt, and the model. The prompts also allow specified numeric formatting differences and additional writes, so a passing verdict does not completely rule out unintended side effects. Appendix G gives the exact prompts.

3.2 TASK-RELATIVE ENVIRONMENT DEPTH

We define depth relative to a target set of tasks and scenarios specified before environment construction. An environment is sufficiently deep for this set when the required workflows, state dependencies, and relevant visible controls function end to end, and task outcomes can be checked from the environment's database state. For example, a search task may be solvable by inspecting every page of a list. If the intended search control is broken, the environment still fails the workflow requirement. A single successful execution therefore does not establish depth.

These requirements determine both what must be built and what must be checked. The factory in Sec. 4.1 turns them into tightly coupled *worlds* and uses agent rollouts to identify what still needs repair in them.

4 ECHOVERSE

4.1 THE FACTORY AND THE CO-EVOLUTION LOOP

Echoverse produces full-domain worlds with realistic workflow depth and structure, and capability worlds that isolate and vary diagnosed interactions or capabilities. Owning their databases lets us check outcomes against state rather than infer them from screenshot trajectories.

The factory runs in two phases (Figure 2). In Phase 1, hand-written seeds are expanded into scenarios and task proposals that inform the specification. *Claims* are structured assertions of expected environment properties or behaviors, derived from the specification and seed scenarios before frontend, backend, and database generation. Frontier LLM agents verify claims by inspecting code, querying the database using MCP tools, and browsing the hosted environment. Related issues are triaged for layer-specific repair, and claims are rechecked through agent-based testing.

Phase 2 grounds task proposals in the live database and binds each task to a user profile. Agents assess task quality and solvability using code, database evidence, and browser trajectories. Findings guide task, database, backend, and frontend repairs. After verified fixes, tasks are reassessed before computing pass rates.

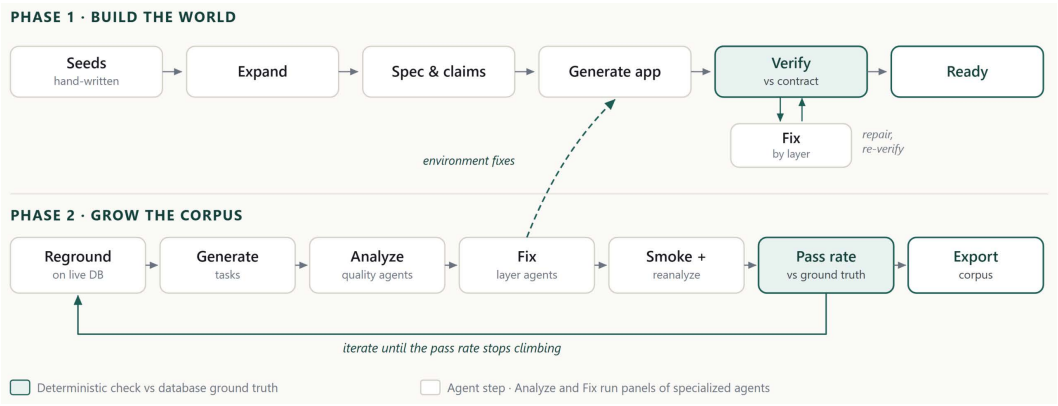


Figure 2: The two-phase factory. Frontier LLM agents verify claims and tasks; failures return to layer-specific repair, and corpus repairs can propagate back to the environment (dashed arrow).

Both phases share this verification, triage, and repair cycle, which continues after a world ships as failures of the trained model provide further evidence.

Building and repairing worlds. Builder, verifier, triage, and fixer agents use engineering, browser-use, and database tools on the running environment under human QA. Acceptance requires 95% of Phase 1 claims and 95% of surviving Phase 2 tasks to be judged passing, with no hard blocker. Agents assess task validity, plausibility, interface feasibility, and grading correctness. Runs stopped by budget, iteration, or plateau limits remain incomplete. Isolated per-task database copies provide exact resets without cross-run contamination. Appendix B gives the roles, checks, thresholds, seeding, and reset mechanics.

One rollout, two signals. A score depends on the model, environment, task, and verifier, so failures return to the factory's verifier and triage agents. World defects are repaired before training, without weakening goals. Failures on valid, solvable, and correctly graded tasks provide capability signals. Grouping failures by cause lets one control repair restore all tasks it blocked. Appendix B.5 details this procedure.

4.2 The environment suite

We select diverse domains with workflows users want to automate, especially login-gated or write-heavy activities impractical to exercise repeatedly on live services. The ten full-domain environments cover communication (EchoMail, EchoCalendar, EchoChat), technical workflows (EchoML, EchoForge), transactions and records (EchoBank, EchoCare), and community, media, and travel (EchoForum, EchoTunes, EchoStay). Tasks read and write shared records, exercising permissions, histories, and dependencies between workflow steps. Appendix C describes the environments and seed data.

Two capability worlds. The capability worlds vary datepicker and nested-filter controls across 100/80 and 200/100 training/held-out frontends, respectively. Held-out widget families are excluded from training, testing transfer beyond the training controls. Tasks are scored against environment state or the environment's own filtering logic. Appendix H summarizes the control types.

5 How environment design shapes agent learning

We examine how task-relative depth (Sec. 3.2), variation in difficult controls, and environment repair affect learning and transfer, using SFT as a common training procedure. Separate scaling sweeps vary trajectory volume and environment coverage.

Setup. GPT-5.4 attempts each task in the live environment; the grounded verifier retains passing trajectories for SFT (Zelikman et al., 2022; Yuan et al., 2023). This rejection-sampled distillation

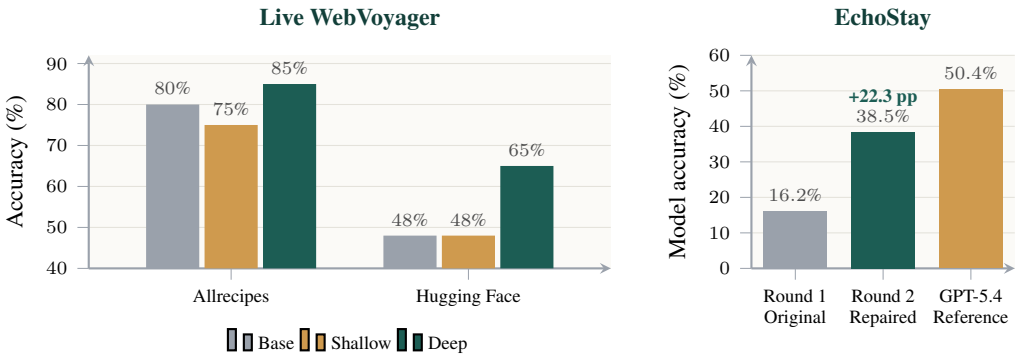


Figure 3: Environment depth and co-evolution. Left: live-site accuracy after SFT on trajectories from shallow or deep synthetic versions of two domains. Trajectories from deep environments improve accuracy on both. Right: ECHOSTAY model accuracy after SFT on trajectories from the original and repaired environments. Evaluation is held fixed; task feasibility is measured separately.

follows feasibility checks: teacher failures contribute no demonstration but do not invalidate feasible tasks. Appendix D.1 describes collection; Appendix E gives the scaling-corpus composition.

π_{base} is Qwen3.5-9B (Qwen Team, 2026) trained for the browser action space with only a few hundred synthetic trajectories, all from ECHOMAIL. We use Fara1.5's action space and CUA harness (Awadallah et al., 2026). Each study independently fine-tunes Qwen3.5-9B on its designated subset plus the data used to train π_{base}. Checkpoints are served with vLLM (Kwon et al., 2023). We set a step budget of 100 for the rollouts. Synthetic tasks use our grounded verifier (Sec. 3.1), while WebVoyager and Online-Mind2Web use their published judges (details in Appendix D.2).

Environment depth. We compare comparably sized trajectory corpora from "shallow" and "deep" synthetic versions of Allrecipes and Hugging Face. Here, "shallow" means the environment is built using a similar pipeline to Echoverse, but with a fixed task set and a test-driven development loop Zhang et al. (2026). The shallow environments have lower frontend fidelity and more limited feature support than the deep environments. On live WebVoyager tasks, SFT on trajectories from shallow environments changes Allrecipes accuracy from 80.0 to 75.0 and leaves Hugging Face at 48.0; trajectories from deep environments yield 85.0 and 65.0, respectively (Figure 3, left).

Environment Co-Evolution. In ECHOSTAY, rollouts exposed a guest-count control defect and some nonfunctional listing-filter widgets. Repairing the guest-count control recovers 15 of 24 blocked bookings and raises task feasibility from 48% to 78%. We compare checkpoints trained on trajectories collected from the original and repaired environments. **Both checkpoints are evaluated on the same improved test set from the repaired *world*, using the same environment and verifier.** Model accuracy increases from 16.2% to 38.5% (Figure 3, right).

Targeting Capability Gaps. We test transfer to held-out widget families by adding capability-world trajectories to the alignment data. SFT with both worlds raises held-out datepicker accuracy from 34.0 to 57.3, nested-filter accuracy from 62.8 to 84.8, and Online-Mind2Web from 29.5 to 34.3. These results support practice across varied layouts and constraints. Table 1 reports the separate and joint training conditions.

Table 1: Capability ablation: four models over four splits (task success, %). Best per row in bold. "Held out" widget families never appear in training.

