Gym-Anything: Turn Any Software into an Agent Environment

Pranjal Aggarwal [1] **Graham Neubig** [1] **Sean Welleck** [1]

https://cmu-l3.github.io/gym-anything

Abstract

Computer-use agents hold the promise of assisting in a wide range of digital economic activities. However, current research has largely focused on short-horizon tasks over a limited set of software. A key reason is that creating environments for complex software requires significant time and human effort, and therefore does not scale. To address this, we introduce **Gym-Anything**, a framework for *converting any software into an interactive computer-use environment*. We frame environment creation itself as a multi-agent task: a coding agent writes setup scripts, downloads real-world data, and configures the software, while producing (visual) evidence of correct setup. An independent audit agent then verifies evidence for environment setup against a quality checklist. Using a taxonomy of economically valuable occupations grounded in U.S. GDP data, we apply this pipeline to 200 software applications with broad occupational coverage. The result is **CUA-World**, a collection of over 10,000 long-horizon tasks spanning domains from medical science and astronomy to engineering and enterprise systems, each configured with realistic data along with train and test splits. Distilling successful trajectories from the training split yields strong gains at multiple student scales (2B, 9B, 35B) CUA-World also includes CUA-World-Long, a challenging long-horizon benchmark with tasks often requiring over 500 steps. We also apply the same auditing principle at test time: a separate VLM reviews completed trajectories and provides feedback on what remains, improving Gemini-3-Flash on CUA-World-Long from 11.5% to 14.0%. We release all code, infrastructure, and benchmark data to facilitate future research in realistic computer-use agents.

[1]Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, USA. Correspondence to: Pranjal Aggarwal <pranjala@andrew.cmu.edu>.

Published as a workshop paper at SCALE – ICML 2026. Copyright 2026 by the author(s).

1. Introduction

Computer-use agents (CUAs, also known as graphical user interface agents) hold the promise of automating and assisting in digitally intensive occupations, which collectively represent trillions of dollars in GDP (U.S. Bureau of Labor Statistics, 2024; U.S. Bureau of Economic Analysis, 2024). Yet whether these agents can handle real professional work remains an open question. Real-world workflows are *long-horizon* and take place in *heterogeneous environments*, often requiring hundreds of steps across diverse software configured with domain-specific data. Existing benchmarks shed little light on these capabilities, as they largely test agents on short-horizon tasks such as changing a desktop wallpaper or filling a web form, over a narrow set of consumer-grade applications (Xie et al., 2024; Zhou et al., 2024a; Koh et al., 2024; Bonatti et al., 2025; Rawles et al., 2025; Deng et al., 2023) that only represent a small slice of the economy (Wang et al., 2026b).

This gap has two consequences. First, *evaluation is unfaithful*: high scores on current benchmarks reveal little about an agent's ability to operate the software that drives real economic activity. Second, *training signal is limited*: short-horizon tasks over few applications may not produce the diverse, long-horizon trajectories needed to train agents for real-world work.

The root cause is that creating realistic environments is prohibitively expensive: each software requires installation, configuration with domain-appropriate data, task design, and verification, often demanding weeks of expert effort per application (Xie et al., 2024; Xu et al., 2025c). The core question we aim to address is: *how can we scale computer-use environments for training and benchmarking agents in settings closer to real-world work?* To address this, we introduce Gym-Anything, a scalable framework for automatically constructing realistic environments across hundreds of economically valuable software applications. We ground the software selection on U.S. digital GDP data,

selecting software based on economic impact and coverage across strategic and STEM domains, occupations, and industries (Figure 1 (i)).

The key idea behind the Gym-Anything framework is that *creating computer-use agent environments is itself a coding and computer-use agent task.* To address this, we develop the Gym-Anything library (§2.2), which reduces every environment to a standardized specification: a set of setup scripts and a configuration file that an AI agent can write to create new environments.

However, without external verification, current agents prompted to create environments in this framework frequently produce incorrect environment setups. The common thread is that the agent's claims about what it has done are not reliable, but the actual state of the environment is. We exploit this observation through a *creation-audit loop* (§2.3; Figure 1 (ii)), in which a creation agent builds an environment and produces evidence of a correct setup through screenshots, execution logs, and data outputs, then an independent audit agent verifies the evidence against a quality checklist and reports issues. In addition, the creation agent builds a shared memory of environment creation strategies that it discovers across attempts.

Next, we adopt a *propose-and-amplify* strategy (Figure 1 (iii); §2.4) for generating realistic tasks at scale. In this pattern, an expensive agentic model proposes a small number of seed tasks per software and runs the tasks, and then a cheaper non-agentic language model amplifies these seeds into a larger set using the seed implementations as in-context examples. To evaluate the resulting tasks, we use a checklist-based VLM verifier that breaks each task into weighted subtasks for partial credit (§2.5). To construct the checklists we leverage data that is embedded in the environment's setup scripts. Agents do not have access to this information when solving a task, thereby making it a form of *privileged information* that the verifier can leverage to check the agent's outputs.

We use Gym-Anything to construct CUA-World, a collection of over 10,000 tasks across 200 software applications. It includes tasks on three operating systems, and is separated into train and test splits (Table 3). To demonstrate the utility of the training split, we distill trajectories from a strong teacher model into student models at three scales (2B, 9B, 35B), with substantial gains at every scale, and find that performance scales with the number of software and environments in the training set.

We further construct a challenging long-horizon benchmark, CUA-World-Long, consisting of one task per software. Even the strongest frontier model achieves only 27.5% pass rate on CUA-World-Long, highlighting the difficulty of long-horizon tasks. To address this, we apply a similar Test-Time Auditing principle (§4.2), where an independent model reviews the agent's trajectory upon completion and provides feedback on what remains, improving pass rate from 11.5% to 14.0% for Gemini-3-Flash. Although Test-Time Auditing helps, CUA-World-Long remains largely unsolved.

