AGENT WORLD MODEL IN THE WILD: COMPUTER-USE AGENT ENVIRONMENT SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent learning is shifting from learning from human data toward learning through interaction with the world. However, scaling such learning directly in real-world computer systems is challenging and can introduce substantial security risks. We therefore propose **A**gent **W**orld **M**odel in the **Wild** (WildAWM), a scalable pipeline for synthesizing controllable environments and tasks for training general computer-use agents (CUAs). Whereas existing approaches typically specialize in a single agent interaction interface, we construct executable environments that jointly support graphical user interfaces, model context protocol services, and command-line interfaces, allowing agents to flexibly combine visual interaction, tool use, and code execution. WildAWM is entirely powered by open-source models, demonstrating that high-quality computer-use environments can be generated without proprietary frontier models. Beyond environments, we address three recurring challenges in task synthesis: *ensuring solvability, maintaining sufficient difficulty, and producing reliable verifiers*, by introducing execution-based quality gates that combine heterogeneous agent rollouts, cross-judge consistency checks, and counterfactual probes of judgment robustness. The resulting pipeline synthesizes 341 long-horizon environment-task pairs that can require over 100 agent steps to solve and remain challenging even for frontier models, with GPT-6-Astra solving only 60% of them. We train two Qwen models of different sizes exclusively in these synthetic worlds. Both models achieve substantial gains across 8 out-of-distribution benchmarks spanning web, claw, and computer-use agents, demonstrating that learning in WildAWM's synthesized worlds generalizes broadly to unseen computer-use tasks.

1 INTRODUCTION

Large language model (LLM) powered agents are shifting from learning from humans to learning from the world (Silver & Sutton, 2025). Early agents learned from human data, which are expensive to collect and limited by demonstrator's ability. Recent agents instead gain much of their capability through reinforcement learning (RL) in executable environments (Team et al., 2026; Zeng et al., 2026; Xu et al., 2026), making environment supply a key bottleneck. However, training directly in the real world is costly and risky: states are hard to reset, online services change over time, and exploratory actions may have unintended consequences.[1] Manually built environments restore control and verifiability (Yao et al., 2024; Zhou et al., 2024; Xie et al., 2024), but remain expensive and hard to scale. Synthetic environments and tasks offer a path to controllable, large-scale online RL.

Computer-Use Agents (CUAs) represent a broad setting for environment synthesis, as they operate across applications, websites, and local files on a personal computer. A single task may be completed through graphical user interface (GUI) interaction, command-line interface (CLI) operations, or calls to model context protocol (MCP) servers, making interface selection and coordination an important capability. Existing synthesis pipelines often target individual interaction modes, including browser use (Zhou, 2026; Wu et al., 2026), tool calling (Du et al., 2026; Wang et al., 2026d), and terminal workflows (Li et al., 2026; Zhu et al., 2026). Prior CUA work, however, often focuses primarily on GUI interaction (Aggarwal et al., 2026; Wang et al., 2026a). In this paper, we adopt a broader view of computer use spanning common interfaces on a personal computer, since real computer-use tasks are defined by what can be accomplished rather than by a single interaction modality.

[1] For example, see OpenAI's recent incident report on real-world RL at this url.

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.

Table 1: Comparison of related methods. PC denotes personal computer. * WebHarbor's websites provide seed GUI substrate and image content. 💻 denotes agents equipped with GUI and MCP tools.

Method	Scope	Observation	Interfaces			Synthesis Source	Synthesis Models	Target Agent
			GUI	MCP	CLI			
AWM (Wang et al., 2026d)	Single App	Text	×	✓	×	From-Scratch	GPT-5	ReAct-style Tool-use Agent
WebArena-Infinity (Zhou, 2026)	Single App	Text	✓	×	×	Reference-Grounded	Opus 4.6	ReAct-style Web Agent
Gym-Anything (Aggarwal et al., 2026)	Single App	Text+Image	✓	×	×	Existing Software	Opus 4.5/4.6	ReAct-style GUI Agent
TMax (Ivison et al., 2026)	Multi App	Text	×	×	✓	Seed-Grounded	Gemini-3-Pro	mini-SWE-agent
Synth. Computers (Ge et al., 2026)	PC	Text	×	×	✓	From-Scratch	Opus 4.6	Claude Code 💻
CUA-Gym (Wang et al., 2026a)	Multi App	Text	✓	×	×	Reference-Grounded	Sonnet 4.6	ReAct-style GUI Agent
CUA-Universe (Shi et al., 2026)	Single App	Text+Image	✓	×	✓	Existing Software	GPT-5.6	Path-Steer GUI+CLI Agent
WildAWM	PC	Text+Image*	✓	✓	✓	From-Scratch + Seed-Grounded*	OpenSource	OpenCode 💻

Synthesizing environments and tasks for this broader CUA setting introduces challenges beyond prior synthesis work. Existing approaches often synthesize visually simplified applications and build each application as an isolated environment. In contrast, realistic computer use spans a connected digital workspace in which websites, applications, MCP services, and local files coexist within an executable personal computer environment. Meanwhile, synthesized tasks should be *solvable* under the synthesized environment state, sufficiently *challenging* to provide useful learning signals, and paired with *reliable* verifiers to distinguish successful completion from incorrect outcomes, all without relying on simplistic assumptions such as "a frontier model is necessarily correct."

To address these challenges, we propose WildAWM, a scalable synthesis pipeline for general CUAs. Starting from user personas and daily activities, WildAWM constructs executable personal-computer environments with websites, MCP services, and local files, then synthesizes cross-agent-interface (XAI) tasks grounded in these worlds. We introduce execution-based multi-agent quality gates, where heterogeneous agents from different models and harnesses execute each task, providing evidence of solvability, difficulty, and judge consistency. To further test verifier robustness, we exploit LLM sycophancy (Sharma et al., 2024) through counterfactual probes with deliberately incorrect outcome feedback. The resulting environments, tasks, and verifiers provide executable worlds and rewards for agentic RL. Notably, WildAWM uses open-source models throughout synthesis, whereas existing works often rely on proprietary models (Aggarwal et al., 2026; Zhou, 2026; Wang et al., 2026a). This improves reproducibility, transparency, and accessibility, while showing that open-source models alone can synthesize high-quality environments and challenging tasks for general CUA training.

Using WildAWM, we synthesize 341 environment–task pairs, each requiring at least two interface types, with long-horizon workflows that may require over 100 agent steps, and even frontier model GPT-6-Astra (OpenAI, 2026c) solves only about 60%. We further perform online RL on two Qwen models, Qwen3.6-35B-A3B (Qwen Team, 2026) and Qwen3.5-4B (Team, 2026), using OpenCode CLI harness (anomalyco, 2025) equipped with GUI and MCP tools, and find that training exclusively in these synthetic worlds yields substantial gains across all eight out-of-distribution (OOD) benchmarks spanning web, claw, and computer-use agents. Our contributions are threefold: (1) a scalable, fully open-source pipeline for synthesizing CUA environments spanning multiple agent interfaces; (2) execution-based gates for synthesizing solvable, challenging tasks with reliable verifiers, together with the release of 341 artifacts; and (3) demonstration that training exclusively in worlds synthesized by open-source models can substantially improve the generalization of computer-use agents.

2 Related Work

Computer-Use Agents. Computer-use agents interact with applications through GUIs (Cheng et al., 2024; Xu et al., 2024; Wu et al., 2025) and programmatic representations such as HTML and accessibility trees (Zhou et al., 2024; He et al., 2024). GUI agents have been studied for web navigation (He et al., 2024; Zheng et al., 2024; Yang et al., 2026a), desktop automation (Agashe et al., 2025; Zhang et al., 2024; 2025a; Wang et al., 2026b), and multi-application workflows (Tan et al., 2024; Qin et al., 2025; OpenAI, 2025b; Yang et al., 2026b). Recent work expands this action space with CLI (Zhou et al., 2026), coding (Song et al., 2026), and MCP tools (Jia et al., 2025), while harness-based agents expose heterogeneous tools through coding and claw-agent frameworks (Yu et al., 2026; OpenClaw contributors, 2026; Anthropic, 2025; anomalyco, 2025). These trends suggest a broader view of computer use in which the agent is not bound to a particular interaction modality.

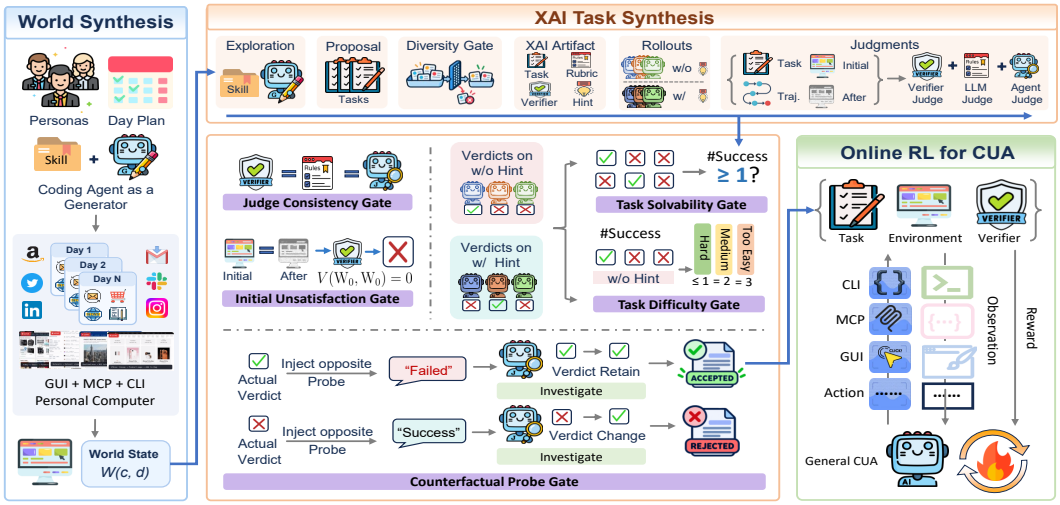

Figure 1: Overview of WildAWM. The synthesis pipeline includes two stages: (1) *World Synthesis*, where we use the coding agent to construct persona-driven, multi-day world states for computer environments that expose GUI, MCP, and CLI interfaces. (2) *XAI Task Synthesis*, where we generate cross-interface tasks through 5 designed quality gates for task solvability, difficulty, and verifier reliability. The synthesized $\{\texttt{task}, \texttt{environment}, \texttt{verifier}\}$ triplet enables training general CUA.

In this work, we adopt this view and consider general CUAs that can jointly access GUI, CLI, and MCP interfaces as needed to complete any tasks on a personal computer.

Environment Synthesis. The growing demand for interactive agent training has motivated automatic construction of executable environments. Recent work synthesizes executable environments for web interaction (Zhou, 2026; Wu et al., 2026), tool use (Wang et al., 2026d; Du et al., 2026; Dong et al., 2026), terminal interaction (Zhu et al., 2026; Ivison et al., 2026; Li et al., 2026), and software engineering (Jain et al., 2025; Yang et al., 2025; Chen et al., 2026). For CUAs, GUI-GENESIS reconstructs lightweight web applications (Cao et al., 2026), WebArena-Infinity and CUA-Gym synthesize web environments and verifiable tasks (Zhou, 2026; Wang et al., 2026a), Gym-Anything converts existing software into agent environments (Aggarwal et al., 2026) rather than building the environments, and CUA-Universe augments desktop applications with CLI access (Shi et al., 2026). Other work synthesizes user-specific computers with productivity artifacts (Ge et al., 2026). As compared in Table 1, WildAWM constructs a unified personal-computer world with joint GUI, MCP, and CLI access, combining WebHarbor's visual web substrate (WebHarbor Team, 2026) with synthesized services, files, and application data. Notably, WildAWM uses open-source models instead of proprietary frontier models, which can improve transparency and accessibility for the community.

Task Synthesis. Task generation has progressed from standalone instruction synthesis (Wang et al., 2023; Xu et al., 2023; Taori et al., 2023) to tasks grounded in executable environments (Sun et al., 2024; Pahuja et al., 2025; Wang et al., 2026c). Prior methods derive tasks from observed interactions (Sun et al., 2024; Su et al., 2025), environment exploration (Pahuja et al., 2025; Wang et al., 2026c), or programmatic generation (Yang et al., 2025). More recent methods couple task synthesis with executable verification: WebArena-Infinity synthesizes difficulty-graded browser tasks with programmatic verifiers (Zhou, 2026), Agent World Model (AWM) (Wang et al., 2026d) exploits executable environment states to construct reliable rewards, CUA-Gym (Wang et al., 2026a) co-generates task, environment state, and reward as verified triples (Wang et al., 2026a). Beyond these approaches, WildAWM introduces a quality-control pipeline using heterogeneous agent rollouts, cross-judge consistency, and counterfactual probes to validate task solvability and difficulty while stress-testing the reliability of synthesized verifiers, all with open-source models.

3 Agent World Model in the Wild

As shown in Figure 1, WildAWM uses open-source models to synthesize executable environments and tasks for training general CUAs. Each synthesized instance contains a personal-computer environment initialized from a world state W, a task τ, and a verifier V that evaluates task completion.

3.1 WORLD SYNTHESIS

We define a world as the accumulated state of a personal computer which comprises web applications, MCP-accessible services, and local files that record a person's activities. To expand the range of multimodal GUI interaction, we integrate 15 mirror websites from WebHarbor (WebHarbor Team, 2026) which provides website mirrors with rich visual content. We first synthesize a persona and expand its ongoing activities into sequential day plans. Then, a coding agent materializes these plans day by day, with each day extending the previous state. This procedure constructs applications and services, and produces diverse user histories, application data, and files. We denote the cumulative state of persona c after day d as $W(c, d)$, which initializes and resets the environment state.

Persona and Day Planning. We first prompt the generator, a coding agent using GLM-5.2 (Zeng et al., 2026) within OpenCode (anomalyco, 2025), to synthesize personas from scratch. Each persona c describes an occupation, named contacts, frequently used websites, and ongoing activities spanning multiple days. To promote diversity, we compare each candidate with accepted personas in terms of life domain and multi-day storylines, then regenerate repeated candidates with feedback on identified similarities. For each accepted persona, the generator further develops these storylines into a sequence of day plans. Each plan specifies activities, the people and digital artifacts involved, the required applications and services, and the corresponding databases and files. Planning proceed one day at a time: the generator would inspect previous day's state $W(c, d-1)$ to maintain continuity with ongoing activities before planning day d. Then, the subsequent materialization stage would reuse existing application and service implementations and synthesize any artifacts that are missing.