Split	Base	+Datepicker	+Nested filter	+Both
Datepicker, in-distribution	60.0	82.6	73.1	**83.5**
Datepicker, held out	34.0	54.0	50.7	**57.3**
Nested filter, in-distribution	76.0	78.0	84.0	**88.0**
Nested filter, held out	62.8	82.1	84.1	**84.8**
Mean	58.2	74.2	73.0	**78.4**

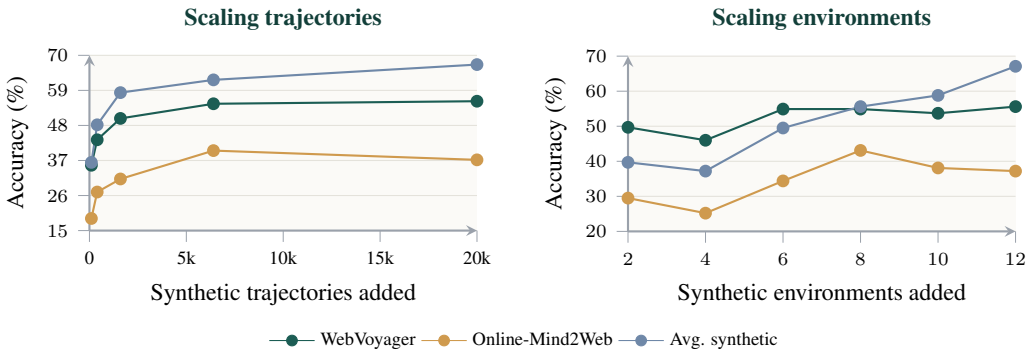


Figure 4: SFT scaling: trajectory volume at fixed per-world proportions (left) and environment coverage (right). Synthetic accuracy improves in both sweeps; live-web transfer is non-monotonic and varies by benchmark.

Data volume and environment coverage. The two sweeps reveal different saturation patterns (Figure 4). At a fixed per-world mixture, more trajectories bring strong early gains but diminishing returns at larger volumes: synthetic accuracy improves more slowly, while live-web performance largely saturates. In the separate environment-coverage sweep, adding new domains sustains gains in synthetic accuracy across the larger mixtures, with no comparable plateau. Broader coverage thus remains a useful scaling direction as the returns from additional trajectories diminish. Live-web gains are less sustained: WebVoyager largely plateaus after six worlds, while Online-Mind2Web peaks at eight before declining. The synthetic tasks are largely write-oriented, creating a task-distribution mismatch with these live benchmarks. Most live-web gains therefore reflect out-of-domain transfer, which helps explain their modest size. Appendix D.5 gives the sweep details.

6 Reinforcement learning in Echoverse and on the live web

The preceding studies examine what agents learn from demonstrations collected in different environments. We now use Echoverse for reinforcement learning, where the policy learns from its own attempts. Database-grounded checks assess the final outcome, while per-action feedback provides guidance within an execution. We first train across six synthetic environments, evaluate transfer to live websites, and then continue training on the public web.

Initialization and task selection. We initialize RL from a separate supervised checkpoint. We independently fine-tune Qwen3.5-9B on GPT-5.4 trajectories collected from 2,000 live-site tasks in the domains covered by WebVoyager, retaining successful trajectories through rejection-sampled distillation. The tasks are separate from WebVoyager's evaluation tasks. Unlike the studies in Sec. 5, this initialization uses only the live-site data.

Synthetic training spans EchoStay, EchoForge, EchoTunes, EchoBank, EchoCare, and EchoML. To select training tasks, we sample four attempts per candidate with the initial policy and retain tasks solved one to three times. This focuses training on tasks where the policy already has some success but remains unreliable, yielding 664 unique tasks. The 290 validation tasks are independently selected and disjoint from training; they are not subjected to this pass@4 filter. Task selection, training, and validation all use a 40-turn limit.

Outcome rewards with local action feedback. The terminal verifier establishes whether an execution achieved its goal, but gives little guidance about which intermediate actions made progress. We pair this outcome signal with a GPT-4.1 step judge that considers screenshots, actions, database changes, and task progress. It assigns 1 to useful actions, 0 to unhelpful or failed actions, and -0.5 to repeated unresolved loops.

Each GRPO update (Shao et al., 2024) samples 12 rollouts for each task from a batch of 16 tasks. Let $R_{q,j}$ be the trajectory reward for retained rollout j of prompt q, derived from its terminal verifier score with the termination, malformed-call, and turn-limit rules in Appendix F.2. For action t, we combine its rollout's relative outcome with the local step score:

$$A_{q,j,t} = R_{q,j} - \frac{1}{n_q}\sum_{k=1}^{n_q} R_{q,k} + 0.5\, s_{q,j,t}, \qquad s_{q,j,t} \in \{-0.5, 0, 1\}. \tag{1}$$

The baseline uses one final trajectory reward for each of the n_q retained rollouts, weighting long and short executions equally. Excluded trajectories do not enter the mean, so n_q can be smaller than 12. We broadcast each rollout's centered reward to all its action rows, then add the local step term with weight 0.5. Step scores are not centered, and advantages are not divided by group standard deviation (Liu et al., 2025c). Training uses 16 GPUs, a peak learning rate of 3×10^{-7}, and a fixed-reference KL penalty of 0.001. Appendix F.3 lists the optimization and rollout settings.

Learning across six environments. We evaluate four rollouts per validation task at temperature 0.7. Mean@4 averages their raw verifier scores within each task and then across tasks; pass@4 is the fraction of tasks solved at least once. These metrics distinguish average performance from success with multiple attempts. Neither includes training-time step shaping or termination penalties.

By update 80, mean@4 rises from 39.05% to 60.34%, while pass@4 rises from 64.83% to 77.59% (Figure 5). Most gains arrive in the first 20 updates; later checkpoints fluctuate around a plateau. All six environments improve in mean@4, with the largest gain in ECHOFORGE. We use the update-80 export for the subsequent transfer and live-web experiments. We train for almost 2 epochs, one epoch consists of 42 training steps for our data mix. Appendix F.4 gives the full results and evaluation denominators.

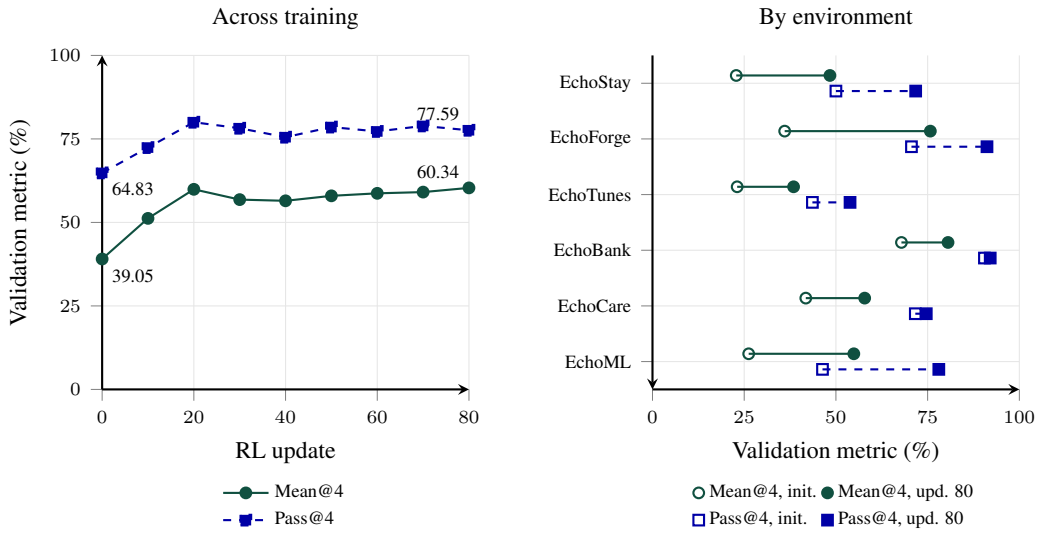


Figure 5: Synthetic RL validation on six environments, with four sampled 40-turn rollouts per task. Mean@4 averages raw verifier scores; pass@4 is the fraction of tasks solved at least once. Left: task-weighted learning curves over complete groups. Right: per-environment scores at initialization (open markers) and update 80 (filled markers), with mean@4 in green and pass@4 in blue; Table 8 gives per-environment counts.

Out of Distribution transfer beyond training environments and domains. Our synthetic RL tasks are mostly write-oriented workflows in login-gated environments, whereas Online-Mind2Web emphasizes read-only research tasks on public websites. This evaluation therefore measures out-of-distribution transfer from the synthetic RL task distribution. We evaluate transfer on 281 matched Online-Mind2Web tasks using Browserbase-hosted sessions to reduce CAPTCHA-related interruptions (Browserbase, 2025). With the same browser configuration and judge, success rises from 54.80% to 56.58% (Figure 6, left). Non-termination falls from 14.9% to 3.6%, and mean trajectory length falls from 36.2 to 31.5 actions. Appendix F.5 gives the counts and paired comparison.

Task and reference co-evolution. We also apply a restricted version of the co-evolution loop (Sec. 4.1) to task definitions and grading references. Trajectories from the preceding run guide revisions that clarify task requirements and correct references, aiming to make trajectory outcome judgments more faithful to the intended goals. Across two task/reference versions, the task-weighted mean@4 gain after 20 RL updates from the live-site-SFT initialization increases from 15.34 to 20.86 percentage points (Figure 6, right). Both runs start from the same SFT checkpoint and use the same 290-task validation set. We compute raw-judge mean@4 from four sampled attempts per task at temperature 0.7, measuring gains relative to each run's initial evaluation.

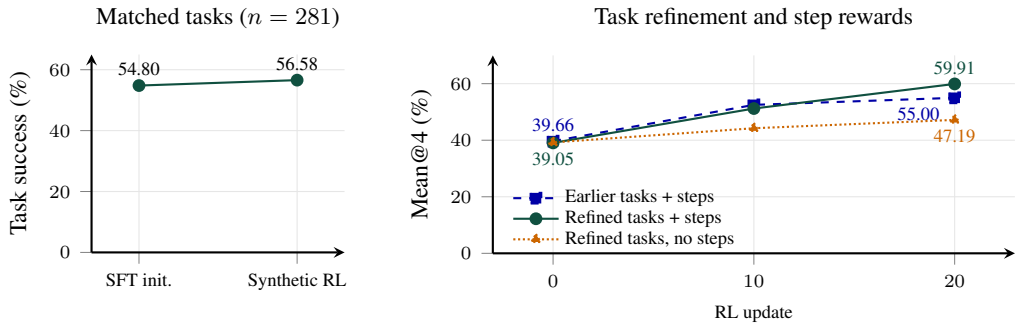


Figure 6: Transfer and RL ablations. Left: Online-Mind2Web success on 281 matched Browserbase-hosted tasks. Right: task/reference refinement and step-reward ablation at updates 0, 10, and 20. All runs share the live-site-SFT initialization. Appendix F.4 defines the raw-judge mean@4 aggregation.