In summary, we contribute (1) Gym-Anything, a modular framework and multi-agent pipeline for converting any software into an interactive computer-use environment; (2) CUA-World, a collection of 10,000 tasks across a GDP-grounded selection of 200 software applications with checklist-based VLM verification, train/test splits, and a challenging long-horizon split (CUA-World-Long) requiring hundreds of steps; (3) training and test-time scaling results, including distillation across three student scales (2B, 9B, 35B) with consistent gains, where the 2B distilled model outperforms models 2× its size, and a test-time audit agent that improves long-horizon performance on CUA-World-Long; and (4) a full release of all code, infrastructure, and benchmark data.

2. Methodology

A task $\mathcal{T} = (\mathcal{E}_{s_0}, p, V)$ consists of an environment $\mathcal{E}$ with initial state s_0, a natural-language instruction p, and a verification function V that maps the agent's trajectory to a score. At each step t, the agent receives an observation o_t (e.g., a screen capture) and executes mouse/keyboard actions a_t, yielding the next observation o_{t+1}.

2.1. GDP-Grounded Software Selection

Determining which software to include in a CUA training dataset and benchmark is an important decision. We use a simple principle: *prioritize software that drives more economic activity.* Unlike prior benchmarks, we ground our selection in U.S. digital economy data. We estimate GDP per occupation, discover software used by each occupation, attribute GDP to individual software, filter to sandboxable candidates, apply tiered selection, and select 200 software applications based on our compute budget.

Estimating GDP per occupation. The O*NET data on the U.S. economy comprises ∼900 standardized occupations, each with publicly available data on employment counts and average wages. [1] For each occupation, we compute a wage bill (employment × mean wage) and scale it to total GDP using national accounts data (U.S. Bureau of Economic Analysis, 2024), yielding a GDP estimate per occupation that sums to the full U.S. GDP.

[1] Occupations from the O*NET database (Peterson et al., 2001); employment and wage data from the Bureau of Labor Statistics (BLS) (U.S. Bureau of Labor Statistics, 2024).

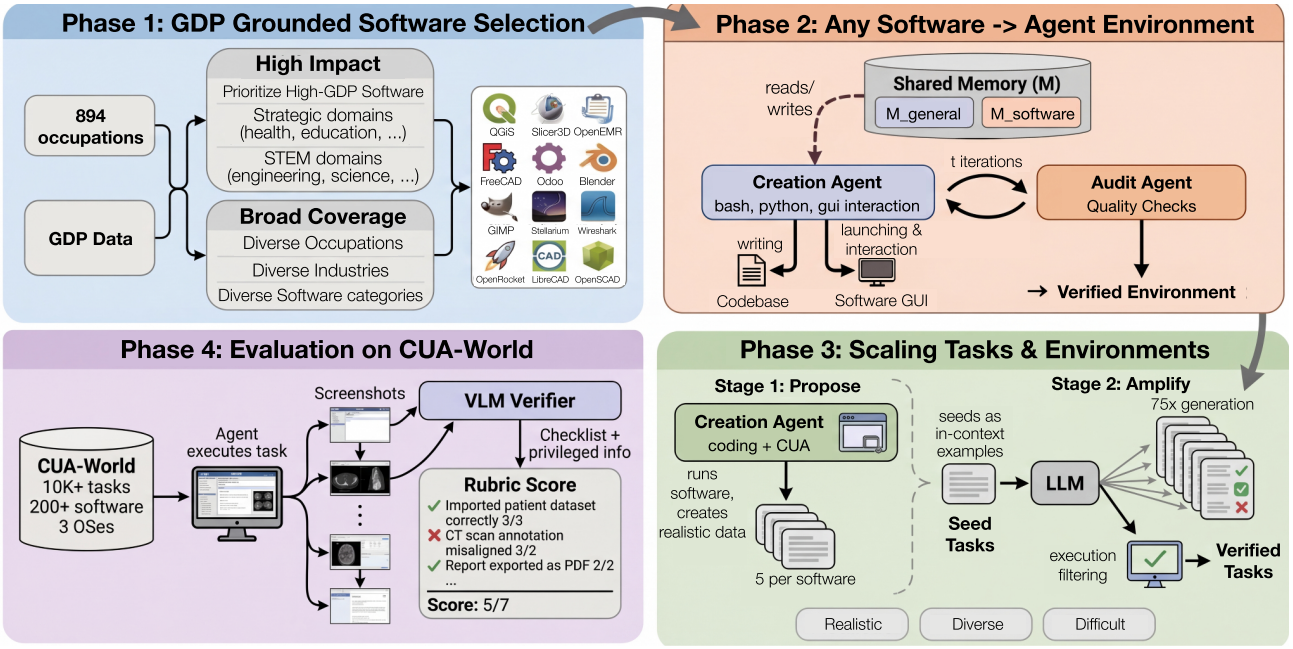


Figure 1. **Overview of the `Gym-Anything` pipeline. Phase 1:** Select ∼200 economically important software grounded in GDP data. **Phase 2:** Each software is converted into an interactive environment via a *creation-audit* loop, where a creation agent iteratively builds and verifies the environment and an audit agent checks quality; learnings accumulate in a shared memory. **Phase 3:** Tasks are scaled with a *propose-and-amplify* pattern, in which an expensive agentic model creates high-quality seed tasks, then a cheaper LLM generates more tasks (75×) using seeds as in-context examples. **Phase 4:** Agents are evaluated on `CUA-World` via a checklist-based VLM verifier with privileged information.

Discovering software per occupation. Next, we need to know what software each occupation actually uses. We use an LLM with web-search access to discover relevant software categories and enumerate software per category for each occupation, producing a catalog of ∼16,600 software applications across ∼1,400 categories.

Attributing GDP to software. Not all of an occupation's output involves computers, and not all computer work uses the same software. We decompose each occupation's GDP into a software-level estimate: $\text{GDP}_{\text{software}} = \sum_{\text{occ}} \text{GDP}_{\text{occ}} \times p_{\text{computer}} \times s_{\text{category}} \times s_{\text{software}}$ Here, p_{computer} is the fraction of the occupation's work that involves computers (available from occupational surveys), s_{category} is the share of that computer work attributed to a software category (e.g., "spreadsheets" for an accountant), and s_{software} is the software's share within its category (e.g., Excel's share of spreadsheets). The share factors s_{category} and s_{software} are estimated by an LLM with web-search access.