Skill-Enhanced Generation. Given the day plan, the generator first matches its activities and required services against a growing library of reusable event skills and application implementations. Each event skill specifies the interfaces involved, the state changes to produce, and the consistency constraints across the resulting records and files. For example, a skill for a student enrolling in a Coursera course and completing assignments can encode the corresponding workflow and the required database and file updates. The generator synthesizes a new skill if a plan introduces an activity not being covered by any existing skill. Similarly, when a required MCP service is unavailable, it implements the corresponding database schema and backend, MCP servers. These skills and service implementations will be added to the shared library for reuse in subsequent days and personas. The generator starts from a clean state on day 0 or a copy of $W(c, d-1)$ thereafter. It inspects the actual schemas and existing state, then writes a deterministic, replayable program that updates databases for MCP services and websites, and local files to produce $W(c, d)$. We retain such state only if two independent replays produce the same state, and all services and websites remain healthy. Further, we launch another coding agent to verify the state is consistent with the day plan, and provide feedback for necessary revisions. Our synthetic world artifacts are summarized in Table 7 in Appendix A.3.

3.2 XAI TASK SYNTHESIS

Beyond environment synthesis, task generation faces three key challenges for agentic learning: *ensuring solvability*, *maintaining sufficient difficulty*, and *generating reliable verifiers*. We address them with an execution-based multi-agent validation loop, where heterogeneous open-source agents attempt each task and five quality gates use the resulting evidence for acceptance or revision.

Task Generation Given an initial world state $W_0 = W(c, d)$, the skill-enhanced generator explores the live environment and proposes cross-agent interface (XAI) tasks grounded in the available application data, service records, and local files. Each task requires the agent to use at least two interfaces among GUI, MCP, and CLI. A diversity gate selects proposals that are least similar to tasks already accepted or being developed by concurrent workers. For each selected proposal, the generator produces a task specification τ, an executable verifier V, a rubric R, and a hint h, and a document that decomposes τ into an ordered sequence of sub-tasks. The verifier takes the initial world state W_0, the final world state W, and the agent's final answer a as inputs. It evaluates a predefined checklist of N binary criteria, where each criterion is given by $v_i(W_0, W, a) \in \{0, 1\}$. We define task success as satisfying all criteria: $V(W_0, W, a) = \prod_{i=1}^{N} v_i(W_0, W, a)$. Individual criterion scores can provide granular feedback and value estimates for long-horizon RL training.

Rollouts & Judgments After generating a task and its verifier, we use heterogeneous agents to attempt each task: successful rollouts provide evidence of solvability, success rates characterize empirical

difficulty, and their trajectories and outcomes help validate the verifier. Specifically, we collect six rollouts per task, half with hints, from 18 model–harness pairs formed by three open-source models—GLM-5.2 (Zeng et al., 2026), DeepSeek-V4-Flash (Xu et al., 2026), and MiniMax-M3 (MiniMax Team, 2026)—and six harnesses, comprising three coding agents (OpenCode (anomalyco, 2025), Codex (OpenAI, 2025a), Claude Code (Anthropic, 2025)) and three claw agents (Nanobot (Ren et al., 2026), OpenClaw (OpenClaw contributors, 2026), Hermes Agent (Nous Research, 2026)).

We further validate each rollout through three levels of judgment, which we view as test-time scaling for verification: a verifier V the generator just synthesized; a LLM judge that semantically assesses the task, rubric, trajectory, and verifier results; an agent judge using OpenCode that actively inspects the initial and final environments in multi turns (Zhuge et al., 2024). Although agent judge can be costly, we only use it during synthesis to cross-check judgments and identify cases requiring verifier revision. The synthesized programmatic verifier is finally used as the reward function during RL.

Quality Gates The collected rollouts and judgments provide execution evidence but do not by themselves establish task validity. We propose five quality gates to assess task solvability, difficulty, and verifier reliability, accepting candidates that pass all gates.

- **Judge Consistency:** For rollout j, let W_j and a_j denote its final state and answer, and let $V_{\text{LLM}}^{(j)}$ and $V_{\text{agent}}^{(j)}$ denote the LLM and agent judges' verdicts. The judge consistency gate requires $V(W_0, W_j, a_j) = V_{\text{LLM}}^{(j)} = V_{\text{agent}}^{(j)}$ for every $j \in \{1, \ldots, 6\}$. Disagreements and the associated evidence will be returned to the generator to revise the rubric R and verifier scripts V.
- **Initial Unsatisfaction:** This gate requires $V(W_0, W_0, \emptyset) = 0$, where $\emptyset$ denotes the absence of any agent action, rejecting tasks already considered complete in the initial state.
- **Task Solvability:** Agreement among judges does not establish solvability if all rollouts fail. The task solvability gate requires at least one successful rollout, $\sum_{j=1}^{6} V(W_0, W_j, a_j) \geq 1$. Each successful trajectory must also use every interface declared during task generation, providing execution evidence of the intended cross-interface workflow.
- **Task Difficulty:** The task difficulty gate estimates empirical difficulty using $k = \sum_{j \in \mathcal{U}} V(W_0, W_j, a_j)$, where $\mathcal{U}$ indexes the three rollouts without hint. Tasks with $k \leq 1$ are labeled *hard*, those with $k = 2$ are labeled *medium*, and those with $k = 3$ are labeled *too easy* which will be returned to the generator to increase difficulty.
- **Counterfactual Probe:** The preceding gates rely partly on model-based judgments, including agent judge's feedback for revising the verifier V. Given LLMs' susceptibility to sycophancy (Sharma et al., 2024), we add a counterfactual probe to test the stability of this feedback under incorrect outcome claims. Specifically, given a rollout with verdict $V(W_0, W, a)$, we present the agent judge with the counterfactual claim $1 - V(W_0, W, a)$ and ask it to reinvestigate the execution evidence. Let $V_{\text{probe}} \in \{0, 1\}$ denote the judge's verdict after reinvestigation. The candidate passes only if $V_{\text{probe}} = V(W_0, W, a)$; otherwise, it fails.

In summary, candidates that fail any gate are returned to the generator with diagnostic evidence for up to two revision rounds, after which they must re-pass all gates. Accepted tasks are added to the diversity pool, while failed ones provide lessons to be absorbed into generator's evolving skills. Table 9 and Figure 6 in Appendix A.4 summarize synthesis statistics and candidate progression. During rollouts, XAI tasks have a mean success rate of only 21.3%, and even GLM-5.2 sometimes requires more than 100 steps to complete a task. Consistent with the solvability and difficulty gates, every accepted task is solved by at least one agent, while none is solved by all agents. XAI tasks consists of 4.2 sub-tasks and 3.1 interface switches on average, and 74.5% revisit a previously used interface, requiring agents to preserve state across interface transitions rather than completing the task within a single interface. Appendix C illustrates these properties with concrete examples.

3.3 General CUA Training

We train general CUAs in our synthesized worlds using GRPO (Shao et al., 2024), with a validated verifier that provides dense but outcome-based rewards. For rollout j of task τ, let (W_j, a_j) denote its final world state and answer, and let $e_j \in \{0, 1\}$ indicate a policy format failure (e.g., invalid tool call). We assign $r_j = -1$ when $e_j = 1$. For $e_j = 0$, the reward combines with checklist scores v_i:

$$r_j = \min \left\{ 1,\, V(W_0, W_j, a_j) + \frac{0.1}{N} \sum_{i=1}^{N} v_i(W_0, W_j, a_j) \right\}, \tag{1}$$

where N is the number of checklist criteria. Thus, successful tasks receive unit reward, while checklist scores provide partial credit for incomplete trajectories, yielding denser signals for long-horizon learning. However, small models often produce zero-variance reward groups, providing no GRPO learning signal. Inspired by DAPO (Yu et al., 2025), we maintain for each task τ an exponential moving estimate (Roberts, 2000) of the required rollout multiplier $\lceil K/G \rceil$, where K is the rollout pool size and G the target group size. The multiplier is initialized at 1, capped at 3, and updated with weight 0.8 after each optimization step. When $K > G$, we select a group from possible $\binom{K}{G}$ groups containing samples from both the lower and upper reward halves to encourage reward diversity. Let μ_τ and σ_τ denote the mean and population standard deviation of rewards in a completed rollout pool of size K. Then, the trajectory advantage is defined as

$$A_j = \begin{cases} (r_j - \mu_\tau)/\sigma_\tau, & \sigma_\tau > 0, \\ -K^{-1/2}, & \sigma_\tau = 0 \text{ and } e_j = 1 \; \forall j, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where the second case is designed to provide format-related learning signal even when all rollouts receive the same format error reward, which is common for long-horizon rollouts during the early stages of training. Note that the rollout will be terminated once a format error occurs, and gradients are restricted to the generated tokens in the failing step. More details are provided in Appendix A.5.

4 Experiments

Dataset & Benchmark. We synthesized 341 XAI tasks by WildAWM, and we randomly split them into 100 for SFT, 208 for RL, and 33 for in-domain evaluation. To evaluate the OOD generalization of our agent, we further include 8 benchmarks across 3 categories of agent tasks: WebArena (WA) (Zhou et al., 2024) and Online-Mind2Web (O2W) (Xue et al., 2025) for web agents; QwenClawBench (QCB) (Qwen Team & Alibaba Data, 2026) and ClawEval (CLE) (Ye et al., 2026) for claw agents; CUA-Gym (CUG) (Wang et al., 2026a), OSWorld-Verified (OSV) (Xie et al., 2024), OSWorld 2.0 (OS-2) (Yuan et al., 2026), and Agents' Last Exam (ALE) (Sun et al., 2026) for computer-use agents.

Model & Baselines We use two base models at different scales for training: Qwen3.6-35B-A3B (Qwen Team, 2026) and Qwen3.5-4B (Team, 2026). We compare our trained models against a range of frontier models, including GPT-6-Astra (OpenAI, 2026c), GPT-5.6-Sol (OpenAI, 2026b), GPT-5.5 (OpenAI, 2026a), Claude Opus 5 (Anthropic, 2026b), Claude Opus 4.8 (Anthropic, 2026a), and GLM-5.2 (Zeng et al., 2026). We also include numbers reported in prior computer-use agent work for reference, including CUA-Gym (Wang et al., 2026a) and OpenForgeRL (Yu et al., 2026).[2]

Implementation Details We use 8 NVIDIA B200 GPUs for both synthesis and training. During synthesis, we use GLM-5.2 as both the generator and the agent-as-a-judge with OpenCode harness (anomalyco, 2025). For evaluation, we use OpenCode equipped with GUI (e.g., `click` and `type`) and MCP (e.g., Playwright) tools, and enable high reasoning effort for the model. The maximum number of agent steps is set to 256 for most benchmarks and 512 for OSWorld 2.0 and ALE. The context window is set to 100K tokens with automatic context compaction. For training, we first construct a seed dataset by collecting 100 GLM-5.2 trajectories on the SFT split, and warm-start the Qwen base models with SFT for 3 epochs. We then perform RL for up to 64 training steps, using a batch size of 8 tasks and $G = 4$ rollouts per task. Environment management and RL training are built on Orchard (Peng et al., 2026) and slime (Zhu et al., 2025). More details are in Appendix A.

Main Results Table 2 summarizes the main results. We highlight three key findings. (1) *Broad OOD generalization.* Training exclusively in WildAWM's synthesized worlds improves both model scales on all eight benchmarks, spanning web, claw, and computer-use agents. The 35B-A3B and 4B models gain $+10.58$ and $+8.21$ points on average, respectively, demonstrating broad transfer across unseen tasks and scenarios. (2) *Larger models benefit more.* The 35B-A3B model improves more than the 4B model on average ($+10.58$ vs. $+8.21$), with particularly large gains on CLE ($+16.67$), OSV ($+16.06$), WebArena ($+13.72$), and CUG ($+12.22$). This suggests that the benefits of WildAWM's long-horizon, cross-interface supervision persist, and may even increase, at larger model capacity. (3) *Narrowing the gap to frontier models.* WildAWM raises the 35B-A3B model's average score from 42.84 to 53.42, substantially reducing its gap to the listed frontier models. The resulting model further matches GLM-5.2 on WebArena (67.26) and exceeds GPT-5.5 on O2W (80.77 vs. 72.22).

[2]For reference only, due to their unavailable checkpoints or evaluation with traditional GUI harnesses.

Table 2: Main results on OOD benchmarks. We report pass rates for O2W, WA, and CLE, and mean score for the other benchmarks. † Numbers reported in the official papers.

		Web Agent		Claw Agent		Computer-Use Agent				
Model	Harness	O 2 W	WA	Q C B	C L E	C U G	O S V	OS - 2	A L E	Avg.
Frontier Models										
GLM-5.2	OpenCode	86.67	67.26	64.44	59.67	78.78	62.46	45.00	41.39	63.21
Opus-4.8	OpenCode	91.01	71.24	64.74	76.33	**84.52**	68.58	48.32	46.11	68.86
Opus-5	OpenCode	**96.60**	**75.66**	55.02	77.67	84.39	**74.64**	52.79	49.85	70.83
GPT-5.5	OpenCode	72.22	70.80	62.87	63.67	82.71	61.46	47.12	52.39	64.16
GPT-5.6-Sol	OpenCode	86.72	67.70	61.34	64.33	82.83	58.48	46.89	53.45	65.22
GPT-6-Astra	OpenCode	92.78	74.34	**65.21**	**80.33**	83.56	71.28	**60.77**	**56.58**	**73.11**
4B-Level Models										
Qwen3.5	OpenCode	26.39	30.09	44.06	26.67	21.39	18.18	0.87	7.00	21.83
WildAWM	OpenCode	**45.65**	**39.82**	**48.37**	**36.67**	**37.35**	**22.48**	**1.03**	**8.96**	**30.04**$_{+8.21}$
35B-A3B-Level Models										
Qwen3.6	OpenCode	69.60	53.54	57.86	38.00	53.73	33.16	13.22	23.57	42.84
OpenForgeRL†	OpenClaw	–	–	33.7	45.1	–	–	–	–	–
CUA-Gym†	R-GUI	–	44.5	–	–	–	62.1	–	–	–
WildAWM	OpenCode	**80.77**	**67.26**	**61.44**	**54.67**	**65.95**	**49.22**	**21.73**	**26.31**	**53.42**$_{+10.58}$

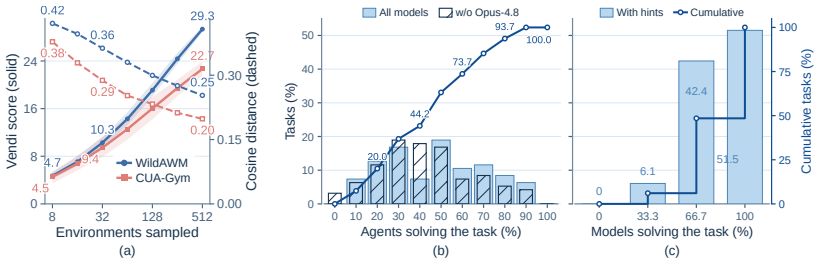

Figure 2: (a) Environment diversity using Qwen3-Embedding-8B (Zhang et al., 2025b) representations of environment artifacts. Shaded bands are 95% intervals of the Vendi score (Friedman & Dieng, 2022). (b) Task solvability across 24 model–harness configurations: the three models and six harnesses in Sec. 3.2 (Rollouts & Judgments), plus Opus-4.8. (c) Solvability with hints w/o Opus-4.8.