Ablating step rewards. We repeat synthetic RL on refined tasks without $0.5\, s_{q,j,t}$ (Eq. 1), holding initialization, training data, trajectory rewards, and optimization settings fixed. By update 20, outcome-only mean@4 rises from 39.22% to 47.19%, compared with 39.05% to 59.91% with step rewards (Figure 6, right). The 12.73-percentage-point endpoint gap supports local action feedback beyond terminal outcome supervision in this comparison.

RL on public websites. Our trajectory-and-step-reward recipe transfers directly to public websites, but unstable interaction makes learning costly and noisy. Without database access, we use the Universal Verifier (Rosset et al., 2026), whose screenshot-based multimodal judgments depend on potentially ambiguous visual evidence. A separate multimodal LLM judge call identifies CAPTCHAs, rate limits, and login pop-ups; flagged trajectories are masked rather than treated as policy failures. In our run, only 132–174 of 192 requested training rollouts per update remain usable after environment, browser, and required-judge exclusions (Figure 7). Heavier multimodal judging contributes to mean update times $3.03\times$ those of synthetic RL, despite fewer usable trajectories (Appendix F.1). This inconsistent coverage also makes gains harder to assess: success on the same 71 tasks with unmasked scores at every checkpoint rises from 22.54% to 29.58% by update 20, while success across the changing scored sets stays near 21.5% (Appendix F.6). Algorithmic transfer thus still leaves a noisy, wasteful learning process, further motivating the importance of resettable synthetic environments with state-grounded verification.

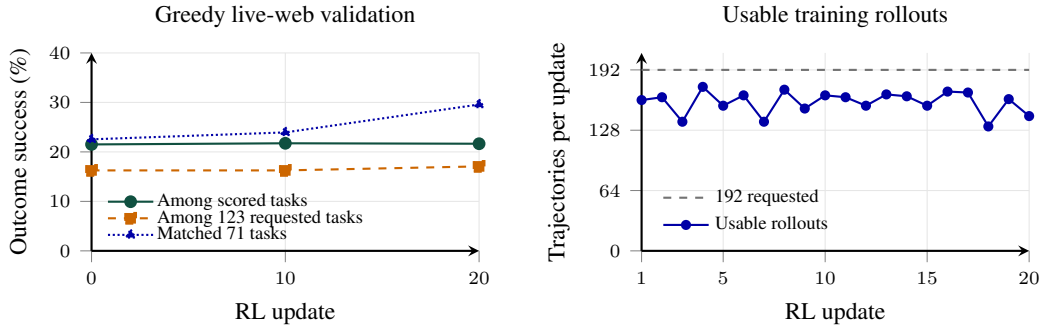


Figure 7: Preliminary live-web RL run through update 20, initialized from synthetic RL. Left: outcome-only success on the scored sets (93, 92, and 97 tasks), all 123 requested tasks, and 71 matched tasks (16, 17, and 21 successes). Right: usable rollouts out of 192 requested per update after environment, browser, and required-judge exclusions; these exclusions are not policy failures.

7 Conclusion

Echoverse combines task-driven application generation, database-grounded trajectory verification, and iterative environment repair. The SFT studies support complete workflows and varied controls as useful training-data properties. The EchoStay comparison further shows improved accuracy after training on trajectories collected from the repaired environment, with both checkpoints evaluated under the same protocol. Scaling gains depend on the evaluation domain. The environments provide resettable interaction and outcome feedback for an RL recipe combining trajectory and per-step rewards.

AI USE STATEMENT

Generative AI was used to aid and polish writing, retrieve and discover relevant literature, draft sections of the paper, and generate synthetic datasets. GitHub Copilot assisted with manuscript editing, literature and citation checks, figure and LaTeX preparation, analysis scripts, metric checks, interpretation of results, and methodological feedback. The research pipeline also uses LLM-based agents for environment construction and repair, task and grading-reference generation, demonstration collection, and trajectory evaluation (Sections 4–6). GPT-5.4 supplied demonstration trajectories; GPT-4.1 supplied synthetic outcome and local step judgments; GPT-5.6-terra supplied live-web process and outcome judgments. AI assistance does not constitute independent validation of scientific claims. The authors retain responsibility for reviewing and approving AI-assisted material and for the final text, methods, results, citations, code, and data.

ETHICS STATEMENT

Echoverse supports research on useful computer-use automation while exposing risks from unintended actions, privacy leakage, and misuse of authenticated accounts. Synthetic worlds isolate state-changing workflows from production services. Mail, calendar, banking, and health-record seed data are generated, while some other worlds use public data (Appendix B.4). Public availability does not remove privacy or licensing obligations: reuse and distribution should respect source licenses, minimize personal information, and protect credentials and private session data. Live-web use requires authorization and appropriate access controls. We adopt Fara-1.5's definition of *critical points* (Awadallah et al., 2026): situations in which the agent should pause and consult the user because the task is underspecified, required user information is missing, or an irreversible action lacks explicit authorization. For example, the agent should ask rather than invent personal information or finalize a purchase without approval. Critical-point adherence is a key part of our safety approach, intended to keep consequential decisions under user control. Detected violations remove the process-reward contribution during live-web RL (Appendix F.2). Neither resettable training environments nor reward penalties guarantee safe deployment; model-based verifiers can miss harmful side effects. The selected domains and UI families do not establish coverage of all users, languages, or accessibility needs. Responsible development and use should follow the ICLR Code of Ethics, including its principles of avoiding harm, respecting privacy, fairness, and transparent reporting.

REPRODUCIBILITY STATEMENT

Section 3 specifies the environment and grading contracts; Section 4 and Appendix B describe construction, verification, failure attribution, and resets. Appendix C documents the environments and seed entities. The SFT protocols, study conditions, and trajectory corpus are described in Appendices D and E. Section 6 and Appendix F give the RL rewards, reported optimization and rollout settings, task counts, metric definitions, and evaluation denominators. Appendices D.5 and F.4 explain the provenance of digitized scaling values and reconstructed pass@4 statistics, and Appendix F.4 defines the fixed-weight comparison metric. Appendix G reproduces the outcome-verifier prompts, and Appendix H summarizes the capability-world controls. We will release datasets containing seed data and test tasks for ECHOSTAY, ECHOFORGE, and the datepicker and nested-filter worlds, alongside their environments and verifiers.

REFERENCES

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-Anything: Turn any software into an agent environment. *arXiv preprint arXiv:2604.06126*, 2026.

Anthropic. Model context protocol. `https://modelcontextprotocol.io`, 2024.

Ahmed Awadallah, Sahil Gupta, Yash Lara, Yadong Lu, Hussein Mozannar, Akshay Nambi, Zach Nussbaum, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Luiz do Valle, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-1.5: Scalable learning environments for computer use agents. *arXiv preprint arXiv:2606.20785*, 2026.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. *arXiv preprint arXiv:2406.11896*, 2024.

Browserbase. Browserbase: Headless browser infrastructure for AI agents. `https://www.browserbase.com`, 2025.

Murray Cox. Inside Airbnb: Adding data to the debate. `http://insideairbnb.com`, 2024.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. The BrowserGym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *International Conference on Machine Learning (ICML)*, 2024.

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.

Yifei He, Rui Yang, Hao Bai, Tong Zhang, and Han Zhao. PRO-CUA: Process-reward optimization for computer use agents. *arXiv preprint arXiv:2605.29119*, 2026.

Mianqiu Huang, Taofeng Xue, Chong Peng, et al. EvoCUA-1.5: Online reinforcement learning for multi-turn computer-use agents. *arXiv preprint arXiv:2607.09773*, 2026.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. ComputerRL: Scaling end-to-end online reinforcement learning for computer use agents. *arXiv preprint arXiv:2508.14040*, 2025.

Pengxiang Li, Zechen Hu, Zirui Shang, et al. Efficient multi-turn RL for GUI agents via decoupled training and adaptive data curation. *arXiv preprint arXiv:2509.23866*, 2025.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. InfiGUI-R1: Advancing multimodal GUI agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*, 2025a.

Yuhang Liu, Zeyu Liu, Shuanghe Zhu, et al. InfiGUI-G1: Advancing GUI grounding with adaptive exploration policy optimization. *arXiv preprint arXiv:2508.05731*, 2025b.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025c.

Dunjie Lu, Shuai Bai, Tianyi Bai, et al. Qwen-CUA: Native computer use for (almost) everything. *arXiv preprint arXiv:2608.02352*, 2026.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, et al. UI-R1: Enhancing efficient action prediction of GUI agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 2025.

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. *arXiv preprint arXiv:2504.10458*, 2025.

Bowen Lv, Xiao Liu, Yanyu Ren, Hanyu Lai, Bohao Jing, Hanchen Zhang, Yanxiao Zhao, Shuntian Yao, Jie Tang, and Yuxiao Dong. SCALECUA: Scaling computer use agents with verifiable task synthesis and efficient online RL. *arXiv preprint arXiv:2607.11185*, 2026.

Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps. `https://playwright.dev`, 2024.

Zehan Qi, Xiao Liu, Iat Long Iong, et al. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, 2025. URL `https://arxiv.org/abs/2411.02337`.

Qwen Team. Qwen3.5: Towards native multimodal agents. `https://qwen.ai/blog?id=qwen3.5`, February 2026.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Hakimi, Yuan Xu, Oriana Riva, and Timothy Lillicrap. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. The art of building verifiers for computer use agents. *arXiv preprint arXiv:2604.06240*, 2026.

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. *arXiv preprint arXiv:2402.03300*, 2024.

Chenrui Shi, Yuwei Wu, Yang Liu, Ruining Feng, Zirui Shang, Zhi Gao, Lifeng Fan, and Che Sun. Interactive reward agent: GUI task evaluation via environment-state verification. *arXiv preprint arXiv:2607.25904*, 2026.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, 2017. URL `https://proceedings.mlr.press/v70/shi17a.html`.

Yucheng Shi, Wenhao Yu, Zaitang Li, Yonglin Wang, Hongming Zhang, Ninghao Liu, Haitao Mi, and Dong Yu. MobileGUI-RL: Advancing mobile GUI agent through reinforcement learning in online environment. *arXiv preprint arXiv:2507.05720*, 2025.