Filtering to sandboxable software. We classify each software application as sandbox-ready if it satisfies the following constraints: (a) self-hostable, (b) free-tier, (c) has a GUI, and (d) does not require hardware that cannot be simulated. We select ∼3,400 of ∼16,600 that satisfy these constraints. Further, when a software is not sandboxable, we substitute the closest sandboxable alternative from the same software category, aiming to preserve the economic signal.

Tiered selection. From the filtered catalog, we select 200 software applications across five tiers that balance economic importance with diversity: (1) highest-GDP software overall, (2a/2b) strategically important domains and STEM domains, (3) cycling through all 22 SOC major groups to select ∼5 per group, (4) software unique to specific occupations or domains not yet covered, and (5) software from uncovered categories, ranked by GDP. The pipeline is fully automated and extensible (Appendix F).

2.2. The Gym-Anything Library

In `Gym-Anything`, each environment is defined by a simple specification: three sequential setup scripts and a declarative configuration file. This design reduces environment creation to a scripting task: users and AI agents create new environments by writing setup scripts and key-value configurations, then interact with any environment through a standard gymnasium-style API (Towers et al., 2025; Brockman et al., 2016). This specification is simple enough that an LLM agent can author environments autonomously (§2.3), yet expressive enough to capture complex, production-grade software configured with realistic data. Combined with network-process-file isolation, this enables massive paral-

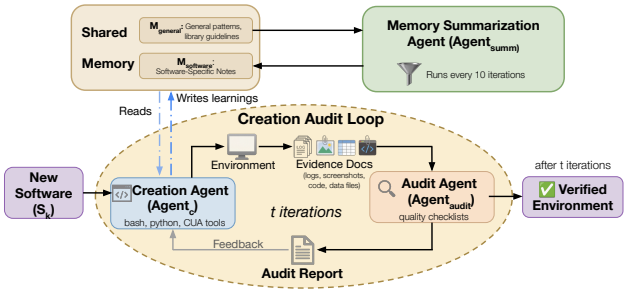


Figure 2. **The creation-audit loop.** Creation Agent writes setup scripts and produces evidence (screenshots, logs); Audit Agent evaluates the evidence against quality checklists and returns feedback. Learnings accumulate in a shared memory M, periodically condensed by a Summarization Agent.

lelization; in our experiments we run 400+ concurrent environments across 1,600 CPUs (Appendix D).

2.3. Scaling Software with a Creation-Audit Loop

Setting up real-world software as interactable environments is hard, laborious, and time-consuming, even for expert humans (Xie et al., 2024; Xu et al., 2025c; He et al., 2024). Each software requires installation, configuration with domain-appropriate data, and verification; for instance, a radiology tool requires annotated clinical CT scans, while an ERP system needs transaction histories and vendor accounts. This often demands weeks of expert effort per application, naturally limiting scalability. While the Gym-Anything library constrains creating environments to a coding task, naively prompting even state-of-the-art agents results in poor environments. For instance the agent stops early, uses fake placeholder data, leaves the software at the wrong starting screen, or claims things are done without actually verifying them. We therefore propose a multi-agent framework that iteratively creates, audits, and improves environments, while accumulating learnings in a shared memory.

Multi-agent framework. Each agent in our framework is an instance of Claude Opus 4.5/4.6 (Anthropic, 2025) run via Claude Code, differentiated by a.) access to specific tools, and b.) the objective described by its system prompt. In a nutshell, these agents iteratively generate environments, audit the quality and improve them, and document the learnings for future attempts in a shared memory M.

Creation agent (Agent_C). This is a coding agent equipped with bash, python, and computer-use (for visual grounding) tools, with complete access to the `Gym-Anything` library and all previously created environments. Before writing any scripts, the agent first researches how the software should be configured, finds and downloads real-world data for the environment (e.g., public medical imaging datasets for radiology software, published email corpora for messaging clients), and studies similar previously created environments. It then implements the setup scripts, launches the environment, takes screenshots, uses visual grounding to check the application reached the expected state, and iteratively debugs failures. Crucially, the agent is required to produce evidence that the software was set up correctly in the form of screenshots of the running software, execution logs, etc. (see Appendix I for an example).

However, the agent often declares the task done prematurely. For instance, it may use placeholder data instead of real datasets, leave the software at the wrong screen, or never verify the task by actual execution. We speculate that these failures are due to context fatigue (Liu et al., 2024; Sinha et al., 2026): after hundreds of thousands of tokens, the agent loses track of what it still needs to do. To address this, whenever the agent stops we re-prompt it to reread the setup guidelines, reread the checklists, and complete any requirements it may have skipped. We find this simple technique recovers many omissions.

Audit agent ($\text{Agent}_{\text{audit}}$). While Agent_C typically gets the environment running, its claims about what it has done are not always reliable. For instance, it may leave the software at a setup wizard instead of the main screen, use placeholder data, or skip verification entirely. However, the evidence produced above reveals the actual state of the environment regardless of what the agent claims: a screenshot shows whether the software is running correctly or stuck on an error screen.

To verify this evidence, we use $\text{Agent}_{\text{audit}}$, a similar coding+computer-use agent that acts as an adversary to Agent_C and evaluates whether the evidence demonstrates that the environment satisfies a set of quality checklists (see Appendix I). It does so by analyzing the screenshots and logs, inspecting the actual config and script files, and if necessary actually running the environment. Given the implementation of software S_k, the audit agent outputs an audit report detailing what is correctly implemented and what the critical issues are (Appendix J).

In principle, both Agent_C and $\text{Agent}_{\text{audit}}$ have access to the same tools and files; the only difference lies in their prompt. We find this separation removes self-confirmation bias, improves interpretability via written audits, and catches self-misleading claims (Appendix J.5). The audit findings are fed back to Agent_C for correction, and this loop runs for t iterations. A key feature of our framework is that agents accumulate learnings in a shared memory, allowing them to improve over iterations.