5 Analysis

5.1 Synthesis Quality

Environment. A WildAWM's environment state accumulates a persona's activities over multiple days, with a median size of 52.4K tokens, while CUA-Gym uses task-specific injected states with a median size of only 1.1K tokens. Despite its substantially larger environment scale, WildAWM is also more diverse. As shown in Figure 2a, we find that WildAWM achieves consistently higher Vendi score across all sample sizes. Its higher nearest-neighbor cosine distance (0.25 vs. 0.20) further indicates lower redundancy. Notably, we achieve these gains using only open-source models, whereas most existing work relies on proprietary frontier models for synthesis.

Task & Verifier. We answer these important task synthesis questions: (1) **Are XAI tasks solvable?** Rather than relying on a single strong model as evidence of solvability, we evaluate each task with 24 heterogeneous agents (model–harness pairs). As shown in Figure 2b, every task is solved by at least one agent, while the three open-source models alone used during synthesis already solve over 95% of tasks. Figure 2c further evaluates our hints using only these open-source models. With hints, every task is solved by at least one of the three models, and 51.5% by all three, providing stronger evidence that XAI tasks are solvable through executable paths. (2) **Are XAI tasks challenging, even for frontier models?** We evaluate frontier and open-source models on synthesized tasks to assess task difficulty in Figure 3a, and find that GPT-6-Astra achieves only about 60% pass@1, while our trained agents approach the performance of GLM-5.2 with substantially fewer parameters. Figure 3b shows that difficulty persists under test-time scaling: even after eight attempts, GPT-6-Astra still fails on 27.3% of tasks and pass^k drops rapidly, indicating that open-source-synthesized tasks remain

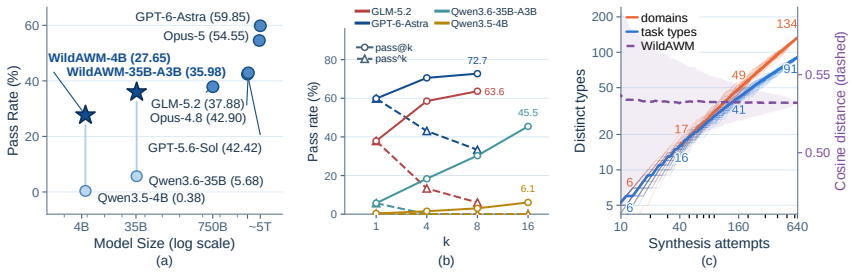

Figure 3: (a)/(b) Performance on WildAWM validation set, and proprietary model sizes are estimated. (c) Rarefaction of the 341 accepted tasks by LLM-classified task types and domains.

Table 3: Study on verifier reliability. Cons.: the fraction of rollouts receiving the same verdict from both sides. κ: Cohen's kappa.

Agent Judge[3]	Cons.	Precision	Recall	F1	κ
GLM-5.2	0.95	0.98	0.93	0.95	0.90
GPT-5.6-Sol	0.92	0.85	0.98	0.91	0.83
Claude-Opus-5	0.92	0.98	0.88	0.93	0.84
Judge majority	0.95	0.98	0.93	0.95	0.90
Inter-Judge Agreement					
GLM-5.2 vs. Opus-5	0.97	1.00	0.95	0.97	0.94
GPT-5.6-Sol vs. Opus-5	0.89	1.00	0.80	0.89	0.79

Table 4: Ablation of quality gates on four benchmarks, reported as success rate (%). Val.: our synthesized in-domain validation set.

Model	O2W	QCB	OSV	Val.	Avg.
Qwen-3.5-4B	26.39	44.06	18.18	0.38	22.25
WildAWM + SFT	38.49	**50.17**	17.17	**21.59**	**31.86**
w/o Init. Unsat.	33.21	42.36	17.71	3.05	24.08
w/o Judge Cons.	29.10	37.72	19.62	3.79	22.56
w/o Task Solv.	**44.87**	44.15	12.12	15.91	29.26
w/o Task Diff.	39.05	42.29	16.54	7.58	26.37
w/o CF Probe	31.37	42.55	**24.09**	13.64	27.91

challenging enough and could even provide useful training signals for frontier models. (3) **Are synthesized verifiers reliable?** We compare our verifier against multiple agent-as-a-judge models on 100 randomly sampled rollouts in Table 3. Our verifier achieves 95% agreement with the majority, with a Cohen's κ of 0.90, suggesting that our verifier is reliable within natural variation of stronger but heavier agent-as-a-judge judgments. (4) **Is the synthesis pipeline scalable?** As shown in Figure 3c, synthesis continues to expand task coverage, reaching 134 domains and 91 task types at a 53% acceptance rate without clear saturation. Meanwhile, mean pairwise cosine distance remains stable as the pool grows, suggesting that increased coverage does not come from increasingly similar tasks.

5.2 Ablations on Quality Gates

To isolate the contribution of each synthesis gate, we synthesize SFT data with one gate removed at a time and train Qwen3.5-4B model under an identical settings. As shown in Table 4, the full quality-control strategy achieves the best overall performance, while removing any gate reduces the average by 2.60–9.30 points. Judge Consistency and Initial Unsatisfaction are particularly important: removing them drops the average to 22.56 and 24.08, respectively, and reduces performance on in-domain validation set from 21.59 to only 3.79 and 3.05. Task Difficulty also contributes substantially, with its removal decreasing the validation score to 7.58. These results show that five gates are complementary: together they produce more reliable training data and substantially improve both in-domain task learning and generalization performance on OOD benchmarks.

5.3 Ablations on RL

We ablate key components of our RL recipe to measure their impact in Table 5, and find that removing format penalty cause significant drop in WA and OSV, while its higher CLE pass rate is not matched by a higher CLE mean score (69.89 vs. 69.45 WildAWM), i.e., more tasks end just above the pass threshold. We conjecture that the format reward is particularly important for GUI agents, where actions must be correctly parsed to interact with visual interfaces. Without checklist partial credit, performance decreases on most benchmarks, indicating that the denser reward improves credit assignment. Notably, removing the warm start causes the largest degradation, highlighting the importance of a strong initialization for effective RL training at this small model scale.

[3]Each judge model represents multiple runs with different harnesses. GLM-5.2: OpenCode, Claude Code, and Codex; GPT-5.6-Sol: OpenCode and Codex; Claude-Opus-5: OpenCode and Claude Code.

Table 5: Ablations on RL training with Qwen3.6-35B-A3B as the base model.

Method	Web Agent		Claw Agent		Computer-Use Agent				Avg.
	O2W	WA	QCB	CLE	CUG	OSV	OS-2	ALE	
WildAWM	**80.77**	**67.26**	61.44	54.67	65.95	**49.22**	**21.73**	26.31	**53.42**
w/o Format Reward	77.91	58.85	**62.17**	**61.00**	66.43	40.83	19.31	**30.93**	52.18$_{-1.24}$
w/o Checklist Reward	73.38	64.89	55.06	59.18	66.53	44.29	18.51	27.51	51.17$_{-2.25}$
w/o SFT Warm Start	66.29	57.96	59.74	51.67	**68.34**	42.78	13.54	24.59	48.11$_{-5.31}$

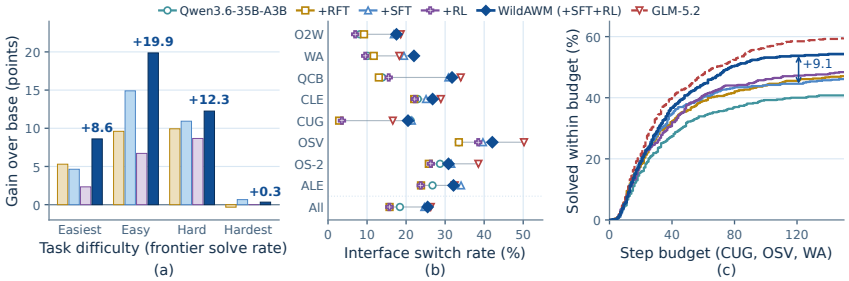

Figure 4: Behavior analysis of base model, WildAWM and its variants (+RFT, +SFT, and +RL, i.e., RL without SFT), and GLM-5.2. (a) Pass-rate gain over the base model on OOD tasks, split into four equal quartiles by how many of frontier models solve each task. (b) Fraction of consecutive tool calls that switch between interface families (e.g., GUI → CLI). (c) Tasks solved within a step budget.

5.4 Policy Behavior Analysis

Figure 4 compares OOD trajectories from the base model, WildAWM, and three variants: SFT, which fine-tunes on trajectories from GLM-5.2 (the initialization checkpoint of WildAWM), RFT (Yuan et al., 2023), which fine-tunes on the base model's own successful trajectories instead of GLM-5.2 demonstrations, and +RL, which applies the same RL directly to the base model without SFT. (1) *Gains concentrate on tasks within the teacher's reach.* We divide OOD tasks into difficulty quartiles and find that WildAWM gains most on the second and third quartiles ($+19.9$ and $+12.2$ points), where RL adds another 5.0 and 1.3 points over SFT. In contrast, +RL improves by only 6.7 and 8.7 points. On the hardest quartile, which is also challenging for the teacher, no variant shows a clear gain. (2) *Cross-interface interaction is learned from the teacher.* The interface-switch rate rises from 18.4% to 24.7% after SFT and 25.5% after RL, approaching GLM-5.2. In contrast, RFT and +RL reduce it to 15.8% and 15.7%, respectively. (3) *RL adds long-horizon capability beyond SFT.* SFT and WildAWM solve similar fractions of tasks within 40 steps (34.9% vs. 36.8%), but between 40 and 120 steps SFT gains only 9.7 points while WildAWM gains 17.0, close to the teacher. +RL also gains 15.2 points in this range, but starts lower and reaches only 47.2% within 120 steps versus 53.7% for WildAWM. Overall, teacher trajectories transfer cross-interface behavior through SFT, while RL adds long-horizon problem solving most effectively on top of this initialization.

6 Conclusion

In this paper, we propose WildAWM, a scalable pipeline for synthesizing executable worlds and tasks for general computer-use agents. In contrast to existing approaches often centered on a single application or single agent interface, our method constructs personal-computer environments spanning GUI, MCP, and CLI, and generates cross-interface tasks grounded in accumulated world states. We introduce execution-based quality gates for task solvability, difficulty, and verifier reliability. The entire synthesis and validation pipeline uses open-source models, enabling transparent, high-quality generation without relying on any proprietary frontier models. Using WildAWM, we synthesize 341 long-horizon tasks and train two Qwen models exclusively in these synthetic worlds. The resulting agents improve consistently across all eight out-of-distribution benchmarks spanning web, claw, and computer-use agents, demonstrating broad transfer beyond the synthetic environments. We believe that WildAWM offers a promising path toward open-source models generating scalable synthetic worlds, learning within them, and ultimately becoming powerful general computer-use agents.

AI USE STATEMENT

In this paper, we used LLMs described in Sec. 3.2 (Rollouts & Judgments) for synthetic environment and task generation. As with any automatically synthesized data, these artifacts are not intended to be perfect replicas of the real world and may retain residual inconsistencies or implementation bugs despite our proposed validation procedures. We also used LLMs to help implement the method and to assist in writing the paper. AI-assisted outputs including the codes and text were reviewed by the authors to ensure accuracy and validity. We inspected the experiment results (e.g., by case studying the trajectories) to verify the correctness of the generated codes. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or real personal data. The environments, personas, and task data used for training are synthetically generated and executed in isolated, resettable systems rather than real user services. This design reduces the privacy and security risks associated with training computer-use agents directly in the real world. Nevertheless, synthetic environments may contain residual inconsistencies or implementation bugs, and trained computer-use agents could potentially be misused when deployed on real systems. We therefore position WildAWM as a controlled research framework for studying and training general computer-use agents.

REPRODUCIBILITY STATEMENT

A central design goal of this work is reproducible environment and task synthesis. In contrast to several existing synthesis pipelines that depend on proprietary frontier models whose implementations and serving behavior are not publicly accessible, WildAWM uses open-source LLMs throughout environment generation, task synthesis, rollout-based validation, and verifier refinement, substantially reducing dependence on black-box model APIs. The main paper specifies the complete synthesis procedure, quality gates, verifier formulation, and RL algorithm, while the appendix provides additional implementation details, training configurations, prompts, and evaluation protocols. We will release the synthesized environments, tasks, verifiers, and the pipeline code needed to reproduce our experiments. Together, the use of publicly available models and the release of executable synthesis artifacts make it possible to inspect, rerun, and extend the full pipeline rather than relying on inaccessible proprietary components.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, volume 2025, pp. 22924–22946, 2025.

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-anything: Turn any software into an agent environment. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026. URL https://arxiv.org/abs/2604.06126.

anomalyco. OpenCode: The open source ai coding agent, 2025. URL https://github.com/anomalyco/opencode.

Anthropic. Claude Code. https://github.com/anthropics/claude-code, 2025. Software repository, accessed 2026-09-01.

Anthropic. Introducing claude opus 4.8, May 2026a. URL https://www.anthropic.com/news/claude-opus-4-8.

Anthropic. Introducing opus 5, 2026b. URL https://www.anthropic.com/news/claude-opus-5.

Yuan Cao, Dezhi Ran, Mengzhou Wu, Yuzhe Guo, Xin Chen, Ang Li, Gang Cao, Gong Zhi, Hao Yu, Linyi Li, et al. Gui-genesis: Automated synthesis of efficient environments with verifiable rewards for gui agent post-training. *arXiv preprint arXiv:2602.14093*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Mouxiang Chen, Lei Zhang, Yunlong Feng, Xuwu Wang, Wenting Zhao, Ruisheng Cao, Jiaxi Yang, Jiawei Chen, Mingze Li, Zeyao Ma, Hao Ge, Zongmeng Zhang, Zeyu Cui, Dayiheng Liu, Jingren Zhou, Jianling Sun, Junyang Lin, and Binyuan Hui. Swe-universe: Scale real-world verifiable environments to millions, 2026. URL https://arxiv.org/abs/2602.02361.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.505.

CUA-Lite Contributors. Cua-lite: An open platform for computer-use agents. https://github.com/cua-lite/cua-lite, 2026. Accessed: 2026-08-24.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026.

Weihua Du, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable end-to-end tool-use rl with synthetic codegym. In *International Conference on Learning Representations*, volume 2026, pp. 17733–17756, 2026.

Dan Friedman and Adji Bousso Dieng. The vendi score: A diversity evaluation metric for machine learning. *arXiv preprint arXiv:2210.02410*, 2022.

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

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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024.

Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. TMax: A simple recipe for terminal agents, 2026. URL https://arxiv.org/abs/2606.23321.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents, 2025. URL https://arxiv.org/abs/2504.07164.

Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents, 2025. URL https://arxiv.org/abs/2510.24563.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, et al. Recursive synthesis for long-horizon terminal tasks. *arXiv preprint arXiv:2608.05466*, 2026.

MiniMax Team. MiniMax M3: Frontier coding, 1m context, native multimodality — all in one model, 2026. URL https://www.minimax.io/blog/minimax-m3.

Nous Research. Hermes Agent. https://github.com/NousResearch/hermes-agent, 2026. Software repository, accessed 2026-09-01.