Marta Sumyk and Oleksandr Kosovan. Reinforcement learning for computer-use agents with autonomous evaluation. *arXiv preprint arXiv:2606.24515*, 2026.

Qiushi Sun, Kanzhi Cheng, Yian Wang, et al. OSReward: Instituting standardized evaluation for cross-platform computer-use reward models. *arXiv preprint arXiv:2607.28609*, 2026.

Yang Wan, Zhenhao Zhang, Jierui Wang, and Linchao Zhu. SeekJudge: A practical reward framework for reinforcement learning in computer-use agents. *arXiv preprint arXiv:2607.23263*, 2026.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. CUA-Gym: Scaling verifiable training environments and tasks for computer-use agents. *arXiv preprint arXiv:2605.25624*, 2026.

Haoming Wang, Haoyang Zou, Huatong Song, et al. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025a.

Kangrui Wang, Pingyue Zhang, Zihan Wang, et al. VAGEN: Reinforcing world model reasoning for multi-turn VLM agents. *arXiv preprint arXiv:2510.16907*, 2025b.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. WebAgent-R1: Training web agents via end-to-end multi-turn reinforcement learning. *arXiv preprint arXiv:2505.16421*, 2025.

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 (NeurIPS)*, 2024.

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

Chenyu Yang, Shiqian Su, Shi Liu, et al. ZeroGUI: Automating online GUI learning at zero human cost. *arXiv preprint arXiv:2505.23762*, 2025.

Rui Yang, Qianhui Wu, Zhaoyang Wang, Hanyang Chen, Ke Yang, Hao Cheng, Huaxiu Yao, Baolin Peng, Huan Zhang, Jianfeng Gao, and Tong Zhang. GUI-Libra: Training native GUI agents to reason and act with action-aware supervision and partially verifiable RL. *arXiv preprint arXiv:2602.22190*, 2026a.

Yuhao Yang, Zhen Yang, Zi-Yi Dou, Anh Nguyen, Keen You, Omar Attia, Andrew Szot, Michael Feng, Ram Ramrakhya, Alexander Toshev, Chao Huang, Yinfei Yang, and Zhe Gan. UltraCUA: A foundation model for computer use agents with hybrid action. *arXiv preprint arXiv:2510.17790*, 2026b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Tong Ye, Kunyang Han, Guozhi Wang, et al. BlueLM-GUI technical report: A real-device-centric flywheel for self-improving mobile GUI agents. *arXiv preprint arXiv:2609.12394*, 2026.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Zhong Zhang, Yaxi Lu, Yikun Fu, et al. AgentCPM-GUI: Building mobile-use agents with reinforcement fine-tuning. *arXiv preprint arXiv:2506.01391*, 2025.

Ziyun Zhang, Zezhou Wang, Xiaoyi Zhang, Zongyu Guo, Jiahao Li, Bin Li, and Yan Lu. InfiniteWeb: Scalable web environment synthesis for GUI agent training. *arXiv preprint arXiv:2601.04126*, 2026. Accepted to ACL 2026.

Shuyan Zhou. WebArena-Infinity: Generating browser environments with verifiable tasks at scale. `https://webarena.dev/webarena-infinity/`, March 2026.

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 *International Conference on Learning Representations (ICLR)*, 2024.

A Related RL training settings

Table 2 summarizes the training settings most relevant to Sec. 2. Grounding and offline action prediction measure different capabilities from complete task execution. Online systems also differ in application access, reward availability, and permitted actions. We therefore compare their training designs without ranking scores from different benchmarks.

Table 2: Selected RL approaches for GUI and computer-use agents.

Work	Relevant training design
DigiRL (Bai et al., 2024)	Offline-to-online Android learning with a VLM evaluator and automatic curriculum.
WebRL; WebAgent-R1 (Qi et al., 2025; Wei et al., 2025)	Multi-turn web RL using an evolving curriculum and learned outcome model (WebRL), or binary completion rewards (WebAgent-R1).
ZeroGUI (Yang et al., 2025)	Online GUI learning with VLM-generated tasks and estimated task-completion rewards.
MobileGUI-RL (Shi et al., 2025)	Online mobile training with trajectory-aware advantages and success/efficiency rewards.
ComputerRL; DART; UI-TARS-2 (Lai et al., 2025; Li et al., 2025; Wang et al., 2025a)	Scalable multi-turn training systems, including distributed environments, asynchronous rollouts, and training-stability mechanisms.
UltraCUA (Yang et al., 2026b)	SFT and online RL on verifiable synthetic tasks with hybrid GUI and high-level tool actions.
SCALECUA; EvoCUA-1.5 (Lv et al., 2026; Huang et al., 2026)	Verifiable task synthesis, adaptive task selection, and efficient online policy optimization.
Qwen-CUA; BlueLM-GUI (Lu et al., 2026; Ye et al., 2026)	Large-scale verifiable computer-use training and real-device mobile RL, respectively.
PRO-CUA; VAGEN (He et al., 2026; Wang et al., 2025b)	Step-level process feedback for computer use and turn-level world-modeling rewards for visual agents.
UI-R1; GUI-R1; InfiGUI-G1 (Lu et al., 2025; Luo et al., 2025; Liu et al., 2025b)	RL for action prediction and visual grounding with verifiable action-level rewards.
AgentCPM-GUI; InfiGUI-R1 (Zhang et al., 2025; Liu et al., 2025a)	Reinforcement fine-tuning after grounding or reasoning supervision; InfiGUI-R1 also constructs recovery scenarios.
GUI-Libra (Yang et al., 2026a)	Offline step-wise RL accounting for partially verifiable actions and the mismatch between step accuracy and online success.

B Factory and co-evolution details

Figure 2 shows the factory end to end. Phase 1 builds and validates the environment; Phase 2 grows a grounded task corpus on top of it.

B.1 Agent roles and tools

Frontier LLM agents use the GitHub Copilot SDK with repository access, a terminal, Playwright (Microsoft, 2024), and database tools exposed through MCP (Anthropic, 2024). Builders implement schemas, seed data, routes, interfaces, and tasks. Verifier agents inspect these outputs through code, database queries, and browser interaction. Triage groups findings by cause and affected layer; fixer agents apply repairs and check regressions. Human QA provides additional review.

B.2 Phase 1: building the environment

Hand-written seeds expand into scenarios and task proposals that inform the environment specification. Claims are derived from the specification and seed task scenarios before builders generate a FastAPI and SQLite backend with a React frontend. Each claim is a structured assertion with a type, a subject, and a type-specific predicate describing an expected property or behavior; its source section and line range link it back to the specification. Claim types cover data-model fields and

constraints, seed data, endpoint contracts, UI components, interactions and navigation, and feature-level capabilities. For example, a UI-interaction claim can identify the Create Playlist button as its subject and require that clicking it opens the creation dialog. The claim type guides the evidence gathered: code and database inspection for schema properties, HTTP requests for endpoint behavior, and browser interaction for UI behavior. These assertions specify requirements, rather than constituting executable unit tests themselves. Verifier agents inspect source code, query the database, and use the hosted interface to assess the claims. Failed claims guide repairs to the database, backend, or frontend and are then rechecked. The configured acceptance threshold is 95% of claims judged passing, with no unresolved hard blocker. This threshold records agent judgments under the performed checks, not a guarantee of correctness beyond their coverage.

B.3 Phase 2: growing the corpus

Phase 2 grounds task proposals in the live database and binds each task to a user profile. Task goals and grading references use database entities available to that profile. Verifier agents assess validity, clarity, plausibility, difficulty, solvability, and the grading reference using source code, database evidence, and browser trajectories. Their white-box access helps distinguish an unavailable workflow from a policy failure; teacher success during later trajectory collection does not determine task feasibility.

Findings are consolidated by affected layer before task, database, backend, frontend, or verifier repairs. After verified fixes, tasks are reassessed before pass rates are computed. The configured acceptance threshold is 95% of surviving tasks judged passing. Environment and task versions are updated together, and each exported task carries its grading reference. In either phase, runs stopped by budget, iteration, or plateau limits are marked incomplete.

B.4 Seed data and reset

EchoStay uses public listings, hosts, reviews, and amenities from InsideAirbnb (Cox, 2024). EchoForum uses a public corpus with 2.55 million comments. Mail, calendar, banking, and health-record environments use generated seed data subject to schema and consistency checks. Table 4 summarizes the seeded entities.

Each rollout uses an isolated per-task database copy. Changes made in one episode therefore do not affect the initial database of another. Seed-data updates trigger regeneration and validation of the task references. The same copy-and-reset procedure supports repeated RL rollouts.

B.5 Failure attribution and regression checks

Triage checks whether the goal's entities exist, whether the required workflow is reachable, and whether the verifier checks the intended outcome. Defects are grouped by cause and prioritized by affected tasks. A repair must preserve the goal; weakening the task to obtain a passing result is treated as a regression. Failures on tasks judged valid, solvable, and correctly graded are retained as policy failures. Ambiguous cases are withheld until further inspection. The EchoStay guest-count and listing-filter defects in Appendix D.6 illustrate findings from agent rollouts.

C Environment suite details

C.1 Full-domain environments

Table 3 groups the full-domain environments by workflow category. Their task sets exercise shared records, permissions, persistent changes, and multi-step dependencies. Figure 8 shows the interfaces, including held-out examples from the two capability worlds. Environment names indicate the workflow category; benchmark and public-data names retain their published names.

A booking in EchoStay involves search, listing selection, availability, and payment in an environment with roughly 87 routes and 23 tables. EchoMail tracks drafts, delivery, replies, and labels across users; EchoCare records order changes in an audit history. These dependencies provide the state required for the corresponding read and write tasks.

Table 3: Full-domain environments and their workflow requirements.