Shared memory. The creation agent maintains a shared memory M, effectively a directory of files that grows over time. M is initialized with the hand-written prompt for Agent_C and evolves as agents add their learnings. After each environment, Agent_C documents what failed, and how it fixed it, updating M in two places: software-specific

notes M_{soft} and general notes M_{general} that could help future agents. This ensures sublinear growth in creation time: as more environments are built, newer environments can be created faster. To prevent M_{soft} from outgrowing the agent's context, every L environments a summarization agent ($\text{Agent}_{\text{summ}}$) consolidates patterns from M_{soft} into M_{general}.

2.4. Scaling Tasks via Propose-and-Amplify

Recall that a task requires a starting environment state $\mathcal{E}(s_0)$, a natural-language instruction p, and a verifier V. Once the base software is configured, creating new tasks reduces to generating these task-specific assets. Nonetheless, naively prompting a model to generate these often results in subpar quality. For instance, setup scripts reference non-existent data, formats mismatch the software's expectations, and instructions are either trivially simple or impossible to execute from the given starting state. Conversely, relying purely on agentic models to author and validate every task is prohibitively expensive for scaling.

To scale task creation efficiently, we propose a *propose-and-amplify* strategy. First, a proposer agent (Claude Opus 4.5/4.6 via Claude Code, equipped with computer-use tools) proposes a small set of high quality, difficult *seed tasks* per software. The agent is provided with a set of guidelines for high quality tasks across three dimensions: a.) realism (does the instruction reflect a genuine, real-world use case?), b.) difficulty (does the task require a long-horizon, multi-step trajectory to solve?), and c.) diversity (do the tasks cover varied functionalities of the software?). An agentic loop is necessary here because the model must actively run the software, search, download or generate realistic data, interact via the GUI, and verify the resulting state. Crucially, this expensive step only occurs once per software.

Second, for amplification, a non-agentic LLM (Gemini 3 Pro) uses these high-quality seeds as in-context examples to generate additional tasks at scale. While the agentic seeds ensure realism and difficulty in further generated tasks, naively sampling from a non-agentic LLM often yields repetitive or very similar instructions. To enforce diversity, we generate tasks sequentially, providing the model with all previously generated instructions $1, \ldots, t$ as context for task $t{+}1$. We subsequently apply semantic similarity filtering to discard duplicate tasks.

Finally, because the non-agentic LLM generates tasks without interactive execution, we implement an automated filtering step. We launch each generated task, capture the starting state observation o_0, and pass it alongside the instruction to a VLM to check whether the start state matches the expectation from task description. Tasks that fail this test are filtered from the dataset (Appendix L).

2.5. Task Verification

Evaluating long-horizon trajectories requires V to be both robust and granular. We construct V as a checklist-based VLM verifier augmented with *privileged information*.

Privileged information. Each task's starting state s_0 is configured by the setup scripts $\mathcal{S}$. These scripts contain ground-truth data that is not present in the task description p but is deterministically tied to the environment's configuration. We call this *privileged information* $\mathcal{I} = \text{Extract}(\mathcal{S}, p)$, extracted automatically by a separate coding agent that parses the scripts, retrieves or searches online for the relevant ground truth. For instance, in a medical imaging task, the correct tumor location is already known from the downloaded dataset; in a financial task, the expected account balances are determined by the initialization data. Importantly, $\mathcal{I}$ assists the VLM verifier rather than making the task artificially harder for the evaluated computer-use agent.

Checklist-based verification. The VLM verifier uses $\mathcal{I}$ alongside the task instruction p to generate a granular checklist $C = \{(c_i, w_i)\}_{i=1}^{N}$, where each c_i defines a specific subtask to verify and w_i is its point value. Given the evaluated agent's trajectory τ, the verification score is: $V(\tau) = \sum_{i=1}^{N} w_i \cdot \text{VLM}(\tau, c_i, \mathcal{I})$ where each $\text{VLM}(\tau, c_i, \mathcal{I})$ returns a binary judgment of whether subtask c_i was completed. This formulation allows partial credit without requiring manual annotation. In addition to C, the VLM verifier evaluates a separate integrity checklist C_{int} to ensure the evaluated computer-use agent did not bypass the intended workflow: a.) the intended software was used to complete the task, b.) the required application state was reached through the software's own interface rather than by directly editing configuration or data files, and c.) the agent did not exploit environment artifacts to shortcut the work. Failing any integrity item sets $V(\tau) = 0$.

3. CUA-World

Applying Gym-Anything with our compute budget, the proposer generates 5 and the amplifier 75 tasks per piece of software. After filtering, this yields 12,103 tasks and environments across 200 software applications, each with checklist-based verification. CUA-World is the first collection to simultaneously provide interactive environments at scale (200+ varieties of software, 10K+ tasks), support long-horizon evaluation, cover all 22 major occupation groups, offer automated environment creation, and include a training split (Figure 3; Appendix C).

Contamination filtering. To prevent data leakage between train and test splits, we use an LLM to score pairwise similarity between task instructions, group highly similar tasks together, and assign each group entirely to same split

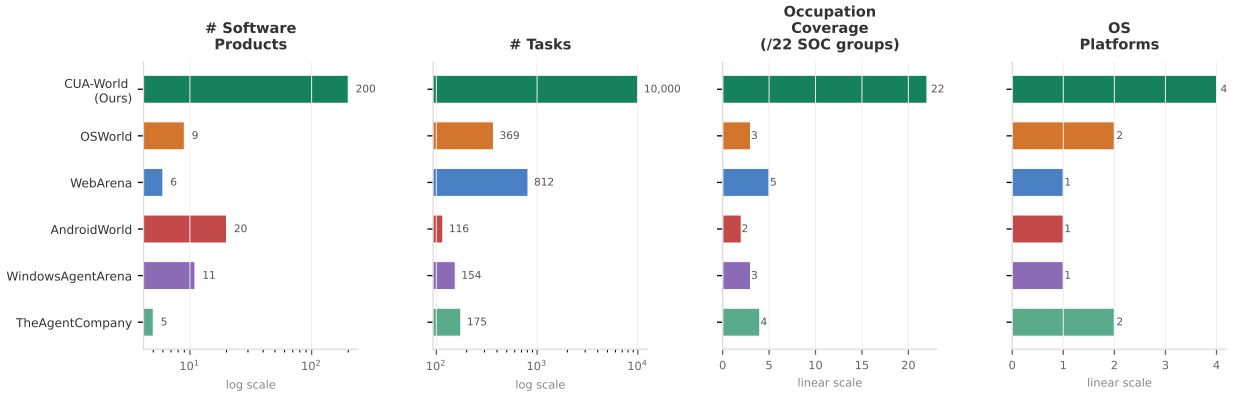


Figure 3. **Quantitative comparison of `CUA-World` against existing benchmarks**.