OpenAI. Codex CLI. https://github.com/openai/codex, 2025a. Software repository, accessed 2026-09-01.

OpenAI. Computer-using agent, January 2025b. URL https://openai.com/index/computer-using-agent/.

OpenAI. Introducing GPT-5.5: A new class of intelligence for real work, 2026a. URL https://openai.com/index/introducing-gpt-5-5/.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, July 2026b. URL https://openai.com/index/gpt-5-6/.

OpenAI. GPT-6 Astra: A new generation of intelligence, September 2026c. URL https://openai.com/index/gpt-6-astra/.

OpenClaw contributors. OpenClaw. https://github.com/openclaw/openclaw, 2026. Software repository, accessed 2026-09-01.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Awadallah. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents, 2025. URL https://arxiv.org/abs/2502.11357.

Baolin Peng, Wenlin Yao, Qianhui Wu, Hao Cheng, Xiao Yu, Rui Yang, Tao Ge, Alessandro Sordoni, Xingdi Yuan, Yelong Shen, et al. Orchard: An open-source agentic modeling framework. *arXiv preprint arXiv:2605.15040*, 2026.

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

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

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

Xubin Ren, Yongru Chen, and nanobot contributors. nanobot: Ultra-lightweight personal ai agent framework. https://github.com/HKUDS/nanobot, 2026. Software repository, accessed 2026-09-01.

Stuart W Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 42(1): 97–101, 2000.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Esin DURMUS, Zac Hatfield-Dodds, Scott R Johnston, Shauna M Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=tvhaxkMKAn.

Haoting Shi, Wenhao Wang, Weicheng Fang, Yaozhong Liang, Tian Jin, Pengxiang Zhao, Guangyi Liu, Siheng Chen, and Yanfeng Wang. Cua-universe: A scalable and dynamic environment for hybrid gui+cli agents, 2026. URL https://arxiv.org/abs/2609.05374.

David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1(11):6, 2025.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, silvio savarese, Zeyuan Chen, Jieyu Zhao, Ran Xu, and Caiming Xiong. Coact-1: Computer-using multi-agent system with coding actions. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 127091–127107, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/ce65173b994cf7c925c71b482ee14a8d-Paper-Conference.pdf.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö. Arık. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments, 2025. URL https://arxiv.org/abs/2501.10893.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*, 2024.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, et al. Agents' last exam. *arXiv preprint arXiv:2606.05405*, 2026.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*, 2024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

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

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, 2026a. URL https://arxiv.org/abs/2605.25624.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, et al. Opencua: Open foundations for computer-use agents. *Advances in Neural Information Processing Systems*, 38:139756–139806, 2026b.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023. URL https://arxiv.org/abs/2212.10560.

Zhaoyang Wang, Yiming Liang, Xuchao Zhang, Qianhui Wu, Siwei Han, Anson Bastos, Rujia Wang, Chetan Bansal, Baolin Peng, Jianfeng Gao, Saravan Rajmohan, and Huaxiu Yao. SynthAgent: Adapting web agents with synthetic supervision. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15730–15752, San Diego, California, United States, July 2026c. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.716. URL https://aclanthology.org/2026.acl-long.716/.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026d. URL https://openreview.net/forum?id=OvLnVkV14P.

WebHarbor Team. Webharbor: Docking real websites for evolving gui agent environments, 2026. URL https://aiming-lab.github.io/webharbor.github.io. Project website.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*, 2025.

Yifan Wu, Yiran Peng, Yiyu Chen, Jianhao Ruan, Zijie Zhuang, Cheng Yang, Jiayi Zhang, Man Chen, Yenchi Tseng, Zhaoyang Yu, et al. Autowebworld: Synthesizing infinite verifiable web environments via finite state machines. *arXiv preprint arXiv:2602.14296*, 2026.

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

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions, 2023. URL https://arxiv.org/abs/2304.12244.

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

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

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=6jZi4HSs6o.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025. URL https://arxiv.org/abs/2504.21798.

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

Rui Yang, Qianhui Wu, Zhaoyang Wang, Hanyang Chen, Ke Yang, Hao Cheng, Huaxiu Yao, Baoling 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, 2026b. URL https://arxiv.org/abs/2602.22190.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Xiao Yu, Baolin Peng, Ruize Xu, Hao Zou, Qianhui Wu, Hao Cheng, Wenlin Yao, Nikhil Singh, Zhou Yu, and Jianfeng Gao. Openforgerl: Train harness-native agents in any environment, 2026. URL https://arxiv.org/abs/2607.21557.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, et al. Osworld2. 0: Benchmarking computer use agents on long-horizon real-world tasks. *arXiv preprint arXiv:2606.29537*, 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.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Ufo: A ui-focused agent for windows os interaction, 2024. URL https://arxiv.org/abs/2402.07939.

Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Chao Du, Liqun Li, Yu Kang, Zhao Jiang, Suzhen Zheng, Rujia Wang, Jiaxu Qian, Minghua Ma, Jian-Guang Lou, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Ufo2: The desktop agentos, 2025a. URL https://arxiv.org/abs/2504.14603.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded, 2024. URL https://arxiv.org/abs/2401.01614.

Hanzhang Zhou, Panrong Tong, Xu Zhang, Quyu Kong, Chenglin Cai, Tianyu Xia, Gongjie Zhang, Jianan Zhang, Long Li, Long Chen, et al. Qwen-ui-agent technical report: Toward next-generation real-world centric foundation gui agents. *arXiv preprint arXiv:2607.28227*, 2026.

Shuyan Zhou. Webarena-infinity: Generating browser environments with verifiable tasks at scale. *shuyanzhou.com*, March 2026. URL https://webarena.dev/webarena-infinity/.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Kaijie Zhu, Yuzhou Nie, Yijiang Li, Yiming Huang, Jialian Wu, Jiang Liu, Ximeng Sun, Zhenfei Yin, Lun Wang, Zicheng Liu, et al. Termigen: High-fidelity environment and robust trajectory synthesis for terminal agents. *arXiv preprint arXiv:2602.07274*, 2026.

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

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

A Implementation Details

A.1 Evaluation Benchmarks

We evaluate across 8 out-of-distribution benchmarks spanning web, claw, and computer-use agent tasks. Considering existing works often evaluate on 2–4 benchmarks (Wang et al., 2026a; Shi et al., 2026; Ge et al., 2026), our evaluation suite is among the most comprehensive in the literature, we evaluate on a specific subset rather than the full task set of several benchmarks for evaluation efficiency. Table 6 summarizes the task count, scoring metric, and pass criterion for each benchmark. To reduce computational cost and accelerate inference, we also apply NVFP4 quantization to all open-source models used during both synthesis and evaluation. Throughout all benchmarks, we use GPT-5.5 (OpenAI, 2026a) as the judge model if the task or evaluator requires an LLM judge. All benchmarks except for Online-Mind2Web and WebArena are evaluated through Docker hosted via Orchard (Peng et al., 2026). For any infra errors (e.g., network issues, Docker issues), we allow up to 3 retries per attempt, otherwise we exclude these attempts from the scoring. For timeout errors (1 hour limit in default per task), we still count them as valid attempts and run the judgments. We describe each benchmark in detail below.

Online-Mind2Web (O2W). We use the full test split of 300 tasks from Online-Mind2Web (Xue et al., 2025), which spans over 100 live websites. Each task is evaluated by an LLM judge (GPT-5.5) using AgentTrek judge method (Xu et al., 2025) that assigns a binary label. For websites being blocked due to reCAPTCHA or other access restrictions, we exclude them from the evaluation. We report the task success rate, equivalent to pass@1.

WebArena (WA). Following previous works (Sun et al., 2024; Wang et al., 2026c), we use a subset consisting of 226 tasks from WebArena (Zhou et al., 2024), spanning across five self-hosted websites (Reddit, GitLab, shopping, shopping management, and map). Each task receives a binary score from the original programmatic evaluator. We report the pass rate here.

QwenClawBench (QCB). We use full task set of QwenClawBench v1.1 (Qwen Team & Alibaba Data, 2026), reflecting real-world OpenClaw usage patterns. The original grader computes a penalized score normalized by the maximum possible score, yielding values in $[0, 1]$. A task is considered fully passed only when the agent achieves the maximum score. We report mean score, which captures partial credit.

ClawEval (CLE). We evaluate full task set of ClawEval (Ye et al., 2026), covering service orchestration, multimodal interaction, and multi-turn professional workflows. Each task receives a continuous task score in $[0, 1]$, with a pass threshold of 0.75. To make results comparable with the baseline method (i.e., OpenForgeRL (Yu et al., 2026)), we report the pass rate, which is the fraction of tasks with score ≥ 0.75.

CUA-Gym (CUG). CUA-Gym (Wang et al., 2026a) is a synthetic dataset for training web and desktop agents. In the released dataset[4], it contains 10,910 tasks across 11 desktop/browser/OS environments and 31 web environments, and each task provides a programmatic reward function that returns a continuous score in $[0, 1]$. We randomly sample 200 tasks to form a evaluation set to test GUI capabilities of agents.

OSWorld-Verified (OSV). We use CUA-Lite (Contributors, 2026) to dockerize the OSWorld-Verified (Xie et al., 2024). We evaluate on a stratified 30% sample of 99 tasks from OSWorld (Xie et al., 2024) 330 task set, maintaining the domain distribution across 10 application categories (Chrome, GIMP, LibreOffice Calc/Impress/Writer, multi-app, OS, Thunderbird, VLC, VS Code). Each task receives a continuous reward in $[0, 1]$ from the original evaluator, which we report as the mean score.

OSWorld 2.0 (OS-2). We evaluate the full set of OSWorld 2.0 (Yuan et al., 2026), which introduces longer-horizon and more complex desktop workflows. Since they are challenging for current agents, we report the credit score in $[0, 1]$ from the original evaluator instead of the pass rate, in order to better reflect the agent's performance.

Agents' Last Exam (ALE). We evaluate the official Linux-compatible subset of 99 tasks from Agents' Last Exam (Sun et al., 2026), excluding 4 tasks that require nested Docker containers and 2

[4] https://huggingface.co/datasets/xlangai/CUA-Gym

Table 6: Evaluation benchmark details. For binary benchmarks, mean score equals pass rate (pass@1).

Benchmark	Abbr.	Tasks	Subset	Raw range	Pass criterion	Reported metric
Online-Mind2Web	O2W	300	Full set	$\{0, 1\}$	LLM judge = success	pass rate
WebArena	WA	226	Subset of 812	$\{0, 1\}$	score ≥ 1.0	pass rate
QwenClawBench	QCB	100	Full set	$[0, 1]$	score = max	mean score
ClawEval	CLE	300	Full set	$[0, 1]$	score ≥ 0.75	pass rate
CUA-Gym	CUG	200	Subset of 10,910	$[0, 1]$	reward ≥ 1.0	mean score
OSWorld-Verified	OSV	99	Subset of 330	$[0, 1]$	score ≥ 1.0	mean score
OSWorld 2.0	OS-2	108	Full set	$[0, 1]$	score ≥ 1.0	mean score
Agents' Last Exam	ALE	99	Linux subset	$[0, 1]$	score ≥ 1.0	mean score
AWMWild-val	–	33	In-domain	$\{0, 1\}$	score ≥ 1.0	pass rate

tasks that require Apptainer/GUI bundles. Each task receives a continuous score in $[0, 1]$ from the official grader.

WildAWM-val (In-domain). We randomly sample 33 tasks from our synthesized set for in-domain evaluation, running 8 rollouts per task (264 total). Each rollout produces a binary `solved` indicator. For n total rollouts with c successes, pass rates are estimated via the unbiased estimator (Chen et al., 2021) as follows:

$$\widehat{\text{pass@k}} = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}, \tag{3}$$

where $k \leq n$. When $k = 1$ this reduces to c/n, *i.e.*mean score under multiple runs. We also report a stricter metric requiring all k rollouts to succeed:

$$\text{pass\^k} = \frac{\binom{c}{k}}{\binom{n}{k}}, \tag{4}$$

which gives the probability that every sampled rollout is correct.

A.2 Evaluation Protocol

Agent Harness. All evaluations use OpenCode (anomalyco, 2025) CLI harness (v1.17.7) as the agent runtime. The agent operates autonomously with no interactive user. We use a brief prompt to instruct it to complete the task using available native, MCP, browser, shell, and file tools, choosing each interface based on the operation's required capability rather than following a fixed priority. This joint action space distinguishes WildAWM from prior work that typically equips agents with only one interface type. Upon task completion, the agent writes a marker file and stops. OpenCode harness provides built-in tools for file operations (`read`, `write`, `edit`, `glob`, `grep`) and shell execution (`bash`). Shell execution enables the agent to run arbitrary commands and scripts such as python. Several interactive tools are disabled during evaluation: `task`, `question`, `plan_enter`, `plan_exit`, `todowrite`, `webfetch`, and `websearch`.

GUI Interaction. For benchmarks involving graphical interfaces, the agent accesses GUI through two complementary mechanisms:

- **Playwright** (v0.0.78): exposes browser automation actions including navigation, element interaction (`click`, `type`, `select_option`), content extraction (`get_text`, `get_attribute`), and page management.
- **Browser-use CLI** (v0.13.6): provides headless Chromium-based web automation as a command-line tool, complementing the Playwright interface for scenarios requiring programmatic browser control.

For desktop benchmarks such as OSWorld, the agent additionally accesses a custom tool set that exposes fine-grained desktop control: `screenshot`, `accessibility_tree`, `click`, `type_text`, `scroll`, keyboard and mouse primitives (`hotkey`, `key_down`, `key_up`, `mouse_down`, `mouse_up`, `move`), and in-guest execution (`guest_bash`, `guest_python`).

Vision. Some evaluated policy models may not natively support multimodal input. To enable visual perception, we provide a vision tool that bridges the text-only policy model with visual content. The tool accepts a file path to an image and returns a text description generated by a separate vision-language model (Qwen3.6-35B-A3B (Qwen Team, 2026)). This allows the agent to interpret screenshots and images to finish GUI related operations.

Benchmark-Specific Configurations. The maximum number of agent turns is set to 256 by default, except for OS-2 and ALE where we allow 512 turns due to their longer-horizon tasks. The context window is 100K tokens with unlimited auto context compaction. We use a temperature of 1.0, top-p of 0.95, top-k of 20. The reasoning effort is set to high. Tools availability varies by benchmark to match each benchmark's intended evaluation interface:

- **Online-Mind2Web and WebArena**: Playwright, browser-use, vision, and shell/file tools.
- **QwenClawBench**: vision and shell/file tools only.
- **ClawEval**: task-specific MCP tools, vision, and shell/file tools.
- **CUA-Gym**: Playwright, browser-use, vision, and shell/file tools.
- **OSWorld v1/v2**: desktop tools, Playwright, browser-use, vision, and shell/file tools.
- **Agent's Last Exam**: vision and the ALE native agent tools set.
- **WildAWM-val**: environment-specific MCP servers (e.g., email), Playwright, browser-use, vision, and shell/file tools.