Workflow category	Worlds	Workflow requirements
Communication & coordination	EchoMail, EchoCalendar, EchoChat	Shared threads, schedules, participants, permissions, histories
Technical creation & operations	EchoML, EchoForge	Artefacts, configuration, dependencies, roles, multi-stage changes
Regulated records & transactions	EchoBank, EchoCare	Balances or records, authorisation, audit history, consequential writes
Community, media & travel	EchoForum, EchoTunes, EchoStay	Persistent preferences, social state, search, booking, account actions

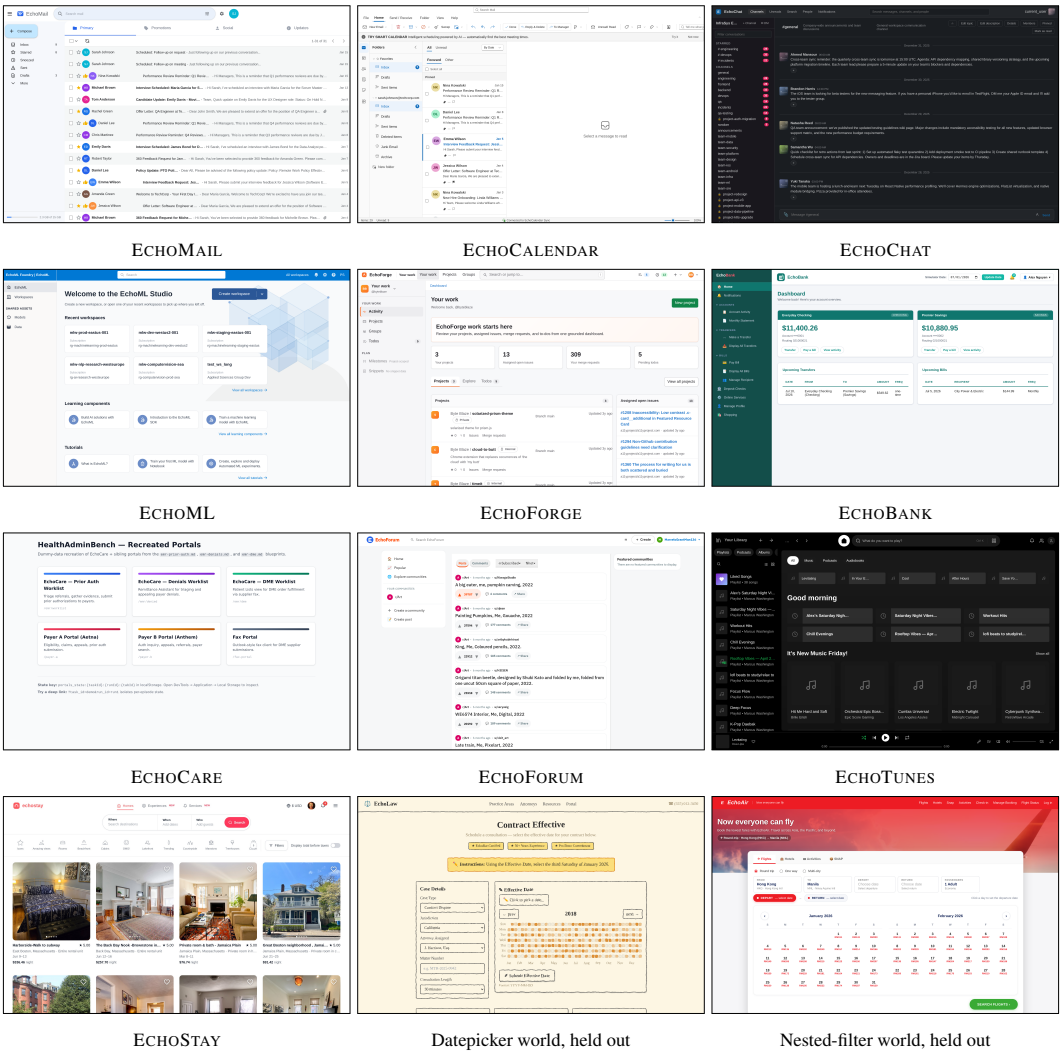


Figure 8: Interfaces in the environment suite. The first ten panels show full-domain environments. The final two show held-out capability frontends: a calendar heatmap and a compound booking-search panel.

Table 4: Schema sizes and representative seed entities in the full-domain environments.

World	Domain	Tables	Representative seeded entities
ECHOMAIL	Email workspace	13	1.1K emails, 1.1K threads, 170 labels, 374 contacts
ECHOCALENDAR	Calendar and mail	13	196 events, 102 folders, 374 contacts
ECHOCHAT	Team messaging	15	188 channels, 1.3K workspaces, 3.2K users
ECHOML	ML platform	11	173 models, 444 datasets, 152 endpoints, 309 compute resources
ECHOFORGE	Developer collaboration	15	175 projects, 81K issues, 134K merge requests
ECHOBANK	Online banking	10	100 accounts, 4.1K transactions, 201 transfers, 672 bills
ECHOCARE	Health revenue cycle	14	250 prior authorisations, 250 denials, 200 orders, 1.3K documents
ECHOFORUM	Social forums	11	95 forums, 127K posts, 2.55M comments, 662K users
ECHOTUNES	Music streaming	42	2.5K songs, 1.1K albums, 863 artists, 544 playlists
ECHOSTAY	Lodging marketplace	23	3.7K listings, 5.4K bookings, 5.3K reviews, 6.1K users

C.2 TWO CAPABILITY WORLDS

Capability environments vary a selected control across layouts, states, and constraints. The datepicker training set includes six core widgets across ten contexts, with 100 frontends in six styles. The held-out set contains ten widget families across 36 scenarios, with 80 frontends in nine styles. Tasks include relative-date computations such as the last Thursday of January 2026 or ten business days after a specified date.

The nested-filter world includes twenty training families across 200 frontends in fifteen styles, and nine held-out compound families across 100 frontends in nine styles. Submissions are checked against the requested filter conditions using the environment's filtering logic. Thematic variation covers six domain verticals for filters and ten contexts for datepickers. Appendix H summarizes the represented interaction types.

D ENVIRONMENT-DESIGN STUDY DETAILS

D.1 TRAJECTORY COLLECTION AND TRAINING

GPT-5.4 attempts the generated tasks, and the grounded verifier in Sec. 3.1 retains passing trajectories for SFT. Selection is over trajectories: a teacher failure contributes no demonstration but does not invalidate a task whose feasibility was established by the factory. This is verifier-filtered rejection-sampled distillation (Zelikman et al., 2022; Yuan et al., 2023).

Each SFT study independently fine-tunes Qwen3.5-9B on its designated verified GPT-5.4 demonstrations plus the ECHOMAIL alignment data used to train π_{base}, producing a separate checkpoint. The shared alignment data contain only a few hundred synthetic ECHOMAIL trajectories. The full twelve-world corpus in Appendix E supplies the trajectory-volume sweep; it is distinct from the two-domain depth and capability-study corpora. Its per-world counts therefore do not specify the training-set sizes of the depth or capability conditions.

D.2 EVALUATION SETUP

For the SFT studies in Sec. 5, policies use Fara-1.5's action space and CUA harness (Awadallah et al., 2026) and are served with vLLM (Kwon et al., 2023). Evaluation rollouts are capped at 100 actions. A rollout reaching the cap without `terminate` is recorded as over-budget. Synthetic evaluations use the grounded verifier or the capability environment's state checks. Live WebVoyager and Online-Mind2Web evaluations use their published judges. Browserbase (Browserbase, 2025) is

used where a hosted-browser condition is explicitly reported; the scaling sweeps run without this service. The Browserbase-hosted RL transfer comparison is reported separately in Appendix F.5.

D.3 Environment depth

The comparison in Sec. 5 uses trajectories from shallow or deep synthetic versions of Allrecipes and Hugging Face, alongside the common EchoMail alignment data. The shallow environments have lower frontend fidelity and more limited feature support. The study-specific corpora cover the same two domains and are comparably sized, without trajectory-level matching. Evaluation uses the corresponding live WebVoyager sites; Figure 3 reports accuracy. Among 37 Hugging Face tasks, the number exhausting the action budget falls from 15 to 9 after SFT on trajectories from the deep environments.

D.4 Targeting Capability Gaps

Table 1 compares the baseline with checkpoints whose added training trajectories come from datepickers, nested filters, or both. All conditions include the common alignment data and use the independent fine-tuning procedure in Appendix D.1. Held-out widget families are excluded from the capability training frontends. The separate conditions show cross-world transfer: adding datepicker trajectories improves held-out nested-filter accuracy to 82.1, while adding nested-filter trajectories improves held-out datepicker accuracy to 50.7. Joint training has the highest reported accuracy on all four widget splits.

D.5 Data volume and environment coverage

The trajectory sweep subsamples the 21,009-trajectory corpus at fixed per-world proportions. The environment sweep varies the included worlds. Figure 4 reports synthetic accuracy and live-web evaluations without a hosted browser. The synthetic average is an unweighted mean over fourteen splits: ten domain evaluations and two splits for each capability world.

Table 5 lists all values plotted in Figure 4. Intermediate accuracies were digitized from the original plot rather than recovered from evaluation logs, and the 400-trajectory position was inferred from the linear horizontal axis. These values should therefore be treated as approximate plot coordinates, not additional measurements. The environment-count axis does not specify per-condition trajectory budgets or domain ordering, so that curve alone does not establish a matched-budget effect of environment diversity.

Table 5: Values plotted in Figure 4 (accuracy, %). Intermediate values are digitized approximations; the 400-trajectory position is inferred from the original plot.

Training condition	WebVoyager	Online-Mind2Web	Avg. synthetic
Synthetic trajectories added			
100	35.5	18.8	36.4
400	43.5	27.1	48.2
1,600	50.2	31.2	58.3
6,400	54.8	40.1	62.3
20,000	55.6	37.2	67.1
Synthetic environments added			
2	49.7	29.5	39.7
4	46.0	25.2	37.2
6	54.9	34.4	49.5
8	54.9	43.1	55.6
10	53.7	38.1	58.8
12	55.6	37.2	67.1

From 6,400 to 20,000 trajectories, WebVoyager changes from 54.8 to 55.6, while Online-Mind2Web declines from 40.1 to 37.2. With additional worlds, synthetic accuracy continues to improve across the larger mixtures, while WebVoyager largely plateaus after six worlds and Online-Mind2Web

peaks at eight before declining. These are the saturation patterns discussed in Sec. 5. The write-oriented synthetic tasks differ from the live benchmarks' task distribution, so the live-web gains largely measure out-of-domain transfer.