(Appendix M). Manual verification confirms the pipeline is conservative, i.e. it occasionally flags task pairs that aren't actually duplicates, but rarely misses real overlap.

`CUA-World`-Long. To evaluate agents on extremely long-horizon tasks, we introduce `CUA-World`-Long, a set of 200 tasks (one per software). The key challenge is generating tasks that are genuinely harder than those already in the benchmark while remaining solvable. We address this with a trajectory-guided strategy: for each piece of software, we first generate k trajectories from a strong computer-use agent on existing tasks, then prompt a coding and visual agent to analyze these trajectories. The agent also receives a set of 8 quality guidelines (see Appendix K.1). Based on this analysis, the agent creates a new task designed to be harder than the existing ones for that software application. We manually verify that all 200 tasks are set up correctly. These tasks often require more than 200 steps for human completion, and current models frequently fail even after 500 steps.

4. Experiments

4.1. Training

To evaluate the utility of `CUA-World`-Train, we distill execution trajectories from a strong teacher model (Kimi-K 2.5 (Team et al., 2026)) into student models at three scales: Qwen3-VL-2B-Thinking (Yang et al., 2025), Qwen3.5-9B, and Qwen3.5-35B-A3B. For every task in the training split, we generate $k = 4$ trajectories from the teacher until at least one is correct, and utilize these successful rollouts for fine-tuning. Cumulatively, we collect roughly 2000 trajectories across all 10,000 tasks.

4.2. Test-Time Auditing (TTA) Agent

Some of the tasks in `CUA-World` are extremely long. This opens up a unique opportunity to test agents capable of working over extended horizons. However, we find that current agents often stop after a few dozen steps, making mistakes or prematurely claiming the task is complete when it is not. Inspired by our approach in software generation (§2.3), we introduce an audit agent to address this. Whenever the main model signals that the task is terminated, we run this audit agent. It takes the complete trajectory as input and determines whether the task is actually complete. If the audit agent determines that the task is not completed, it generates an explanation of what is missing. We provide this feedback back to the main computer-use agent, prompting it to continue completing the task.

4.3. Evaluation

We evaluate agents using the checklist-based VLM verifier described in §2.5. Each task's checklist consists of weighted subtasks; we report two metrics: (1) **Average Score** (0-100), the mean checklist score across tasks, which captures partial credit, and (2) **Pass Rate** (%), the fraction of tasks fully completed, i.e., achieving a perfect checklist score. We evaluate on `CUA-World`-Test and `CUA-World`-Long. We use Gemini 3 Flash as the VLM verifier. Each agent is given a maximum budget per episode: 200 steps for `CUA-World`-Test, and 500 steps or $5 whichever hits first for `CUA-World`-Long.

4.4. Main Results

Distillation on `CUA-World`-Train yields consistent gains across model scales. Table 1 shows results on `CUA-World`-Test. Gemini-3-Flash is strongest with 50.1 average score and 22.6% pass rate, followed by Kimi-K 2.5 with 45.1 and 17.8%. Untrained open student models perform poorly (1.6–8.7% pass rate). Distillation on `CUA-World`-Train yields substantial improvements at every scale: Qwen3-VL-2B from 1.6% to 4.4%, outperforming Qwen3-VL-4B, a model 2× its size. Further larger models: Qwen3.5-9B imprvove from 8.2% to 15.8%, and Qwen3.5-35B-A3B from 8.7% to 15.3% both approaching

Table 1. **Results on CUA-World-Test.** Our 2B distilled model outperforms models 2× its size.

Model	Avg.	Pass
Frontier		
Gemini 3 Flash	**50.1**	**22.6**
Kimi-K 2.5	45.1	17.8
Small open		
Qwen3-VL-2B	12.7	1.6
Qwen3-VL-4B	19.3	3.9
Ours (2B distilled)	22.5 (+9.8)	4.4 (+2.8)
Qwen3.5-9B	20.4	8.2
Ours (9B distilled)	36.4 (+16.0)	15.8 (+7.6)
Qwen3.5-35B-A3B	25.3	8.7
Ours (35B distilled)	33.8 (+8.5)	15.3 (+6.6)

Table 2. **Results on CUA-World-Long.** The strongest frontier model reaches only 27.5% pass rate with a 2,000-step budget.

Model	Avg.	Pass
500 steps, $5 cap		
Gemini 3 Flash	36.2	7.5
GPT-5.4	22.7	3.0
Sonnet 4.6	20.5	6.0
Kimi-K 2.5	33.9	5.5
2000 steps, no cap		
Gemini 3 Flash	38.7	11.5
GPT-5.4	**55.5**	**27.5**

performance of teacher models. This demonstrates that CUA-World-Train provides useful supervision signal across the spectrum of open models.

CUA-World-Long is challenging for frontier models. Table 2 shows the performance of multiple frontier models on CUA-World-Long. Even the strongest model, Gemini-3-Flash, achieves a pass rate of only 7.5% and an average score of 36.2. To test whether budget is a bottleneck, we remove the cost cap and raise step limit to 2,000 for GPT-5.4 and Gemini-3-Flash. Both models substantially improve, notably GPT-5.4 reaching 27.5% pass rate. However, these improvements come at substantial test-time cost. On average, Gemini-3-Flash requires 1,300 steps and $16 per trajectory, while GPT-5.4 requires 242 steps and $18 per trajectory.