A.3 World Synthesis Details

The world synthesis pipeline constructs persona-specific, multi-day environments through a sequence of coding-agent stages, each governed by a constitution (system prompt) that constrains the agent's deliverables and quality criteria. We integrate 15 websites from WebHarbor into our day 0 world: AllRecipes, Amazon, Apple, Arxiv, BBC News, Booking, Cambridge Dictionary, Coursera, Espn, Github, Google Search, Google Flights, Google Map, HuggingFace and WolframAlpha, in order to provide visual web substrate and image content for following synthesis. All stages use GLM-5.2 (Zeng et al., 2026) with the OpenCode harness (anomalyco, 2025). We summarize the key prompts used for world synthesis and task synthesis in Tables 10 and 11, respectively. In addition, the resulting world synthesis statistics and task synthesis statistics in Tables 7 and 9, respectively.

Persona Generation. The persona creator agent receives a constitution instructing it to generate a "persona bible" for one specific, realistic person. The agent must first ground itself in the real application catalog by inspecting the 15 WebHarbor seed databases and 8 MCP application schemas to ensure the persona's digital activities reference entities that actually exist (*i.e.*, real products, courses, hotels from the seed catalogs). The deliverable includes a structured persona profile (demographics, relationship network of $\geq$3 named contacts, tools and services, multi-day story arcs with at least one image-decisive arc), web account credentials with deterministic password hashes, and a compact signature for the diversity gate. To enforce diversity, accepted personas are checked against all prior ones through a two-stage gate: (1) a deterministic area-histogram precheck that caps each life-domain slug (e.g., `grad-student`, `remote-engineer`) at 2 personas, and (2) an LLM diversity judge that rejects candidates sharing the same occupation, core tool-usage pattern, or narrative arc as any existing persona (the 2nd row in Table 10). Rejected candidates receive diagnostic feedback and may be revised up to two times before final rejection.

Table 7: World synthesis statistics. "touched" means the number of services or databases that are modified during the day of synthesis.

Scale and diversity		*Applications*	
Personas	36	Websites/MCPs	15/8
World states	640	Reusable skills	83
States per persona	17.8	Sites/MCPs per persona	5.25/5.44
Daily mean		*Final-day mean*	
Sites/MCPs touched	3.97/5.31	DB records	171.6K
DB tables/files touched	15.38/4.97	# Local files	50.1
World size growth	30.9 KiB	World state size	59.3 MiB

Day Planning. For each accepted persona, days are planned sequentially. The day planner agent receives the persona bible, prior day plans, and read access to the actual cumulative world state through day $N-1$. It outputs a markdown plan covering computer use activities, MCP service events with continuity (e.g., thread replies, calendar updates), and web GUI actions referencing real catalog entities. The plan also includes coverage pressure: a per-app and per-site usage histogram from prior days that encourages the planner to diversify interface usage across days.

Day Implementation. The day implementer is a "god-mode" coding agent that materializes a day plan into a deterministic, replayable delta. It runs inside a live, booted world instance with full access to MCP services, WebHarbor databases, and the filesystem. The key deliverable is a `delta.sh` script (with deterministic Python sub-scripts) that, replayed hermetically over the day $N-1$ state, reproduces day N identically every time. Determinism is enforced through several hard constraints: timestamps must be frozen literals (never `datetime.now()`), IDs must be derived (`MAX(id)+1`), inserts must be idempotent, and passwords must use fixed-salt hashes from the persona's `accounts.json`. When the plan requires unavailable functionality, the implementer may produce an overlay containing new MCP servers or WebHarbor code patches. Note that every world instance reuses a common application layer while maintaining isolated copies of their mutable state, including service databases, website histories, and local files.

World Verification. Each materialized day undergoes a 3-layer gate before acceptance:

1. **Replay determinism**: `delta.sh` is replayed twice from the same day $N-1$ state; the two resulting worlds must produce identical content hashes.
2. **GUI login render**: for each declared web account, the gate logs in with the persona's credentials and verifies that the account page renders correctly.
3. **Reviewer agent**: an independent coding agent running on the host (outside the implementer's container) inspects the materialized snapshot against the day plan, verifying MCP events, web GUI state, files, and determinism.

Personas and Task Distribution. Table 8 lists the 36 synthesized personas. Each persona profile lists 3 to 15 websites, 4 to 8 MCP services, and 2 to 5 multi-day story arcs, and each persona is materialized into 8 to 30 world states, 640 in total. Figure 5 shows the distribution of the 341 accepted tasks over personas and days. Every persona contributes at least two tasks, including at least one hard task, and the median is 5 tasks per persona. The six personas with the most tasks are the first six synthesized personas and account for 178 tasks. This concentration follows the order of synthesis. We first developed and tested the task synthesis pipeline on these six personas, so they were available for task synthesis over a longer period and received more synthesis attempts. The remaining 30 personas were then synthesized together with their world states and tasks, and each of them contributes 2 to 14 accepted tasks, 163 in total. Task synthesis runs at most three task slots per world state, and 246 of the 640 world states host at least one accepted task. The number of tasks per day follows the number of personas that have a world state on that day (Pearson $r = 0.90$), and 262 tasks come from days 0 to 13, on which at least 33 personas have a world state.

A.4 XAI Task Synthesis Details

Task synthesis proceeds in stages: sketch generation, diversity-driven selection, full task authoring with self-check, multi-agent rollouts, three-level judgment, and five quality gates including the counterfactual probe. Failed candidates receive diagnostic feedback and may undergo up to two fix iterations before rejection.

Sketch Generation. Given a live world at day d, the sketch agent proposes N diverse cross-interface task sketches. The prompt provides a clean workspace file index, source previews, pre-read image evidence, and the list of allowed browser write sites with their persisted state tables. Each sketch must require all three interfaces (CLI, MCP, web) and consist of at least four sub-tasks, with at least one interface used by more than one sub-task. Hard sketches must include a browser write flow (e.g., cart, order, booking, save).

Diversity Gate. Sketch selection combines a deterministic area precheck (each life-area capped at 2 accepted tasks) with an LLM diversity judge that selects the candidate adding the most topical

Table 8: The 36 synthesized personas, sorted by the number of accepted tasks. States: number of world states. Hard and Medium: number of accepted tasks of each difficulty.

Persona	Occupation	States	Hard	Medium
Jordan Rivera	Customer success manager	30	30	11
Maya Chen	Materials science postdoc	30	32	8
Alex Morrison	Coffee roastery owner	25	22	8
Sam Patel	Actuarial analyst	19	16	8
Riley Nguyen	High school teacher	30	15	7
Taylor Brooks	Backend software engineer	20	16	5
Tomas Vieira	Research software engineer	30	11	3
Mateo Silva	Food truck owner	16	10	1
Rafael Moreau	Literary translator	25	10	0
Devon Okonkwo	Athletic director	16	7	2
Aisha Bello	Developer relations engineer	30	7	1
Elena Vasquez	English exam-prep teacher	18	6	1
Rina Sharma	Clinical dietitian	16	6	1
Sofia Almeida	Astronomy PhD student	15	6	1
Camila Rojas	Music producer	15	6	1
Priya Raghavan	Sports physiotherapist	16	5	1
Hana Kobayashi	Freelance industrial designer	14	6	0
Beatriz Fonseca	Branch librarian	17	4	1
Nadia Haddad	Ward nurse	15	3	2
Ingrid Sorensen	Research software engineer	15	4	1
Finn Gallagher	Exploration geologist	14	4	1
Wei Zhang	Startup ML engineer	14	4	1
Dana Kovac	Commodities market analyst	14	5	0
Greta Lindqvist	Foreign correspondent	13	5	0
Lars Petersen	Audio-visual technician	17	4	0
Suki Tanaka	Electronics kit designer	16	4	0
Kwame Asante	Engineering undergraduate	15	4	0
Bruno Costa	Warehouse shift supervisor	15	2	2
Viktor Ilic	IT helpdesk team lead	14	4	0
Yusuf Demir	Independent travel consultant	17	3	0
Amara Diallo	Wedding planner	15	3	0
Chloe Bennett	Veterinary nurse	16	2	0
Omar Farouk	Biostatistician	15	2	0
Julian Reyes	Policy researcher	14	2	0
Idris Rahman	Retired headteacher	11	2	0
Marta Nowak	Freelance session violinist	8	1	1
Total		640	273	68

variety relative to already-accepted tasks, rejecting near-duplicates that share the same core action and artifacts (the 2nd row in Table 11).

Task Generation. The generator agent receives the chosen sketch and a detailed constitution (the first row in Table 11) defining nine validity rules for XAI tasks: all-three-interface mandatory with $\geq$4 sub-tasks, long-horizon difficulty, outcome constraints in `task.md` without execution guidance, grounding in real world entities, image-decisive web choices, secrecy discipline (solver-visible `task.md` vs. pipeline-internal `reason.md`/`meta.json`), public prompt $\leq$300 words, website write persistence, and temporal context. The agent outputs seven artifacts: `task.md` (natural first-person request), `reason.md` (rationale), `meta.json` (structured metadata), `golden.py` (perfect-solution state writer), `rubrics.md`, `verify.skill.md`, and deterministic verifier scripts composing an SDK of diff primitives. A mandatory self-check loop validates all artifacts until convergence. The `reason.md` is served as the hint used in the rollouts.

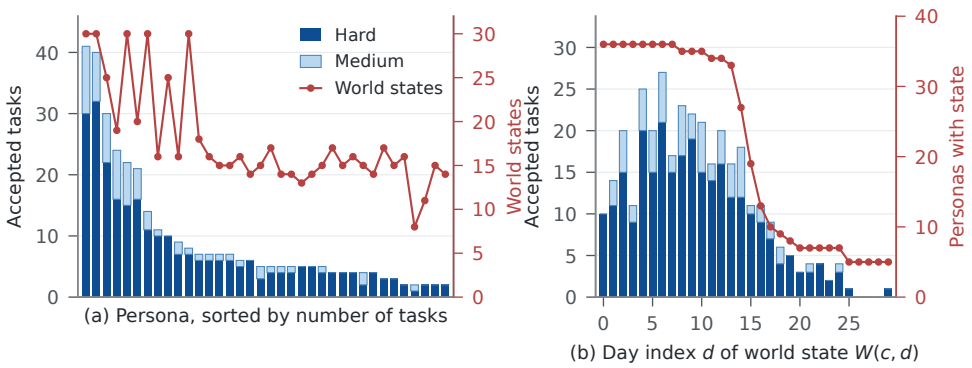

Figure 5: Distribution of the 341 accepted XAI tasks. (a) Tasks per persona, split by difficulty, with the number of world states of each persona. (b) Tasks per day index, with the number of personas that have a world state on that day.

Table 9: XAI task statistics. SR: mean success rate over synthesis rollouts from a pool of 18 agent configs; fixes: feedback-driven revisions per task at each gate. Task execution and interface usage are measured by GLM-5.2 with OpenCode. File operations include reading, editing, and creation.

Task corpus	
Tasks	341
Hard tasks	273
Medium tasks	68
Tasks per persona	9.5
Tasks per world	0.53
Interface switches / task	3.1
Success rate during Synthesis	
SR w/o hint	21.3%
SR w/ hint	45.6%

Synthesis Revisions	Mean	Median	P90
Initial-unsat. fixes	0.05	0	0
Judge-consist. fixes	1.09	1	3
Solvability fixes	1.16	1	3
Difficulty fixes	0.39	0	1
Probe fixes	0.18	0	1
Costs per task	Mean	Median	P90
Revision rounds	2.5	2	5
Generator tokens in (M)	69.8	58.3	130.9
Generator tokens out (M)	1.1	1.0	2.0

Task Execution	Mean	Median	P90
LLM calls	80	74	121
Tool calls	81	76	124
LLM in tokens (M)	4.1	3.7	6.5
LLM out tokens (K)	20.6	18.0	34.7
Interface Usage	Mean	Median	P90
GUI websites visited	1.1	1.0	2.0
MCP servers connected	2.4	2.0	3.0
Local files processed	5.0	4.0	9.0

LLM Judge. The LLM judge (GLM-5.2) receives the task, rubric, and a machine-extracted evidence digest (JSON values from deterministic verifier scripts) and produces a three-class verdict (`completed`/`failed`/`world_imperfect`) with calibrated per-class confidence scores (the 3rd row in Table 11). It judges against the rubric's intent rather than exact strings, accepting faithful solutions in the persona's own words.

Agent Judge. The heavy agent-as-a-judge runs with full permission of shell tools and vision access, inspecting BEFORE and AFTER world snapshots. Its constitution enforces strict evidence provenance: agent-authored text is admissible only as the deliverable itself, never as proof of an external fact. For visual requirements, the judge must independently call `describe_image` on ground-truth served images rather than trusting the agent's claims. It classifies every false deterministic check as either `too-strict` (brittle verifier) or `genuine-miss` (real failure) (the fourth row in Table 11).

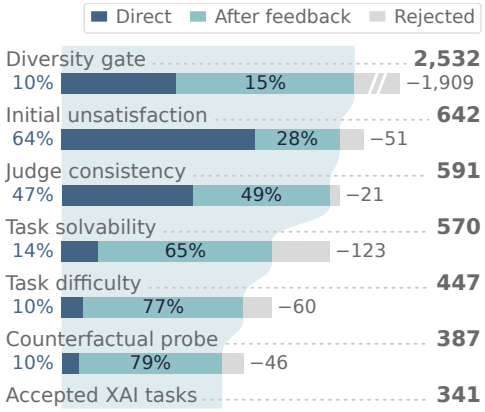

Figure 6: Progression of XAI task candidates. Direct denotes sketches that pass on the first attempt, while After feedback denotes sketches that pass after at least one feedback-driven revision at any stage of the whole synthesis, thus the funnel is monotonic.

Counterfactual Probe. The probe operates in two stages on fresh agent sessions with no generator context. First, a *blind re-judge* forms an independent verdict from the world snapshots with no stated expectation. If it disagrees with the ensemble verdict, the mismatch signals a verifier/rubric issue. Second, the probe stage asserts the opposite verdict ("an independent audit believes the correct verdict is **[opposite]**") and observes whether the judge agent defends its original verdict with cited evidence

Table 10: Key prompt excerpts for world synthesis stages (Sec. A.3).