D.6 Environment Co-Evolution

Rollouts in ECHOSTAY exposed a guest-count control defect and nonfunctional listing-filter widgets. Repairing the guest-count control recovers 15 of 24 blocked bookings and raises task feasibility from 48% to 78%. The model comparison in Sec. 5 uses checkpoints trained on trajectories collected from the original and repaired environments, each with the common alignment data. Both are evaluated on the same improved test set from the repaired *world*, using the same environment and verifier. Figure 3 reports model accuracy (16.2% and 38.5%); task feasibility is a separate measurement.

Other repair outcomes measure corpus usability. Frontend and page-load fixes change one ECHOFORUM set from 0 to 36 of 37 tasks solved. Realigning the ECHOCHAT verifier with its database increases the share of gradable tasks from 34% to 99%. ECHOCARE required a state-wiring fix, and ECHOFORGE required a UI control for existing backend functionality. These outcomes concern world repairs, not changes in trained-policy accuracy.

E Trajectory corpus for scaling

Table 6 reports the 21,009 verified GPT-5.4 trajectories used for the trajectory-volume sweep. Subsampling preserves the reported per-world proportions. The depth and capability studies use their designated additions; all studies follow the common-alignment and independent fine-tuning procedure in Appendix D.1. Mean trajectory lengths range from 4.70 actions for nested filters to 33.80 for ECHOSTAY.

Table 6: Verified GPT-5.4 trajectories in the scaling corpus, per world. *Task sets* counts the separately exported task datasets contributing trajectories from that world. Steps are agent actions.

World	Task sets	Trajectories	Steps	Avg. steps/traj.
ECHOMAIL	5	4,482	91,155	20.34
ECHOCALENDAR	3	4,109	91,891	22.36
ECHOCHAT	1	832	14,954	17.97
ECHOML	4	2,450	70,397	28.73
ECHOFORGE	1	824	20,596	25.00
ECHOBANK	2	852	7,801	9.16
ECHOCARE	1	703	13,680	19.46
ECHOFORUM	3	1,472	25,149	17.08
ECHOTUNES	2	919	15,759	17.15
ECHOSTAY	3	757	25,586	33.80
Datepicker world	3	1,827	19,475	10.66
Nested-filter world	3	1,782	8,371	4.70
Total	**31**	**21,009**	**404,814**	**19.27**

F Reinforcement-learning details

F.1 Synthetic and live-web interaction

Synthetic rollouts use isolated database copies and database-derived task references, supporting repeatable initial state and outcome checks against stored results. Live websites do not expose the same reset and state access, so the Universal Verifier (Rosset et al., 2026) reconstructs outcomes from action history and visual evidence. It scores screenshot relevance to task criteria, analyzes selected evidence, and performs process rescoring, side-effect checks, and outcome verification. These semantic judgments remain model-based. The judging-cost difference involves the model and input modality as well as the number of requests.

Verification cost and latency. Echoverse uses a compact text-only GPT-4.1 outcome check and one batched GPT-4.1 step judgment, which can use screenshots. Live verification instead requires repeated multimodal calls to more expensive GPT-5.6-terra: 47.02 admitted attempts per requested training or validation trajectory, including retries and failures, versus about two nominal synthetic requests (Figure 9). This roughly $24\times$ contrast compares measured live attempts with a synthetic estimate and excludes policy generation. Rate-limit-aware pipelining manages the limited GPT judging throughput. Mean training-update time over updates 1–20 is 65.82 versus 21.70 minutes ($3.03\times$), excluding periodic validation (Figure 10). The slowdown persists despite fewer usable live rollouts. These run-level differences also reflect tasks, rollout budgets, and trajectory lengths; the following analyses give the request mix and timing breakdown.

HTTP judging workload. The normal synthetic training path makes one terminal-verifier request and one batched step-judge request, both using GPT-4.1. The latter evaluates the complete action sequence using action summaries, observations, selected screenshots, database deltas, and loop evidence, returning aligned per-action scores in one response. Per-action feedback therefore does not require one HTTP call per action. The nominal count is two requests per trajectory. Accounting for the 0.70% malformed/aborted fraction over training updates 1–20, whose calls are skipped, gives approximately 1.99 requests per requested trajectory. This is an estimate from the verification procedure: the synthetic run did not preserve complete HTTP-attempt counters, and retries can raise the actual count.

For live-web continuation, the shared limiter records 197,923 admitted HTTP judge attempts for 4,209 dispatched trajectories through update 20, or 47.02 attempts per requested trajectory. This denominator includes training and validation trajectories, including those later masked or dropped. The numerator includes retries, timeouts, HTTP failures, and successful requests, but excludes policy-generation calls. Of these attempts, 192,072 use GPT-5.6-terra for multimodal evidence and decisions (45.63 per requested trajectory, about 97% of traffic); 5,851 use GPT-4.1 for batched step judgment and conditional environment classification (1.39 per requested trajectory). Failed attempts are not equivalent to lost trajectories. Figure 9 distinguishes this measured workload from the nominal synthetic estimate.

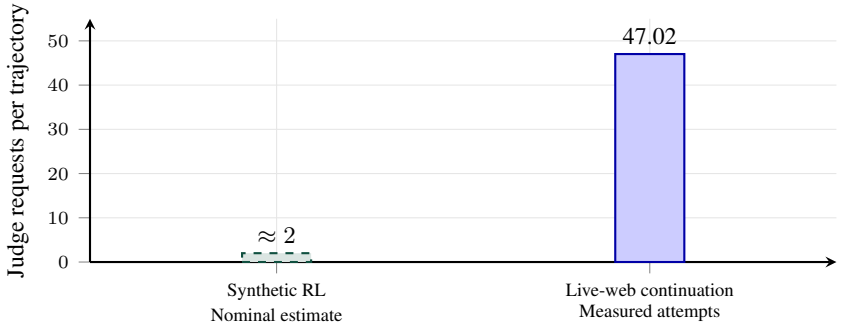


Figure 9: External HTTP judging demand, excluding policy generation. Synthetic RL uses about two requests on its normal training path, estimated from the verifier structure without a measured retry ledger. Live-web continuation averages 47.02 admitted attempts over 4,209 requested training and validation trajectories through update 20, including retries and trajectories later excluded. The roughly $24\times$ contrast is between measured live attempts and a nominal synthetic estimate, not token or monetary cost.

The live run uses local headless Chromium at 1440×900, retains three solver screenshots in context, and allows 1,800 seconds per trajectory. Each update requests 192 trajectories through 64 rolling completion workers. Workers are refilled as trajectories finish, but reused only after browser cleanup and postprocessing are complete. Unconfirmed cleanup retires the worker while the remaining workers continue. Limited GPT API throughput is a constraint on the pipeline: rollout and judging work is pipelined with controlled request dispatch designed to avoid exceeding API rate limits. A limiter shared across nodes and judging models caps concurrent requests at 96, with a 600-second queue timeout, 300-second lease, and 120-second timeout per judge call. This cap bounds in-flight work; it does not denote a request rate or token-throughput allowance. Pipelining allows browser and veri-

fication work to overlap, while controlled dispatch limits pressure on the judging endpoints. It does not eliminate their throughput constraint.

Unavailable site state, browser errors, whole-trajectory timeouts, and required verifier or environment-classifier failures exclude trajectories from training. An optional step-judge failure instead retains the trajectory reward and omits local shaping. Usable live training batches range from 132 to 174 trajectories out of 192 requested through update 20. These exclusions are not policy failures. The synthetic and live stages differ in tasks, rewards, rollout budgets, and validation sampling, and therefore do not constitute a controlled comparison of absolute success rates or wall-clock cost.

Latency under the available API throughput. Figure 10 compares complete training-update times over updates 1–20. Both runs use 16 GPUs and request 192 trajectories per update. The reported means are 21.70 minutes for synthetic RL and 65.82 minutes for live-web RL (3.03×); the medians are 22.13 and 65.60 minutes. These timers exclude separately reported periodic validation.

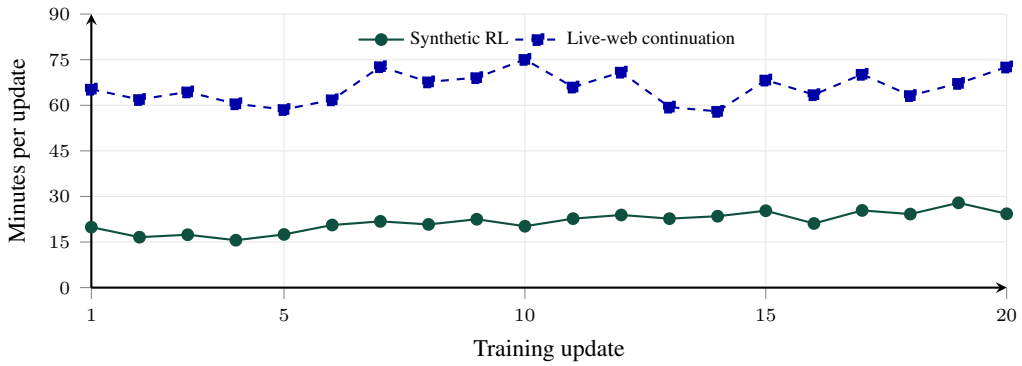


Figure 10: Complete training-update latency over updates 1–20, excluding periodic validation. Mean times are 21.70 minutes for synthetic RL and 65.82 minutes for live-web continuation (3.03×). Per-update values are rounded to one decimal place. The runs use different tasks, rollout budgets, and verification workloads under limited judging API throughput.

Table 7 reports the component timers. Rollout/browser generation is 5.13× slower in the live run and occupies about 62% of its complete-update time. Actor updates also take longer (1.90×). The live run permits 50 turns rather than 40 and produces longer trajectories, so its browser, judging, and optimization workloads differ from synthetic RL. The rollout timer does not isolate API waiting from browser interaction or model inference. Checkpoint saves take about 19 seconds in either run and do not explain the update-time gap. The expanded judging workload contributes to the demand on rate-limited APIs, but request count and wall-clock time need not scale proportionally: calls execute concurrently, their input sizes differ, and policy generation and optimization also contribute to each update. Neither the request counts nor these timings establish a monetary-cost ratio.