Scaling Software Applications and Environments: Performance scales with both increasing software and task count. Figure 4a shows how the distilled 2B model's score on CUA-World-Test changes as we scale training data along two axes: the number of software applications (50, 100, 200), and the fraction of tasks (25%, 50%, 100%). Both curves show consistent log-linear performance improvements (∼3.5 point increase) on doubling the data.

Generalization: Distillation improves performance on both seen and unseen software, but gains are much larger on seen software. We train models on 25% and 50% of the 200 software applications and evaluate separately on software used during training (IID) and software held out (OOD); see Appendix R.3 for the figure. The OOD gain recovers only 22–27% of the improvement obtained by training on all 200 software, compared to 65–87% on IID, implying distillation generalizes to unseen software only weakly. The IID recovery, though higher, is still well below 100%, meaning the *other* software in its training also helps. Together these findings argue for scalable environment creation as in Gym-Anything: agents need training on each target software (limited cross-software transfer) and a diverse pool of additional software (which helps even seen software).

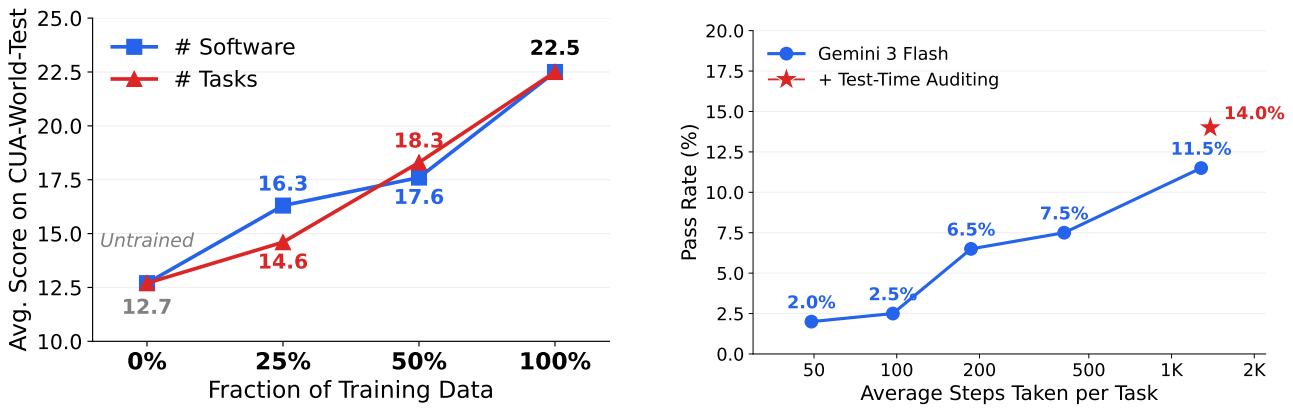
Pass rate scales with step budget, and Test-Time Auditing provides further gains. Figure 4b shows how Gemini 3 Flash's pass rate on CUA-World-Long changes as we increase the maximum step budget per episode. Pass rate stays low between 50 and 100 average steps, then rises steeply at higher budgets (6.5% at ∼200 steps, 7.5% at ∼400 steps), reaching 11.5% at ∼1,300 average steps. Further, the TTA agent uplifts performance, raising pass rate to 14.0% under the same 2,000-step cap.

4.5. Other Analysis

Verifier Robustness: Checklist-based verification achieves highest human agreement. On 60 randomly sampled Gemini-3-Flash trajectories from CUA-World-Test, our checklist-based VLM verifier achieves 93.3% task-level agreement with human annotations, compared to 81.7% for a direct VLM verifier and 43.3% for programmatic verifiers. Across ∼3,000 trajectories, the integrity checklist flags only ∼1.5% of high-scoring runs, of which 15/21 are true positives, catching shortcuts at a low FPR (Appendix R.5).

Pipeline Ablations: We ablate three pipeline choices (Appendix R). **The strongest teacher does not produce the strongest student**: Opus 4.5 is the strongest teacher (53.5 avg. score) yet Kimi-K 2.5 produces the best student (25.3 vs. 19.3 for Qwen3-VL-4B). **The proposal step substantially improves amplified task quality**: tasks amplified from seed examples achieve 88.9% setup success rate vs. 55.2% without seeds. **Cross-model auditing catches more issues than self-auditing**, surfacing on average 2.1 additional issues per environment.

Additional analysis. The appendix contains further results and analysis. **Trajectory behavioral patterns:** an automated pipeline over ∼3,000 trajectories yields 15 canonical patterns; such as retry loops in failed trajectories (Appendix S). **Performance by software category:** high visual complexity is a persistent bottleneck for smaller models and all models show lower performance in software requiring specialized-knowledge (Appendix R.3). We further show how the creation-audit loop iteratively



(a) Training data scaling on `CUA-World`-Test.

(b) Test-time compute scaling on `CUA-World`-Long.

Figure 4. **Scaling behavior on `CUA-World`.** **(a)** Training data scaling: varying the number of software (50, 100, 200) or the fraction of tasks (25%, 50%, 100%). Both axes improve roughly log-linearly (∼3.5 points per doubling). **(b)** Test-time compute scaling: pass rate vs. average steps per task for budgets of 50, 100, 200, 500, 2,000 steps. The star marks Test-Time Auditing (TTA, §4.2).

corrects issues across rounds (Appendix J.4); verify that `CUA-World` covers all 22 SOC major occupation groups (Appendix O); detail the contamination filtering statistics (Appendix M); and provide 12 representative task examples with agent trajectories (Appendix L).