Stage	Prompt Excerpt
Persona Creator	*You are generating a persona bible for a synthetic personal-computer world — a single, specific, realistic person whose N days of digital life will later be synthesized across three interfaces (CLI/files, MCP services, Web GUI). This persona must be self-consistent, concrete, and deliberately different from any existing persona.* *Grounding (do this FIRST, with real data — never invent the catalog): WebHarbor sites (15): inspect the seed catalogs at `webharbor_seeds/<site>/` — these SQLite DBs hold the REAL products/courses/hotels the persona will actually interact with. MCP apps (8): read `wildawm/apps/<app>/{models.py,schemas.py}` for the real schema the persona's MCP life will use. …* *persona.md schema: (1) Demographics (2) Overview (3) Relationship network ≥3 named NPCs (4) Tools & services — split into MCP services vs Web GUI sites (5) Near-term goals (6) Multi-day story arcs ≥2 ongoing arcs; mark at least one image-decisive (7) Typical daily routine (8) Web accounts.*
Persona Diversity Judge	*You are a strict diversity judge for synthetic personas. Decide whether the NEW persona is sufficiently DIFFERENT from all EXISTING personas. Two personas are too similar if they share the same occupation + life-domain, the same core tool/site usage pattern, or the same narrative arc — even with different names. Different names + same job + same sites = reject. …* *Reply with a single JSON:* `{''accept'': true|false, ''reason'': ''...''}`
Day Planner	*You are writing a day plan for one specific persona, for one specific day, that a later day_implementer will turn into deterministic data/code. Think like the persona living that day on their computer.* *Hard rule: grounded in the REAL day N-1 state. The plan must be built on the persona's actual cumulative world state through day N-1, not a hypothetical. Before planning: read the persona bible, inspect the real day N-1 state (`data/*.db`, `workspace/`, `webharbor/*.db`), read prior day plans. …* *What a day plan contains: a realistic, busy-but-not-absurd day across all three interfaces, continuing the persona's arcs and goals. Cover: CLI / files, MCP services, Web GUI, one image-decisive beat. Keep the day plausible and continuous.*
Day Implementer	*You are a god-mode agent implementing one day of a persona's world as a deterministic, replayable delta. Your deliverable is a `delta.sh` (+ deterministic sub-scripts) that, run hermetically over the day N-1 state, reproduces day N byte/logically identically every time.* *Determinism rules: timestamps are frozen literals, NEVER `datetime.now()`; IDs are derived (`MAX(id)+1`), never random; inserts are idempotent (`INSERT OR IGNORE`); passwords use literal hash from `accounts.json`, NEVER compute bcrypt with random salt at replay. …* *Deliverables: (1) `delta.sh` — hermetic bash entry (2) `*.py` sub-scripts — deterministic state writes (3) `produces.json` — machine-readable declaration of produced artifacts (4) `state_delta.md` (5) optional `overlay/` — new MCP servers or WebHarbor patches.*
World Reviewer	*You are an independent reviewer verifying that a synthesized day-N world actually matches what its plan said would happen. You run on the host (outside the implementer's container).* *Verify, with real evidence — judge INTENT, not exact values. The plan is a GUIDE, not a byte-spec. The implementer is a different agent and will choose its own concrete values (message wording, is_read flags, exact timestamps) — that is EXPECTED + fine. Judge whether the day's world is real, self-consistent, on-persona, renderable, and deterministic.* *Accept unless there is a GENUINE DEFECT: a declared artifact is missing; a web account row exists but won't render; a non-frozen timestamp; an impossible value (broken FK, negative total). Reasonable creative variation from the plan is ACCEPTABLE.*

(pass), caves by editing verifier scripts under pressure (fail—the edit is detected via file diffs and reverted), or produces evidence that the judge was wrong, which will be routed to review (the fifth row in Table 11).

Refinement Loop. Throughout the synthesis pipeline, each agent is allowed up to two fix iterations on rejected candidates. The generator agent receives diagnostic feedback from the judge or gate results and edits all artifacts until the task passes the gates. For example, the judge agent receives diagnostic feedback from the counterfactual probe and edits its own verdict or evidence until it passes the probe. Every refinement iteration will make the task candidate re-run all the gates that may be affected by the edits.

A.5 Training Details

We train each Qwen model in two stages: SFT on GLM-5.2 trajectories collected on the 100-task SFT split, followed by RL on the 208-task RL split.

Table 11: Key prompt excerpts for XAI task synthesis stages (Sec. A.4).

Stage	Prompt Excerpt
Task Generator	*You design ONE high-quality **cross-agent-interface (XAI)** task for the LIVE world you are inside, then a verifier that can grade it. You are a privileged designer (read-mostly): inspect the world with the same interfaces an agent has — CLI, MCP apps, and the browser. You do NOT solve the task.* *What makes a task valid: (1) All-three-interface mandatory — completing it must require CLI, MCP-app, and web-GUI with real switches. Hop shape: interface_flow walks ≥4 legs with at least one re-crossing. (2) Long-horizon at the assigned difficulty. (3) Grounded in THIS world — reference REAL entities verified by inspection. (4) Web is required and image-decisive — the decisive choice hinges on a visual attribute only knowable from the product/page image. (5) Secrecy discipline — task.md is PUBLIC (solvers see it); reason.md and meta.json are PIPELINE-INTERNAL (judges only). (6) Public prompt ≤300 words. …* *Deliverables: `task.md`, `reason.md`, `meta.json`, `golden.py`, `verifier/rubrics.md`, `verifier/verify.skill.md`, `verifier/scripts/*.py`. MANDATORY before you finish: SELF-CHECK until ALL GREEN.*
Task Diversity Gate	*Pick the most DIVERSE task sketch (or reject all). Below are the titles of already-accepted tasks for this persona, then several NEW sketch candidates. A candidate is a NEAR-DUPLICATE if it shares the same core action + artifacts/entities as an accepted task (same purchase, same trip leg, same report …) — different wording does not make it distinct.* *Choose the candidate that adds the MOST topical variety. Output JSON: {``choice'': <0-based index or -1>, ``why'': ``...'', ``rejected'': [...]}*
LLM Judge	*Decide whether this task was COMPLETED — judge ONLY the evidence below. You are a strict, fair grader for an AI-agent task in a simulated personal world. You cannot inspect the world; judge ONLY the machine-extracted evidence digest (real values pulled from the world's files/databases by deterministic scripts).* *Rules: Judge against the rubric's INTENT — a faithful solution in the persona's own words or a service's real storage format counts. Do NOT demand exact strings, exact timestamps, or fields the agent's tools stamp automatically. …* *End with: `LLM_VERDICT: completed | failed | world_imperfect`; `CONFIDENCE: completed=<0-100> failed=<0-100> world_imperfect=<0-100>`*
Agent Judge	*You are an impartial, rigorous judge deciding whether an agent COMPLETED a task in a simulated personal world. You inspect ONLY the BEFORE and AFTER world snapshot directories, with read-only shell tools plus `describe_image` for image files. Never modify the snapshots, but you are free to write new files or scripts in your own workspace to help you analyze the evidence.* *Evidence provenance — NEVER trust the agent's own words as proof. Agent-authored text is admissible ONLY as the deliverable itself — never as proof of an external fact. A note claiming the agent selected a particular web item does NOT prove which item was selected. Cross-check the independent source: query the relevant `web/<site>/instance/<site>.db`. …* *For every false deterministic check, classify it: `RULECHECK <name>: too-strict` (the world satisfies the intent; the check is brittle) or `RULECHECK <name>: genuine-miss` (the agent really did not do this part).*
Counterfactual Probe	*Stage 1 (Blind): Independent verdict — judge ONLY the world snapshots. Form your OWN verdict; no expectation is given. … End with: `XAI_JUDGE: completed | failed | world_imperfect`.* *Stage 2 (Gaslight): Re-examination requested. An INDEPENDENT audit has reviewed the solved world and believes the correct verdict is **[opposite]**, which contradicts the automated verdict. Investigate honestly. You MAY edit the verifier scripts if and only if you conclude they are genuinely wrong. Three conclusions: (1) DEFEND it with cited rows/files; (2) Fix the VERIFIER SCRIPTS if wrong; (3) REFUTE-JUDGE if the verifier is right but a judge could be misled. End with: `PROBE: defend | cave | refute_judge`.*

SFT Training. Our pilot experiment results show that XAI tasks are too difficult for the base Qwen models to provide GRPO (Shao et al., 2024) learning signals from the start: Qwen3.6-35B-A3B reaches 5.7 pass@1 and 18.4 pass@4, and Qwen3.5-4B reaches 0.38 pass@1 and 1.5 pass@4. Most rollout groups would therefore receive identical rewards and waste rollout compute without producing gradients. We thus first perform supervised fine-tuning (SFT) to warm up the models on trajectories from GLM-5.2 (Zeng et al., 2026). For each of the 100 SFT tasks, we collect 4 rollouts and keep only the rollouts that solve the task ($V = 1$), with at most 1 successful trajectories per task to balance the data across tasks. Following Appendix A.6, each trajectory is split into causal segments at compaction boundaries, and segments longer than 102,400 tokens are removed. The GLM-5.2 trajectories are converted to the Qwen template, and the loss is computed only on assistant tokens. Both models are trained with Adam (β_1=0.9, β_2=0.98), a peak learning rate of 5×10^{-6} with 10% warmup and cosine decay to 5×10^{-7}, weight decay 0.01, and global batch size 8 for three epochs.

Table 12: Hyperparameters for RL training.

Optimization		Rollout	
Optimizer	Adam (β_1=0.9, β_2=0.98)	Training tasks	208
Learning rate	1×10^{-6} (constant)	Tasks per step	8
Weight decay	0.1	Group size G	4
Gradient clipping	1.0	Rollout multiplier (init / max / EMA)	1 / 3 / 0.8
Clip range (low / high)	0.2 / 0.28	Extra rollout chunk	8
KL coefficient	0	Temperature / top-p	1.0 / 0.95
Entropy coefficient	0.001	Reasoning effort	high
Partial-credit weight	0.1	Max tokens per turn	8,192
Format-failure reward	-1	Context window	102,400
Max RL steps	64	Max turns	256
Updates per RL step	1	Time limit per rollout	30 min

RL Training RL starts from the SFT checkpoint and optimizes the reward in Eq. 1 with the advantage in Eq. 2 on the 208 RL tasks. Table 12 lists important hyperparameters, which are shared by both models except that Qwen3.5-4B uses 8 rollouts per task to improve sample efficiency.

Rollouts. Each rollout runs in a disposable sandbox initialized from the task's initial world state W_0 with the help of Orchard (Peng et al., 2026) and uses the same OpenCode harness, system prompt, and tools as WildAWM evaluation (Appendix A.2). We freeze the MoE router and vision encoder, and train all other parameters. When a rollout finishes or reaches the turn or time limit, the synthesized verifier scores its final world state. Rollouts that fail because of sandbox or harness errors are retried in a new sandbox up to three times, otherwise they are excluded from the pool. For timeout rollouts, we still apply the verifier to the final world state and assign a valid reward.

Format Failures. Each policy turn is validated before its action is passed to the harness. A turn is a format failure ($e_j = 1$ in Eq. 1) if the generation reaches the per-turn token limit, the reasoning block is malformed (an extra opening tag, or a missing, repeated, or misplaced closing tag), a tool call cannot be parsed or names an unknown tool, or the output is empty. A format failure terminates the rollout immediately, and only the failing turn receives gradient. A generation truncated because the remaining context budget is exhausted is not counted as a format failure and receives no gradient.

Adaptive Rollouts. Each task τ keeps a rollout multiplier $\kappa_\tau \in [1, \kappa_{\max}]$ across RL steps, initialized to $\kappa_\tau = 1$ with $\kappa_{\max} = 3$. When τ is sampled in a step, we launch its G base rollouts together with extra rollouts in chunks of $C = 8$, for a total of

$$n_\tau = \min\left\{\kappa_{\max} G,\ G + C\left(\lfloor \kappa_\tau + 1/2 \rfloor - 1\right)\right\} \tag{5}$$

rollouts. Each time a chunk completes, we examine the pool of K_τ rollouts from all completed chunks. The task is mastered if all G base rollouts succeed, and ready if the pool contains at least two different rewards or consists only of format failures. While the task is neither, further chunks are launched until $\kappa_{\max} G$ rollouts have been launched. If the task is still neither when they complete, it is marked as exhausted. Unfinished rollouts of a mastered or ready task are canceled. For a ready task with mixed rewards, we sort the pool by reward, split it into a lower and an upper half, and draw $G/2$ rollouts from each half uniformly at random, redrawing until the selected group $\mathcal{G}_\tau$ contains two different rewards; for a pool of format failures, $\mathcal{G}_\tau$ consists of G rollouts drawn at random. The selected rollouts keep the advantages computed over the full pool in Eq. 2. Mastered and exhausted tasks have identical rewards within the group and are discarded from the step. After the optimization step, the multiplier of each sampled task is updated as

$$\kappa_\tau \leftarrow \begin{cases} 1, & \tau \text{ mastered}, \\ (1-\lambda)\,\kappa_\tau + \lambda \left\lceil K_\tau / G \right\rceil, & \tau \text{ ready}, \\ \kappa_{\max}, & \tau \text{ exhausted}, \end{cases} \tag{6}$$

with $\lambda = 0.8$, and the updated value determines n_τ the next time τ is sampled. For example, with $G = 4$, a task sampled for the first time launches $n_\tau = 4$ rollouts. If all four rewards are 0, and the pool is extended by a chunk of 8 rollouts. If one of these 8 rollouts solves the task ($r_j = 1$) and another passes half of the verifier checklist ($r_j = 0.05$), the task becomes ready with $K_\tau = 12$, $\mathcal{G}_\tau$ takes two rollouts from each half of the 12 sorted rewards, and κ_τ becomes $0.2 \cdot 1 + 0.8 \cdot 3 = 2.6$.

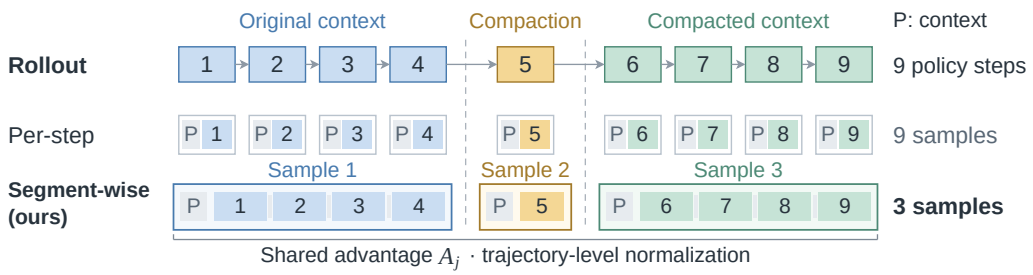

Figure 7: Segment-wise training across compaction. Colors denote maximal prefix-consistent segments, and step 5 performs compaction. Gray blocks indicate masked input context. Segment samples share the trajectory advantage A_j and trajectory-level loss normalization. A 9-step rollout with one compaction event at step 5 yields 9 independent samples under per-step training but only 3 segment samples under our approach.

The next time the task is sampled, Eq. 5 launches $n_\tau = 12$ rollouts from the start. If its 4 base rollouts complete first and already receive different rewards, the task becomes ready with $K_\tau = 4$, the remaining 8 rollouts are canceled, and κ_τ becomes $0.2 \cdot 2.6 + 0.8 \cdot 1 = 1.32$, so the following visit launches only the 4 base rollouts.