Table 7: Mean trainer timing over updates 1–20. Complete-update time is measured separately from the component timers and excludes periodic validation. Times and live/synthetic ratios are rounded independently.

Timer	Unit	Synthetic	Live web	Ratio
Complete training update	min	21.70	65.82	3.03×
Rollout/browser generation	min	8.02	41.11	5.13×
Actor update	min	9.20	17.46	1.90×
Old-policy log probabilities	min	2.21	3.68	1.67×
Reference-model pass	min	2.10	3.34	1.59×
Advantage construction	s	2.0	2.8	1.40×
Weight synchronization	s	4.3	7.1	1.64×
Checkpoint save	s	17.5	19.1	1.09×

Periodic validation overhead. At updates 10 and 20, validation takes 26.0 and 25.5 minutes in the synthetic run, compared with 44.1 and 33.9 minutes in the live run. The protocols differ: synthetic

validation requests four attempts for each of 290 tasks, while live validation requests one for each of 123 tasks. Amortizing these two validation events over updates 1–20 gives approximately 24.3 minutes per update for synthetic RL and 69.7 minutes for live-web RL, a $2.87\times$ wall-clock ratio. This calculation excludes initial validation and does not equate the two validation workloads.

F.2 METHOD

Both stages use VERL with GRPO (Shao et al., 2024). The actor is the only learned network in the policy update; there is no separately trained value model. A low-variance KL loss with coefficient 0.001 penalizes divergence from a fixed reference policy.

Synthetic trajectory reward. Let J denote the raw terminal verifier score. For a retained synthetic training rollout, the trajectory reward is

$$R^{\text{syn}} = \begin{cases} -1, & \text{malformed or aborted tool call,} \\ 0.5, & \text{turn limit with an exact pure-write state change,} \\ 0, & \text{turn limit otherwise,} \\ J - 0.1\,\mathbf{1}[J < 0.5], & \text{normal explicit termination.} \end{cases} \tag{2}$$

The incorrect-termination deduction applies only to the last branch, not to an aborted rollout or one reaching the turn limit. The synthetic outcome check uses the database-grounded verifier in Sec. 3.1. The GPT-4.1 step judge additionally considers screenshots, actions, database or state changes, task progress, and repeated unresolved loops.

Credit assignment. For each prompt, we use only the final row of each retained rollout to compute the mean trajectory reward. The nominal group size is 12, but excluded rollouts have no rows and do not enter the realized group. Subtracting this mean yields one trajectory advantage per rollout, broadcast to all its action rows. The local term $0.5s_{q,j,t}$ is then added as in Eq. 1. Long trajectories do not receive extra weight in the group baseline. The local contributions are 0.5 for useful progress, 0 for an unhelpful or failed action, and -0.25 for a repeated unresolved loop. They are not centered, and the final advantages are not divided by group standard deviation (Liu et al., 2025c).

Live-web trajectory reward. The Universal Verifier (Rosset et al., 2026), using GPT-5.6-terra for evidence and decision stages, returns a normalized process score $P_j \in [0, 1]$ and a binary outcome $O_j \in \{0, 1\}$. Its process score is a trajectory-level rubric assessment, distinct from the local step score. If C_j indicates a critical-point violation, such as crossing a transactional boundary without permission, the training reward is

$$R_j^{\text{live}} = 0.5(1 - C_j)P_j + \begin{cases} -0.2, & \text{incorrect explicit normal termination,} \\ 0.5O_j, & \text{otherwise.} \end{cases} \tag{3}$$

The critical-point gate affects process credit only, not a valid outcome reward. The -0.2 penalty is absolute, applied after weighting. Turn limits and other non-terminated failures do not receive this penalty. We combine the centered trajectory reward with the same local shaping term. GPT-4.1 supplies step and environment-classification judgments; process and outcome feedback serve as fallible context for the step judge. Missing optional step judgments contribute zero local shaping while retaining the trajectory reward.

F.3 SETUP

The RL initialization is an independent fine-tune of Qwen3.5-9B on GPT-5.4 trajectories collected from 2,000 live-site tasks in WebVoyager domains. These tasks are separate from WebVoyager evaluation tasks. This checkpoint uses only the live-site data, without the common ECHOMAIL alignment data used in Sec. 5.

For synthetic RL, we attempt each candidate training task four times with the initial policy and retain tasks solved one to three times, using a 40-turn limit. This selects tasks of intermediate difficulty without guaranteeing mixed outcomes in later rollout groups. Table 8 reports the resulting training pool. The 290 validation tasks are selected independently and are not subjected to the pass@4 filter.

Training and validation share neither task IDs nor normalized goals. The 672 training rows contain 664 unique tasks, and each contiguous 16-row batch contains distinct task IDs. Synthetic training and validation both use 40 turns. Each epoch is 42 updates; the run logged 82 of 84 planned updates, with update 80 used as the last evaluated, durable export.

Table 8: Synthetic RL task pool. Training contains 672 rows and 664 unique tasks. The 290 validation tasks are selected independently without the training pass@4 filter and are disjoint from training by ID and normalized goal. Per-environment training counts are rows, not unique tasks.

World	Training rows	Validation tasks
EchoStay	111	46
EchoForge	192	34
EchoTunes	62	39
EchoBank	86	63
EchoCare	131	67
EchoML	90	41
Total	**672**	**290**

Table 9 gives the shared optimization settings and the stage-specific rollout schedules. The live stage initializes from the synthetic update-80 export and uses 400 curated training tasks and 123 validation tasks. Its task-specific rubrics cover all 523 selected tasks. Live results are an interim snapshot through update 20, not the completed 50-update schedule.

Table 9: RL configurations. Planned schedules are distinguished from the observed updates. Both stages use the advantage in Eq. 1.

Setting	Synthetic RL	Live-web continuation
GPUs	16 on two nodes	16 on two nodes
Prompts / rollouts per prompt	16 / 12	16 / 12
PPO mini-batch	16	16
Micro-batch per GPU	2	4
Peak learning rate	3×10^{-7}	3×10^{-7}
KL loss / coefficient	Low-variance / 0.001	Low-variance / 0.001
Entropy coefficient	0.001	0.001
Advantage std normalization	Disabled	Disabled
Rollout correction	Token importance weights clipped at 2	Token importance weights clipped at 2
Training temperature	0.7	0.7
Repetition penalty	1.0	1.0
Training / validation turns	40 / 40	50 / 50
Validation sampling	4 rollouts, temperature 0.7	1 rollout, greedy
Planned updates / epochs	84 / 2	50 / 2
Warmup / stable / decay updates	9 / 33 / 42	6 / 19 / 25
Final learning-rate ratio	0	0
Checkpoint / validation interval	5 / 10 updates	10 / 10 updates
Observed training updates	82	20 at snapshot

F.4 Results

The validation in Figure 5 samples four 40-turn rollouts per task at temperature 0.7, using raw verifier scores J only. The trajectory-reward adjustments in Eq. 2 and the local step term are not included. For the complete task groups $\mathcal{Q}_u$ at update u, the metrics are

$$\text{mean@4}_u = \frac{1}{|\mathcal{Q}_u|} \sum_{q \in \mathcal{Q}_u} \frac{1}{4} \sum_{j=1}^{4} J_{q,j}, \tag{4}$$

$$\text{pass@4}_u = \frac{1}{|\mathcal{Q}_u|} \sum_{q \in \mathcal{Q}_u} \mathbf{1}\left[\max_j J_{q,j} \geq 1\right]. \tag{5}$$

For pass@4, we reconstruct the binary success-count histogram from the logged raw-judge means, within-task standard deviations, and best/worst-of-four bootstrap statistics (1,000 resamples, seed 42). Each environment/checkpoint admits a unique histogram under this scoring and aggregation protocol. These results are pooled over complete tasks, not averaged equally across environments. Incomplete four-rollout groups are omitted. All 290 groups are complete except at update 30 (one ECHOML task missing), update 50 (one ECHOCARE task missing), and update 70 (one ECHOSTAY task missing). Table 10 gives the aggregate values in the left panel of Figure 5. Its right panel compares each environment at initialization and update 80, when all groups are complete.

Table 10: Synthetic validation across training. Mean@4 averages raw verifier scores; pass@4 counts tasks with at least one success in four attempts.

Update	Complete groups	Mean@4 (%)	Pass@4 (%)
0	290	39.05	64.83
10	290	51.21	72.41
20	290	59.91	80.00
30	289	56.83	78.20
40	290	56.47	75.52
50	289	57.96	78.55
60	290	58.71	77.24
70	289	59.08	78.89
80	290	60.34	77.59

Task/reference refinement and step-reward comparisons. Figure 6 (right) uses fixed nominal environment weights, rather than weighting environments by their realized number of complete validation groups. Let N_e be the nominal task count for environment e in Table 8, and let $\mathcal{Q}_{e,u}$ contain its complete four-rollout task groups at update u. The reported metric is

$$\text{mean@4}_u^{\text{mix}} = \sum_e \frac{N_e}{290} \left(\frac{1}{|\mathcal{Q}_{e,u}|} \sum_{q \in \mathcal{Q}_{e,u}} \frac{1}{4} \sum_{j=1}^{4} J_{q,j} \right). \tag{6}$$

The nominal counts are 46, 34, 39, 63, 67, and 41 for ECHOSTAY, ECHOFORGE, ECHOTUNES, ECHOBANK, ECHOCARE, and ECHOML, respectively. Their weights $N_e/290$ remain fixed across runs and checkpoints. Each within-environment mean uses only completed four-rollout groups and raw verifier scores, excluding training-time shaping and penalties. These comparisons use four sampled 40-turn rollouts at temperature 0.7 from the same live-site-SFT initialization.

The completed task groups can differ between runs. Fixed nominal weights hold the environment mixture constant, but do not make the completed task cohorts identical or correct for within-environment missingness. When all 290 task groups are complete, this estimator equals the pooled task mean defined above. Gains and endpoint differences are computed before rounding.