5. Related Work

Benchmarks for computer-use agents. Static benchmarks (Deng et al., 2023; Rawles et al., 2023; Li et al., 2024; Kapoor et al., 2024) evaluate via action-matching rather than execution, which penalizes valid alternative strategies. Interactive benchmarks across web (Liu et al., 2018; Zhou et al., 2024a; Koh et al., 2024; Drouin et al., 2024; Boisvert et al., 2024), desktop (Xie et al., 2024; Bonatti et al., 2025; Cao et al., 2024; Gao et al., 2024; Xu et al., 2025b; Sun et al., 2026; Aggarwal & Welleck, 2025), and mobile (Rawles et al., 2025) are execution-based but manually constructed and small-scale: none combines training data, long-horizon tasks, and broad occupational coverage. Programmatic verifiers (Xie et al., 2024; Zhou et al., 2024a) are also labor-intensive to create and offer only binary pass/fail. `CUA-World` addresses these gaps. Automated environment creation yields 10,000+ tasks across 200+ software with train/test and long-horizon splits, and a checklist-based VLM verifier provides fine-grained evaluation via privileged information (Appendix C).

Automated environment and task generation. Some methods generate tasks within pre-existing environments (Xu et al., 2025c; Sun et al., 2025a; Zhou et al., 2024b; Murty et al., 2024; Qi et al., 2025) but cannot create new ones. LLM-based environment generation has been explored for non-GUI domains (Hu et al., 2025; Yang et al., 2024; Wang et al., 2026a; Pan et al., 2025; Zhu et al., 2025; Zhang et al., 2025), but real GUI software requires installation, configuration, and realistic data that these methods do not handle. Concurrent GUI-GENESIS (Cao et al., 2026) synthesizes lightweight web replicas, but does not install or configure real software or target long-horizon evaluation. `Gym-Anything` addresses these gaps with three components: a creation-audit loop that converts real GUI software into interactive environments, a propose-and-amplify strategy that grounds tasks in actual software execution, and a shared memory that accumulates learnings across environments. We provide an extended related work in Appendix T.

6. Conclusion

We introduced `Gym-Anything`, a scalable multi-agent framework for converting arbitrary software into interactive computer-use environments. Applying this framework, we built `CUA-World`, a GDP-grounded collection of over 10K tasks and environments across 200 software applications spanning diverse occupations, domains, and operating systems, together with checklist-based VLM verification and train/test/long-horizon splits. We further showed that `CUA-World` provides useful supervision for training smaller agents through distillation, and that test-time auditing can improve performance on especially long-horizon tasks. At the same time, `CUA-World`-Long is challenging even for frontier models, indicating that realistic computer-use remains far from solved. We hope that `Gym-Anything`, `CUA-World`, and the released code and infrastructure provide a foundation for future work on long-horizon, economically relevant computer-use, including more capable agents, stronger verifiers, and broader coverage of the software that underlies real-world digital work.

References

Acemoglu, D. The simple macroeconomics of ai. *SSRN Electronic Journal*, 2024. doi: 10.2139/ssrn.4843046. URL http://dx.doi.org/10.2139/ssrn.4843046.

Aggarwal, P. and Welleck, S. Programming with pixels: Can computer-use agents do software engineering? *arXiv preprint arXiv:2502.18525*, 2025. URL https://arxiv.org/abs/2502.18525.

Anthropic. The claude model family, 2025. URL https://www.anthropic.com/claude.

Bai, H., Zhou, Y., Cemri, M., Pan, J., Suhr, A., Levine, S., and Kumar, A. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning, 2024. URL https://arxiv.org/abs/2406.11896.

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., Zhong, H., Zhu, Y., Yang, M., Li, Z., Wan, J., Wang, P., Ding, W., Fu, Z., Xu, Y., Ye, J., Zhang, X., Xie, T., Cheng, Z., Zhang, H., Yang, Z., Xu, H., and Lin, J. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025. URL https://arxiv.org/abs/2502.13923.

Boisvert, L., Thakkar, M., Gasse, M., Caccia, M., De Chezelles, T. L. S., Cappart, Q., Chapados, N., Lacoste, A., and Drouin, A. WorkArena++: Towards compositional planning and reasoning-based common knowledge work tasks. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 5996–6051. Curran Associates, Inc., 2024. doi: 10.52202/079017-0195. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0b82662b6c32e887bb252a74d8cb2d5e-Paper-Datasets_and_Benchmarks_Track.pdf.

Bonatti, R., Zhao, D., Bonacci, F., Dupont, D., Abdali, S., Li, Y., Lu, Y., Wagle, J., Koishida, K., Bucker, A., Jang, L. K., and Hui, Z. Windows Agent Arena: Evaluating multi-modal OS agents at scale. In Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., and Zhu, J. (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4874–4910. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/bonatti25a.html.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym, 2016. URL https://arxiv.org/abs/1606.01540.

Cao, R., Lei, F., Wu, H., Chen, J., Fu, Y., Gao, H., Xiong, X., Zhang, H., Mao, Y., Hu, W., Xie, T., Xu, H., Zhang, D., Wang, S., Sun, R., Yin, P., Xiong, C., Ni, A., Liu, Q., Zhong, V., Chen, L., Yu, K., and Yu, T. Spider2-V: How far are multimodal agents from automating data science and engineering workflows? In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 107703–107744. Curran Associates, Inc., 2024. doi: 10.52202/079017-3421. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c2f71567cd53464161cab3336e8fc865-Paper-Datasets_and_Benchmarks_Track.pdf.

Cao, Y., Ran, D., Wu, M., Guo, Y., Chen, X., Li, A., Cao, G., Zhi, G., Yu, H., Li, L., et al. Gui-genesis: Automated synthesis of efficient environments with verifiable rewards for gui agent post-training. *arXiv preprint arXiv:2602.14093*, 2026.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2Web: Towards a generalist agent for the web. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf.

Drouin, A., Gasse, M., Caccia, M., Laradji, I. H., Del Verme, M., Marty, T., Vazquez, D., Chapados, N., and Lacoste, A. WorkArena: How capable are web agents at solving common knowledge work tasks? In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

ElMallah, R., Chhajer, K., and Lee, C.-G. Score the steps, not just the goal: Vlm-based subgoal evaluation for robotic manipulation, 2025. URL https://arxiv.org/abs/2509.19524.

Eloundou, T., Manning, S., Mishkin, P., and Rock, D. Gpts are gpts: An early look at the labor market impact potential of large language models, 2023. URL https://arxiv.org/abs/2303.10130.