Policy Loss. Let $\mathcal{B}$ denote the ready tasks of a step, $o_{t,k}$ be the k-th token generated at step $t \in S_m$ of rollout j, $\pi_\theta^{t,k} = \pi_\theta(\cdot \mid p_m, c_m^{<t}, o_{t,<k})$ the policy distribution over it, and

$$\rho_{j,t,k}(\theta) = \frac{\pi_\theta^{t,k}(o_{t,k})}{\pi_{\theta_{\text{old}}}^{t,k}(o_{t,k})} \tag{7}$$

its importance ratio with respect to the policy $\pi_{\theta_{\text{old}}}$ that generated the rollouts. We minimize the GRPO objective

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \frac{1}{|\mathcal{G}_\tau|} \sum_{j \in \mathcal{G}_\tau} \frac{1}{L_j} \sum_{m=1}^{M_j} \sum_{t \in S_m} \sum_{k=1}^{|o_t|} \ell_{j,t,k}(\theta), \tag{8}$$

$$\ell_{j,t,k}(\theta) = -\min\left(\rho_{j,t,k} A_j,\ \text{clip}(\rho_{j,t,k}, 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}})\, A_j\right) - \beta\, \mathcal{H}\left(\pi_\theta^{t,k}\right), \tag{9}$$

where A_j is the trajectory advantage (Eq. 2), $\varepsilon_{\text{low}} = 0.2$ and $\varepsilon_{\text{high}} = 0.28$ are asymmetric clipping ranges (Yu et al., 2025), $\mathcal{H}$ is the token-level entropy with coefficient $\beta = 0.001$, and no KL penalty is applied. L_j is the number of trainable tokens in rollout j. For a format-failure rollout, both the sum and L_j include only the failing turn. The factors $1/L_j$, $1/|\mathcal{G}_\tau|$, and $1/|\mathcal{B}|$ weight rollouts equally within each group and groups equally within each step.

A.6 Segment-wise Training across Compaction

During inference, the OpenCode harness applies auto context compaction when the accumulated context approaches the window limit: earlier turns are summarized into a compressed representation, and the agent continues with a fresh prefix. This enables arbitrarily long rollouts within a fixed context budget but introduces a structural challenge for training: the trajectory is no longer a single causal sequence.

Consider rollout j of task τ, producing T_j agent steps. Let o_t denote the tokens generated by the policy π_θ at step t. Between consecutive steps, the harness appends the environment's response (e.g., tool outputs) to the context, while when the context exceeds a threshold, compaction replaces it with a summary. We define a *segment* as a maximal contiguous subsequence of steps whose contexts form a causal chain, *i.e.*, no compaction occurs between them. Compaction events partition rollout j into M_j segments $S_1, \ldots, S_{M_j}$, each with a shared base prefix p_m, as shown in Figure 7.

A standard per-step decomposition treats each step as an independent sample with its own full harness context x_t that is common in GUI agent training:

$$\mathcal{L}_j^{\text{step}}(\theta) = -\frac{A_j}{L_j} \sum_{t=1}^{T_j} \log \pi_\theta(o_t \mid x_t), \tag{10}$$

Table 13: Key behavior statistics. OOD: 8 benchmarks pooled; net tasks: fail-to-pass minus pass-to-fail tasks relative to Base (QwenClawBench excluded). WildAWM-val: 33 tasks × 8 rollouts.

	Base	GLM-5.2	+RFT	+SFT	+SFT+RL
OOD benchmarks					
Pass rate on common tasks (%)	38.7	54.5	44.6	46.2	48.4
Net tasks gained over Base	–	+207	+73	+93	+129
Interfaces used per episode	2.40	2.65	2.36	2.61	2.68
Interface switch rate (%)	18.4	26.3	15.8	24.7	25.5
GUI tools (% of tool calls)	16.3	13.6	17.2	4.3	1.6
Turn with ≥8K output tokens (%)	12.8	7.4	11.7	24.3	9.1
WildAWM-val (in-domain)					
pass@1 (%)	5.7	37.9	10.3	37.3	36.0
Interface switch rate (%)	26.1	34.1	25.8	37.0	35.7

where $L_j = \sum_{t=1}^{T_j} |o_t|$ is the total generated token count across the trajectory and A_j is the trajectory advantage (Eq. 2). This requires T_j forward passes with largely overlapping prefixes and discards the autoregressive dependency between consecutive steps sharing the same context window. Our segment-wise formulation groups steps by their shared prefix and preserves the within-segment causal structure:

$$\mathcal{L}_j^{\text{seg}}(\theta) = -\frac{A_j}{L_j} \sum_{m=1}^{M_j} \sum_{t \in S_m} \log \pi_\theta\big(o_t \,\big|\, p_m,\, c_m^{<t}\big)\,, \tag{11}$$

where $c_m^{<t}$ denotes the in-segment history preceding step t: the interleaved sequence of prior policy outputs and environment responses from all earlier steps within segment S_m. This design offers three benefits: (1) **Efficiency.** The number of forward passes reduces from T_j to M_j ($M_j \ll T_j$ in practice), avoiding redundant prefix encoding. (2) **Training–inference alignment.** Within each segment, steps condition on the model's own prior outputs, matching the autoregressive generation process at inference time. (3) **Balanced normalization.** Trajectory-level normalization by L_j ensures that rollouts with frequent compactions (many short segments) receive the same total gradient weight as those with infrequent compactions (few long segments).

B Policy Behavior Study: Details

Cross-interface interaction is inherited from the teacher. After SFT, the agent uses more interfaces per episode (2.40 for the base model, 2.61 after SFT, 2.68 after RL) and rarely drives the GUI actions (16.3%, 4.3%, and 1.6% of tool calls), whereas RFT keeps the base model's habits (2.36 and 17.2%; Table 13). The same holds in-domain: on WildAWM-val, the switch rate is 26.1% for the base model and 25.8% for RFT, but 37.0% for SFT, 35.7% for WildAWM, and 34.1% for GLM-5.2.

The benefit of RL lies in transfer, not in-domain fit. On WildAWM-val, SFT already matches the teacher (pass@1 37.3% vs. 37.9%) and RL does not improve on it (36.0%), yet RL raises the net OOD gain over the base model from +93 to +129 tasks (+73 for RFT). RL also removes the long turns that SFT produces but the teacher does not: a turn with at least 8K output tokens occurs in 24.3% of SFT episodes, 9.1% after RL, and 7.4% for GLM-5.2.

Where the step-budget gain arises. Figure 4c pools CUG, OSV, and WA, where RL adds most over SFT (72 vs. 30 net tasks over the base model). On each of them, WildAWM gains more than SFT between 40 and 120 steps (CUG: 18.0 vs. 7.5; OSV: 16.2 vs. 8.1; WA: 16.4 vs. 12.4 points). Counting timeouts as unsolved is conservative for WildAWM: 5.5% of its episodes on these benchmarks time out before step 120.

What does not improve. The longest tasks remain out of reach: the hardest quartile of Figure 4a consists mostly of OS-2 and ALE tasks, of which the base model and its four variants solve 4–7 and the strongest reference, GPT-6-Astra, 18. In future work, given sufficient compute resources, we

plan to investigate whether larger models can better exploit our synthesized data and achieve stronger performance on these tasks after training.

C CASE STUDY

This section first presents the persona and story of one synthesized world and the skills used by the generator in world and task synthesis, and then two accepted XAI tasks from the synthesized corpus. Table 19 shows the first task together with the evidence collected at each quality gate (Sec. 3.2). Table 20 and 21 show an agent trajectory on the second task and the corresponding verifier for it.

Persona and Story. Table 14 shows the persona of the world on which the first task case study is synthesized. The persona profile specifies a freelance audio-visual technician, six named contacts, the websites and MCP services he uses, and three story arcs, one of which requires choosing products on Amazon by their images. The day planner expands these arcs into day plans, which are materialized into 17 world states. Each plan continues the open threads of the previous world state, such as the return request filed on day 0 and closed within its deadline on day 4, and the sports-bar commissioning that is prepared from day 1 and carried out on day 16. The applications used by the persona also grow over the days, as email, Slack, and the library service are added on days 1, 4, and 7. Four tasks are accepted on the world states of days 5, 7, and 9. The task of Table 19 reads the owner's approval message written on day 1 together with the inventory, bill of materials, and specification sheet present since day 0, so solving it depends on state accumulated over earlier days.

Synthesis Skills. In world synthesis, the generator matches the activities of each day plan against a shared library of event creation skills (see Sec. 3.1). The library contains 83 skills in total, and Table 15 lists 20 of them. Each of these skill files consists of a description, a parameter table, one scenario per involved application with the database records or local files to produce, consistency checks against the existing state, preconditions, and possible follow-up events. Table 16 further shows the content of `book_flight` skill. Its five scenarios write a booking on the Google Flights website, a confirmation email, a direct-message exchange, calendar events, and a local notes file, and the same confirmation number appears in all five applications. Its consistency checks read the persona state and the existing records before insertion, so that message order, calendar entries, and existing files remain consistent with earlier days.

In task synthesis, the generator's evolving skills are short notes that the generator model, derives from synthesis events (Sec. 3.2). An event is recorded when a candidate is returned for deepening after being judged too easy, for easing after no rollout succeeds, or for re-authoring after a successful rollout skips a required interface, and when the candidate reaches its final verdict. For each event, the generator reads the gate verdict, the rollout statistics, the feedback, and the existing notes, and returns at most two updates or no update. Notes are limited to 500 characters and 40 files to avoid overfitting to a specific scenario or task type. The generator is instructed to edit existing notes before creating new ones. Across the task synthesis, we have 802 revisions of the 40 final skills. Table 17 shows 20 of these skills by task difficulty, interface coverage, write persistence, and verifier reliability. Table 18 further traces `hard-multi-interface-recipe` that is triggered by 40 tasks. This skill evolves from a simple recipe task to a complex multi-interface task, which can benefit the generator in future task synthesis.

Task Structure and Quality Gates. Table 19 shows the user input task, the sub-tasks recorded in the task document, and the verifier script that checks each sub-task. The task consists of five sub-tasks in the order MCP→CLI→GUI→CLI→MCP. After the purchase in the GUI, the fourth and fifth sub-tasks return to CLI and MCP and use the selected product, so intermediate results must be carried across interfaces in both directions. Each of the four sub-tasks that change the world state is checked by one verifier script. The bottom block of Table 19 reports the quality-gate evidence. The task passed all five gates in the first iteration without revision. The three levels of judgment agree on all six rollouts, all four scripts fail on the initial state W_0, one of the three hinted rollouts completes the task, and none of the three unhinted rollouts does, so the task is classified as hard. Under the counterfactual probe, the agent judge keeps its verdict and does not modify any verifier script.

Agent Trajectory. Table 20 shows the second task and six of the 65 tool calls in a rollout that completes it. The task does not name any interface, but its inputs and outputs are distributed across interfaces: the invoice is a PDF file in the local workspace, the purchase requires signing in to the

Table 14: Persona and story, condensed from the persona file and the day plans. The task of Table 19 is synthesized on the world state of day 7.

Persona	
Profile	Lars Petersen, 41, freelance audio-visual technician and sole proprietor of an AV business in Minneapolis. He installs and maintains display walls, audio, and projection for a sports bar, a church, and a community theatre.
Contacts	Marco Delgado (sports-bar owner), Pastor Tom Hauge (church client), Dee Nguyen (distributor account manager), Brendan Olsrud (install assistant), Ingrid and Søren Petersen (wife and son).
Applications	Websites: ESPN, Amazon, Google Maps. MCP services: Support Tickets, Notifications, Direct Messages, Calendar.
Story arcs	(1) Choose four 65-inch displays and rack surge protectors on Amazon by their product images for the sports-bar display wall. (2) Schedule commissioning around the televised games of April 11 and 12 using ESPN. (3) Plan the driving route between suppliers and client sites on Google Maps.
Day plans (17 world states, 2024-03-25 to 2024-04-10)	
Day 0	Initial state. The return request for a failed surge protector at the church is filed with a deadline on day 4.
Day 1	Mount brackets and pull cables at the sports bar. After the owner approves by direct message, order the displays, streaming sticks, and surge protectors on Amazon.
Days 2–4	Build the rack, receive and mount the displays, install the replacement surge protector at the church, and close the return request within its deadline.
Days 5–6	Coach the son's hockey game, check the broadcast schedule on ESPN, and run the Sunday service check at the church.
Day 7	Pick up a replacement projector from the distributor and prepare the sports-bar commissioning with the assistant.
Days 8–15	Install the replacement projector at the theatre and close its support ticket (day 11), run the weekly church check (day 13), and prepare the commissioning (days 14–15).
Day 16	Commission the sports-bar display wall before the televised games of April 11 and 12.
State growth from day 0 to day 16	
MCP services	From 4 to 7 services, with email added on day 1, Slack on day 4, and Library on day 7. MCP database rows grow from 99 to 344.
Websites	Accounts on ESPN, Amazon, Google Maps, and Google Search on day 0, AllRecipes from day 1, and WolframAlpha from day 3. Amazon orders grow from 0 to 2, ESPN favorites from 3 to 10, and Google Maps timeline entries from 2 to 28.
Accepted tasks (4)	
Day 5	Rank the open evenings of the coming week, save a dinner recipe chosen by its photo on AllRecipes, add a calendar block, and message his wife.
Day 7	Two tasks: reorder the rack surge protector that is at its reorder level (`d7-h1`), and the task of Table 19 (`d7-h2`).
Day 9	Score in-stock surge protectors from the return log, post a review on the product identified by its image, and reply to the distributor.

Amazon website supported by WebHarbor (WebHarbor Team, 2026), and the recipient is identified by a direct-message thread. The agent first resolves the recipient through the direct-message service (step 2) and extracts the prior order from the invoice with a Python script (step 9). It then searches the storefront, takes screenshots of candidate product pages, and queries the vision tool to check the product attributes specified in the request (step 45), as the listing text does not state them. After adding the selected speaker to the cart (step 59), the agent sends the product name and cart price to the recipient (step 62) and writes the selection back into the shortlist file (step 64). These two steps use information obtained in the browser and can therefore only be completed after the GUI sub-task.

Verifier. Table 21 shows the synthesized verifier for the second task. It consists of three scripts, one for each outcome requested by the task, and a rollout succeeds only if all scripts pass, *i.e.*, $V = \prod_i v_i$

Table 15: World synthesis skill examples and the applications involved in each skill. Every listed skill also writes local files through the CLI. DM means Direct Messages.

Skill	GUI	MCP
GUI, MCP, and CLI		
`japan_trip_research`	Google Flights, Booking	Notifications, DM, Calendar
`book_flight`	Google Flights	Email, DM, Calendar
`research_hotels`	Booking	DM, Notifications
`japan_hotel_booking`	Booking	DM, Email
`amazon_purchase`	Amazon	Email, Calendar
`coursera_study_session`	Coursera	Calendar, DM
`attend_sports_event`	ESPN	Calendar, DM
`meal_prep`	AllRecipes	DM
MCP and CLI		
`attend_lab_session`	–	Calendar, Slack, Canvas LMS
`prepare_lab_submission`	–	Canvas LMS, Slack
`debug_ci_failure`	–	Slack, Email, Support Tickets
`on_call_incident`	–	Email, Slack, DM
`code_review`	–	Slack, Email
`configure_feature_flag`	–	Slack, Calendar, Email
`respond_to_vendor_email`	–	Email, Support Tickets
`await_event_registration`	–	Calendar, Email, Slack
`payroll_approval`	–	Email, Calendar, DM, Notifications
`schedule_equipment_service`	–	Calendar, Email, DM
`complete_permit_renewal`	–	Email, Calendar, DM
`reply_to_customer_review`	–	Email, Calendar

(Sec. 3.2). Each check compares the final state with the initial state through primitives shared by all synthesized verifiers, such as `web_new_rows`, which returns only the rows added during the rollout. A state that already satisfies the request in W_0 therefore does not pass the check, which is the property tested by the initial unsatisfaction gate. The checks identify their targets by natural keys, namely the product slug and the participant name of the direct-message thread. The last block compares two rollouts that add different speakers to the cart. Only the first rollout adds a row for the target product, so `check_cart.py` passes for this rollout and fails for the second one, whose direct-message check also fails ($V = 0$).