F.5 ONLINE-MIND2WEB TRANSFER

The synthetic update-80 export and its SFT initialization are compared on 281 matched Online-Mind2Web tasks using the same Browserbase-hosted browser configuration and judge. Hosted sessions mitigate CAPTCHA-related interruptions. The comparison contains one trajectory per checkpoint for each matched task. All 562 trajectories are readable; none are corrupt. All success, termination, and action-count summaries below use this same 281-task cohort rather than the full benchmark. Successes increase from 154 to 159, with 44 tasks solved only after RL and 39 solved only by the initialization. The improvement is 1.78 percentage points, with a reported 95% confidence interval of $[-4.6, 8.1]$ percentage points. This interval includes zero and does not establish a statistically significant accuracy gain. This evaluation is separate from the public-web continuation's 123-task validation set.

Table 11 reports success, action counts, and termination on the same matched cohort. The initialization terminates on 239 tasks, with 152 successes among them and two among the 42 non-terminated attempts. The RL checkpoint terminates on 271 tasks, with 159 successes among them and none among the ten non-terminated attempts. Mean action count falls by 4.7, while the median rises

from 22 to 24 and the 90th percentile falls from 100 to 64. Conditional accuracy compares different terminating subsets, not a fixed subset shared by both checkpoints.

Table 11: Online-Mind2Web transfer using Browserbase-hosted sessions. Both checkpoints use the same 281 matched tasks and judge. Action-count summaries include all matched trajectories; conditional means separate successful and unsuccessful attempts.

Metric	SFT initialization	Synthetic RL, update 80
Matched tasks	281	281
Successes	154	159
Task success (%)	54.80	56.58
Mean actions	36.2	31.5
Median actions	22	24
90th-percentile actions	100	64
Mean actions, successful	22.2	22.4
Mean actions, unsuccessful	53.1	43.4
Terminated attempts	239	271
Termination (%)	85.1	96.4
Non-termination (%)	14.9	3.6
Success given termination (%)	63.60	58.67
Success without termination (%)	4.76	0.00

F.6 LIVE-WEB CONTINUATION

We continue from the synthetic update-80 checkpoint on 400 curated public-web tasks, with 123 validation tasks and a 50-turn rollout limit. Training uses the trajectory reward in Eq. 3, centered and combined with local step feedback as in Appendix F.2. Validation instead uses one greedy rollout per task and the unweighted binary outcome only: process credit, step shaping, and incorrect-termination penalties are excluded. Table 12 gives the counts underlying Figure 7. Scored-task success stays near 21.5% through update 20, and the raw success count changes from 20 to 21 while the scored denominator varies. On the fixed intersection of 71 tasks scored at all three checkpoints, successes increase from 16 to 17 to 21 (22.54%, 23.94%, and 29.58%). The separate 78-task cohort with comparable records at updates 0 and 10 moves from 16 to 19 successes (8 gains and 5 regressions; reported McNemar $p \approx 0.58$). This pairwise cohort differs from the fixed 71-task cohort plotted in the figure. These are preliminary observations through update 20 at the September 23, 2026 snapshot.

A separate multimodal GPT-4.1 environment-classification call examines trajectory evidence for interruptions such as CAPTCHAs, website rate limits, and login pop-ups. Flagged trajectories are masked from learning and excluded from scored validation sets, rather than assigned policy-failure labels. Browser failures and missing required judgments also exclude rollouts; a missing optional step judgment instead contributes zero local shaping while retaining the trajectory reward (Appendix F.2). Across training updates 1–20, only 132–174 of 192 requested rollouts per update remain usable. Screening adds judging work without recovering the interaction spent on discarded trajectories. The resulting variation in usable training groups and validation coverage makes learning and progress measurement noisy.

Table 12: Preliminary live-web validation. Scored tasks exclude environment masks and browser or required-judge failures. Success/requested exposes coverage losses without attributing excluded rollouts to policy failure. The matched cohort contains the same 71 tasks scored at all three checkpoints.

Metric	Update 0	Update 10	Update 20
Requested tasks	123	123	123
Scored tasks	93	92	97
Successes	20	20	21
Success/scored (%)	21.51	21.74	21.65
Success/requested (%)	16.26	16.26	17.07
Matched tasks	71	71	71
Matched successes	16	17	21
Matched success (%)	22.54	23.94	29.58

G VERIFIER PROMPTS

The prompts below implement the read and write comparisons in Sec. 3.1. The read judge receives the goal, reference answer, and agent answer. The write judge receives the goal, reference state change, and `sqldiff` between D_0 and D_T. Neither receives screenshots or the action history. Each returns a binary score and a short explanation; mixed tasks combine both checks.

The prompts allow specified formatting differences and numeric rounding, such as $287.62 to $288, and accept additional writes when the required change is present. A passing judgment can therefore miss unintended side effects or an overly permissive semantic match. Database-derived references ground the comparison; its reliability still depends on the judge prompt and model.

G.1 READ TASKS

```
You are an evaluation judge. You compare an agent's answer to a reference answer for a
Website/application task.

The task asks the agent to look up information (a "read" task). You must decide
whether the agent's answer is semantically equivalent to the reference answer.

Rules:
- Minor formatting differences (extra whitespace, different date formats, quoting) are OK.
- Synonyms and paraphrases are OK as long as the meaning is the same.
- Monetary amounts may be rounded (e.g., $287.62 and $288 are equivalent; $181.63 and $182
are equivalent). Small rounding differences in prices, fees, or totals are acceptable because
 the agent reads displayed values from the UI which may round differently than the database.
- Numeric values may be expressed in equivalent units or human-readable formats. For example:
 157286400 bytes and "150 MB" are equivalent; 2500000000 and "2.5 billion" or "2,500,000,000"
 are equivalent; 42000000 and "42M" or "42,000,000" are equivalent. Accept any representation
 that conveys approximately the same quantity (within reasonable rounding for unit conversion
).
- Counts and IDs should match, but minor differences due to UI display rounding or pagination
 are acceptable if the values are close.
- Email addresses and names must match (case-insensitive).
- If the agent provides MORE information than needed but includes the correct answer, that
counts as correct.
- If the agent includes additional context (e.g., a date alongside a time) but the core
answer is correct, that counts as correct.
- If the agent's answer is completely wrong or missing the key information, it fails.

Return a JSON object with exactly these keys:
{
  "score": <int 0 or 1>,
  "reasoning": "<brief explanation>"
}

score guide:
  1 = answer is semantically correct
  0  = no required changes present or completely wrong changes
```

The user message is instantiated per task as:

```
## Task goal
```

```
$question

## Reference answer (ground truth)
$ref_answer

## Agent's answer
$agent_answer

Compare the agent's answer against the reference answer and return your JSON verdict.
```

G.2 WRITE TASKS

```
You are an evaluation judge for a Website/application. You determine whether
a task was completed correctly by comparing the intended state change with
the actual database changes.

You receive:
1. The task goal (what the agent was supposed to do)
2. The expected state change (reference_state_change)
3. The actual SQL diff (output of sqldiff showing what changed in the database)

The SQL diff contains SQL statements that would transform the BEFORE database
into the AFTER database. Common patterns:
- UPDATE table SET col=val WHERE ...  -> a field was changed
- INSERT INTO table ...               -> a new row was added
- DELETE FROM table WHERE ...          -> a row was removed

Rules:
- The agent may have made additional changes beyond what was required (e.g.,
  marking an email as read while also starring it). Extra changes are OK as
  long as the required change was made.
- If the diff is empty, the agent made no changes -> likely a failure.
- Focus on whether the REQUIRED changes (from reference_state_change) are
  present in the diff.

Return a JSON object with exactly these keys:
{
  "score": <int 0 or 1>,
  "reasoning": "<brief explanation of what matched and what didn't>"
}

score guide:
  1  = all required changes are present
  0  = no required changes present or completely wrong changes
```

The user message is instantiated per task as:

```
## Task goal
$goal

## Expected state change (reference)
$ref_state_change

## Actual database diff (sqldiff output)
$sql_diff

Evaluate whether the required changes were made. Return your JSON verdict.
```

H CAPABILITY-WORLD WIDGET CATALOGUE

Tables 13 and 14 summarize interaction types in the nested-filter and datepicker worlds (Sec. 4.2). Held-out families are used only for evaluation. The rows describe representative control types, not an exhaustive inventory of the generated widget families.

Table 13: Representative nested-filter controls. Training covers 20 families over 200 frontends in 15 visual styles; the held-out set covers 9 compound families over 100 frontends in 9 styles.

Training (in-distribution)		**Held out**	
Control	**Interaction**	**Control**	**Interaction**
Calculator and menu	Compute a value, pick from a dropdown	Booking search	Destination with check-in and check-out dates
Calendar filter	Pick a date to filter by	Override calculator	Calculator with default overrides
Filter dialog	Open an "all filters" modal	Calendar combination	Calendar plus secondary controls
Search field	Type a free-text query	Cost calculator	Region and usage cost estimator
Facet checkboxes	Tick multiple category boxes	Property facets	Beds, baths and price facets
Range slider	Set a minimum and maximum	Modal filter flow	Multi-step modal filters
On/off toggles	Boolean switches	Strict facets	Exact-match facet filter
Filter chips	Click pill toggles on and off	Adoption filters	Species, compatibility and sex
Calculator field	Type or derive a numeric threshold	Full property panel	Complete real-estate filter panel
Active-filter bar	Chips of the applied filters		

Table 14: Representative datepicker controls. Training covers 6 core types over 10 contexts, with 100 frontends in 6 visual styles; the held-out set covers 10 widget families over 36 scenarios, with 80 frontends in 9 styles.

Training (in-distribution)		**Held out**	
Control	**Interaction**	**Control**	**Interaction**
Date and time	A date plus a time of day	Calendar heatmap	Grid coloured by density
Single date	Pick one calendar date	Scroll wheel	Spinning wheel columns
Date range	A start and end span	ISO week picker	Select a whole calendar week
Bounded date	Date within minimum and maximum limits	Multi-date	Pick several dates at once
Date of birth	Birthdate entry	Duration stepper	Step an ISO duration up or down
Month and year	Month and year only	Time-only clock	Hours, minutes and seconds, no date
		Quarter picker	Select a fiscal quarter
		Fiscal-year picker	Pick a fiscal year by decade
		Timeline slider	Slide along a date axis