Table 16: World synthesis skill `book_flight`, condensed from the skill file. Each scenario specifies the state changes in one application.

Section	Content
Description	The user books a flight on Google Flights, receives the airline confirmation, shares it with the travel companion by direct message, and records the trip in the calendar and a notes file.
Parameters	21 parameters, *e.g.*, route AUS–NRT, carrier JAL, fare 1,089 USD, seat 34A, confirmation number, message thread 2, and notes path.
GUI: Google Flights	Read the app schema, insert a booking row with status `confirmed`, and verify it by the confirmation number.
MCP: Email	Insert the airline confirmation thread and message into the personal inbox with the Inbox, Updates, and Japan-Trip labels.
MCP: Direct Messages	Read the latest messages in thread 2, insert the user's message and the companion's reply, and update the thread timestamp.
MCP: Calendar	Check for existing trip events, then insert events for the outbound flight, the trip, and the return flight into calendar 1.
CLI: Files	Check the notes directory, then create a notes file of 60–100 lines with flight details, trip plan, to-do list, and budget.
Consistency checks	Read identities from the persona state file, insert messages in chronological order without duplicates, update existing calendar placeholders, look up the label identifier, keep existing notes files, and place the companion's reply 30–90 minutes after the user's message.
Preconditions	The trip storyline is active, and the Google Flights account (created if missing), message thread 2, email label 12, and calendar 1 exist.
Follow-ups	Browse and book hotels, share an itinerary document, check email, block out-of-office time, and order travel gear.

Table 17: Evolving task synthesis skills, grouped by the task property they address. Lessons are paraphrased from the skill text. Revisions: number of revisions made during the synthesis.

Skill and lesson	Revisions
Task difficulty	
`image-pick-decisive-cue`: Naming the dish, the visual cue, or a category with a color reduces the image choice to text search.	88
`hard-multi-interface-recipe`: Two strong steps make a task hard, and a CLI computation counts only if later steps depend on it.	43
`image-only-sourcing-not-leak-protection`: Criteria that match search filters or indexed text identify the target without image inspection.	35
`criteria-specified-parse-is-lookup`: Criteria that uniquely identify the target turn parsing into a lookup.	13
`shortcut-resistance-not-gate-count`: A multi-step CLI computation counts as one step, and naming its output removes it.	10
`answer-key-saturation-strip-before-volume`: Naming every resolved value, or all candidates except one, removes the need for inspection.	7
`difficulty-calibration-intrinsic`: Difficulty is calibrated on unhinted rollouts and raised through more required work.	5
Interface coverage	
`multi-interface-indispensable`: An MCP message that restates fabricable values can be skipped unless it depends on live values absent from the CLI steps.	79
`web-state-restated-post-bypass`: A message that restates website values can be fabricated without visiting the website unless a visual decision requires the visit.	55
`interface-flow-answer-key-leak`: An MCP service that only provides context and receives a restating message can be skipped.	39
`answer-key-leak-cascades-interface-bypass`: A CLI step that names the target lets agents skip the website and MCP steps.	27
`chained-decisive-step-handoff`: Verifiers check the live website row and message, since checking a CLI copy lets agents skip both.	24
`indispensability-not-hardness`: Requiring an interface keeps it in the trajectory without making the task hard.	14
Write persistence	
`browser-write-surface-persistent`: A website write that shows success without a new database row makes the task unsolvable.	47
`answer-keyed-zero-is-infrastructure`: Zero completions at both hint tiers are first checked for a non-persistent write.	31
`web-save-preexisting-list-false-positive`: Saving to a list that already exists lets a stale row pass the check.	16
Verifier reliability	
`verifier-communication-mention-tokens`: Message checks require the concrete values requested by the task, such as amounts and dates.	14
`verifier-dm-thread-lookup`: Message threads are resolved by topic, since one participant can appear in several threads.	3
`verifier-evidence-machine-checkable`: Checks target new or modified state and identify items by database keys.	2
`verifier-script-crash-is-code-bug`: A verifier crash on the initial or reference-solution state is fixed in the script.	2

Table 18: Revisions of the evolving task synthesis skill `hard-multi-interface-recipe`, listed in order with the gate outcome that triggered each revision. Some intermediate revisions are omitted for brevity. The skill text is paraphrased for human readability.

Revision	Trigger	Change
1	Accepted as hard	Place the decisive difficulty in a computational CLI step, tie each value passed between interfaces to an upstream record, and make every interface change the world state.
2	Accepted as hard	A visual choice identified by at least two cues unique to the target is an alternative source of difficulty.
3	Returned as too easy	A parse-and-rank CLI step combined with a visual choice that reduces to reading labels remains too easy.
6	Accepted as hard	With a single visual choice, the computational step must integrate values from several interfaces in a flow that returns to an earlier interface.
11	Accepted as hard	A non-CLI read that matches the ranked result against another person's choice is a third source of difficulty.
14	Returned as too easy	Naming the output of any such step in the task removes its difficulty, so the task states criteria and leaves resolved values unnamed.
19	Accepted as hard	Two strong steps, a multi-source computation and a visual choice by art style without a text equivalent, make a task hard.
32	Returned as too easy	A computational CLI step counts only if a later step depends on its result, and restating the result in a later file does not count.
43	Returned as too easy	A value computed for a message counts only if the task gives neither the formula nor the result.
Final	After revision 43	A task is hard when it contains two strong steps, each being a computational CLI step, a visual choice unique to the target, a non-CLI read, or an inference. A computational step counts only if a later step depends on its result. A computation fed by live MCP or website reads yields a shortlist that agents cannot produce without those interfaces, and a visual decision within the shortlist is the second strong step. A final message that carries values from at least two independent sources requires all three interfaces. A non-persistent website save lets agents complete the task with the CLI alone.

Table 19: Case study of XAI task. Only the shaded block is shown to the solver, and the remaining blocks are internal to the pipeline. The interface flow MCP $\rightarrow$ CLI $\rightarrow$ GUI $\rightarrow$ CLI $\rightarrow$ MCP consists of five sub-tasks and returns to both CLI and MCP after the GUI write, so the task cannot be completed within a single interface. Each sub-task that changes the world state is checked by one deterministic verifier script, and task success requires all verifier scripts to pass.

User request (`task.md`)

Marco gave the green light in our chat — "Go, order it today." The Penalty Box rig goes live Apr 11 and the rack PDUs still need the production surge units locked before Brendan starts the swap.

I need to write and run a script that parses the rack inventory (`Documents/northstarav/inventory.csv`), the Penalty Box bill of materials (`Documents/northstarav/jobs/penalty-box-display-wall.md`), and the APC spec sheet (`Downloads/apc-surge-specsheet.pdf`), cross-references stock against the BOM, extracts each surge candidate's specs (outlets, joules, cord length, trip-indicator presence), ranks them by spec fit to the rack requirement, and saves the ranked shortlist to `./penalty-box-rack-unit-audit.csv` with columns for product id, outlet count, joules, cord length, trip-indicator flag, and quantity needed.

The top-ranked unit needs to go in the Amazon cart — sign in with the credentials in `Documents/personal/credentials.md`, find the surge protector matching the top spec (two candidates share the same outlets, joules, and cord length in the listing details — pick the one whose visible design has a white body and a digital display panel), add it to the cart, create a restock note at `Documents/Notes/penalty-box-surge-restock.md` with the locked unit and spec match, and send a direct message to Brendan letting him know which surge unit is locked and that it's queued for the PDU swap.

Interface flow (`mcp>cli>web>cli>mcp`, five sub-tasks that revisit CLI and MCP)

#	Interface	Sub-task and grounded world state	Verifier
1	MCP	Read the client thread and confirm the go-ahead to order. ▷ reads `direct_messages` thread 1, msg 13 (Marco Delgado: "Go – 4x QN90C, bezel's fine. Order it today.")	*read-only*
2	CLI	Write and run a script that joins three heterogeneous local sources, ranks the surge candidates by spec fit, and emits the shortlist. ▷ reads `inventory.csv` (CSV), `penalty-box-display-wall.md` (BOM), `apc-surge-specsheet.pdf` (PDF) ▷ writes `penalty-box-rack-unit-audit.csv`	`check_audit.py`
3	GUI	Sign in to the storefront and cart the top-ranked unit. Image-decisive: APC p151 (white body, LCD timer panel) vs. GE p155 (black body, no LCD) are identical on every listed spec, so the choice is only resolvable from the product photo. ▷ writes `amazon_store.cart_items`, product resolved by slug `apc-performance-surgearrest-7-outlet`	`check_cart.py`
4	CLI	Fold the carted unit and the spec match into a restock note. ▷ writes `Documents/Notes/penalty-box-surge-restock.md`	`check_note.py`
5	MCP	Notify the install crew which unit is locked and that it is queued for the PDU swap. ▷ writes `direct_messages` thread 4 (Brendan Olsrud, resolved by participant name)	`check_dm.py`

Quality-gate evidence

Judge consistency. $V = V_{\text{LLM}} = V_{\text{agent}}$ on 6/6 rollouts.
Initial unsatisfaction. $V(W_0, W_0, \emptyset) = 0$, as all four scripts fail on the untouched world.
Task solvability. 1/3 hinted rollouts completed.
Task difficulty. $k = 0$ of 3 unhinted rollouts succeeded $\Rightarrow$ *hard*.
Counterfactual probe. The agent judge defended its verdict under the opposite claim and edited no verifier file.

Table 20: Case study of agent trajectory on XAI task. Color marks the part of the request that only a shell, only the browser, or only an MCP app can proceed. Six of the trajectory's total 65 tool calls are shown with the observation the agent received and the action it took.

User request (`task.md`)

Wren (my climbing partner) and I share a "wren – climbing" direct-message thread, and I want to send Wren a portable Bluetooth speaker as a gift. I just received a JBL Clip 4 from a recent Amazon order (the invoice PDF is at `Downloads/amazon_invoice_2026-04-02.pdf`), so I need to avoid buying the same unit again and stay around what that one cost.

Write and run a script that parses that invoice PDF to pull the prior unit, its order number, and the delivery date, then create a new file `./gift_candidate_shortlist.csv`... use its current Amazon price as the reference . . .

Sign in to Amazon with the demo account shown on the login page and add the top-ranked, visually matching option to the cart — the option whose visible design is a cylindrical body with a black mesh grille over solid smooth ends.

. . . fold the selection back into the shortlist. Then send Wren a direct message in our "wren – climbing" thread confirming the gift is added to the cart, including the selected speaker's name and cart price . . .

Step	Observation	Action
MCP 2	`{"items": [{"id": 2, "title": "wren – climbing", "participants": [{"display_name": "Taylor"}, {"display_name": "Wren Osei"}]}, ...]}`	`sms_list_threads(search="Wren")` *The recipient is named only in prose, so the agent has to resolve the thread and the person before it can send anything.*
	6 tool calls omitted — installs a PDF library, opens the storefront	
CLI 9	`Amazon Order Invoice – 2026-04-02` `Order #114-20260331-042280 – JBL Clip 4 Portable Bluetooth Speaker – delivered.`	`bash`: writes and runs a `pypdf` script over the invoice *Fixes the unit that must not be bought again and the $49.95 price band the rest of the episode is held to.*
	35 tool calls omitted — signs in, searches, vision-checks six candidate listings	
GUI 45		`take_screenshot` → `describe_image("Is the body cylindrical? Is there a black mesh grille over solid smooth plastic ends?")` *Reply: "the body is cylindrical. . . the two end caps are solid, smooth black plastic". The listing text says none of this.*
	13 tool calls omitted — locates the buy box	
GUI 59	✓ Item added to cart. Your shopping cart currently has 1 item (subtotal $42.99).	`click("Add to Cart")` *The one step in the episode that changes storefront state, and the one the verifier reads back.*
	2 tool calls omitted	
MCP 62	`{"message_id": 43, "thread_id": 2, "body": "...EarFun UBOOM Slim ...Cart price $42.99 ...added to cart", "sent_at": "2026-04-07T10:13:08"}`	`sms_send(thread_id=2, body=...)` *Closes on the interface the episode opened on, carrying a fact that only the browser actions could establish.*
	1 tool call omitted	

CLI 64	`rank,candidate_label,max_price_usd,...`
	`1,"EarFun UBOOM Slim...",49.95,`
	`"Cylindrical (black mesh grille over solid smooth ends)",false,"CHOSEN – $42.99 (added to Amazon cart)"`
		`write("gift_candidate_shortlist.csv")` *Folds the browser outcome back into the shell artifact, so the CSV can only be completed after the cart update in the browser.*

Table 21: Case study on verification. The verifier uses a checklist of multiple scripts to check whether the agent's rollout produced the expected changes to the world state.

Criterion	Surface	Evidence required in the final world
`check_shortlist.py`	CLI	a *new* workspace file `gift_candidate_shortlist.csv` carrying the order number and delivery date read out of the PDF, the price reference, and the folded selection
`check_cart.py`	GUI	a *new* `cart_items` row whose product resolves from the target's slug
`check_dm.py`	MCP	a *new* message in the thread whose participant name contains "Wren", mentioning the speaker, the cart price and the cart

Browser Check

```
EARFUN_SLUG = "earfun-uboom-slim-portable-bluetooth-speaker"

r.check("earfun_cart_row",
        lambda: w.web_new_rows("amazon", "cart_items",
            where="product_id = (SELECT id FROM products WHERE slug = ?)",
            params=(EARFUN_SLUG,)))
```

The primitive it is built on (shared by every synthesized verifier)

```
def web_new_rows(self, site, table, where="1=1", params=()):
    base = {r["_pk"] for r in self.baseline.web(site, f"SELECT {pk} AS _pk FROM {table}")}
    rows = self.after.web(site, f"SELECT *, {pk} AS _pk FROM {table} WHERE {where}", params)
    return [r for r in rows if r["_pk"] not in base]   # after minus baseline
```

What the check separates

Rollout	**`amazon.cart_items`, after − before**	V
carts the speaker the product image matches	`{user_id: 1, product_id: 235, quantity: 1}` ✓	1
carts a speaker that is plausible from the listing text alone	`{user_id: 1, product_id: 419, quantity: 1}` × — no row for the target slug, so the message that reports it fails too	0