Felten, E., Raj, M., and Seamans, R. Occupational, industry, and geographic exposure to artificial intelligence: A novel dataset and its potential uses. *Strategic Management Journal*, 42(12):2195–2217, 2021. doi: 10.1002/smj.3286. URL http://dx.doi.org/10.1002/smj.3286.

Frey, C. B. and Osborne, M. A. The future of employment: How susceptible are jobs to computerisation? *Technological Forecasting and Social Change*, 114:254–280, 2017. doi: 10.1016/j.techfore.2016.08.019. URL http://dx.doi.org/10.1016/j.techfore.2016.08.019.

Gabor, J., Lynch, J., and Rosenfeld, J. Evilgenie: A reward hacking benchmark, 2025. URL https://arxiv.org/abs/2511.21654.

Gao, D., Ji, L., Bai, Z., Ouyang, M., Li, P., Mao, D., Wu, Q., Zhang, W., Wang, P., Guo, X., Wang, H., Zhou, L., and Shou, M. Z. AssistGUI: Task-oriented PC graphical user interface automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13289–13298, June 2024.

He, Y., Jin, J., Xia, S., Su, J., Fan, R., Zou, H., Hu, X., and Liu, P. PC agent: While you sleep, AI works – a cognitive journey into digital world. *arXiv preprint arXiv:2412.17589*, 2024.

He, Y., Jin, J., and Liu, P. Efficient agent training for computer use, 2026. URL https://arxiv.org/abs/2505.13909.

Hu, M., Zhao, P., Xu, C., Sun, Q., Lou, J., Lin, Q., Luo, P., and Rajmohan, S. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation, 2025. URL https://arxiv.org/abs/2408.00764.

Kapoor, R., Butala, Y. P., Russak, M., Koh, J. Y., Kamble, K., AlShikh, W., and Salakhutdinov, R. OmniACT: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *Computer Vision – ECCV 2024*, pp. 161–178. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-73113-6_10. URL http://dx.doi.org/10.1007/978-3-031-73113-6_10.

Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M., Huang, P.-Y., Neubig, G., Zhou, S., Salakhutdinov, R., and Fried, D. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.50. URL https://aclanthology.org/2024.acl-long.50/.

Li, W., Bishop, W., Li, A., Rawles, C., Campbell-Ajala, F., Tyamagundlu, D., and Riva, O. On the effects of data scale on UI control agents. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 92130–92154. Curran Associates, Inc., 2024. doi: 10.52202/079017-2925. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/a79f3ef3b445fd4659f44648f7ea8ffd-Paper-Datasets_and_Benchmarks_Track.pdf.

Lin, B. Y., Deng, Y., Chandu, K., Brahman, F., Ravichander, A., Pyatkin, V., Dziri, N., Bras, R. L., and Choi, Y. Wildbench: Benchmarking llms with challenging tasks from real users in the wild, 2024. URL https://arxiv.org/abs/2406.04770.

Liu, E. Z., Guu, K., Pasupat, P., Shi, T., and Liang, P. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1802.08802. ICLR 2018; arXiv:1802.08802.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Mitra, A., Corro, L. D., Zheng, G., Mahajan, S., Rouhana, D., Codas, A., Lu, Y., ge Chen, W., Vrousgos, O., Rosset, C., Silva, F., Khanpour, H., Lara, Y., and Awadallah, A. Agentinstruct: Toward generative teaching with agentic flows, 2024. URL https://arxiv.org/abs/2407.03502.

Murty, S., Manning, C., Shaw, P., Joshi, M., and Lee, K. Bagel: Bootstrapping agents by guiding exploration with language, 2024. URL https://arxiv.org/abs/2403.08140.

Pahuja, V., Lu, Y., Rosset, C., Gou, B., Mitra, A., Whitehead, S., Su, Y., and Awadallah, A. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents, 2025. URL https://arxiv.org/abs/2502.11357.

Pan, J., Zhang, Y., Tomlin, N., Zhou, Y., Levine, S., and Suhr, A. Autonomous evaluation and refinement of digital agents, 2024. URL https://arxiv.org/abs/2404.06474.

Pan, J., Wang, X., Neubig, G., Jaitly, N., Ji, H., Suhr, A., and Zhang, Y. Training software engineering agents and verifiers with swe-gym, 2025. URL https://arxiv.org/abs/2412.21139.

Patwardhan, T., Dias, R., Proehl, E., Kim, G., Wang, M., Watkins, O., Fishman, S. P., Aljubeh, M., Thacker, P., Fauconnet, L., Kim, N. S., Chao, P., Miserendino, S., Chabot, G., Li, D., Sharman, M., Barr, A., Glaese, A., and Tworek, J. Gdpval: Evaluating ai model performance on real-world economically valuable tasks, 2025. URL https://arxiv.org/abs/2510.04374.

Peterson, N. G., Mumford, M. D., Borman, W. C., Jeanneret, P. R., Fleishman, E. A., Levin, K. Y., Campion, M. A., Mayfield, M. S., Morgeson, F. P., Pearlman, K., Gowing, M. K., Lancaster, A. R., Silver, M. B., and Dye, D. M. Understanding work using the occupational information network (O*NET): Implications for practice and research. *Personnel Psychology*, 54 (2):451–492, 2001. doi: 10.1111/j.1744-6570.2001.tb00100.x. URL http://dx.doi.org/10.1111/j.1744-6570.2001.tb00100.x.

Qi, Z., Liu, X., Iong, I. L., Lai, H., Sun, X., Zhao, W., Yang, Y., Yang, X., Sun, J., Yao, S., Zhang, T., Xu, W., Tang, J., and Dong, Y. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning, 2025. URL https://arxiv.org/abs/2411.02337.

Qin, Y., Ye, Y., Fang, J., Wang, H., Liang, S., Tian, S., Zhang, J., Li, J., Li, Y., Huang, S., Zhong, W., Li, K., Yang, J., Miao, Y., Lin, W., Liu, L., Jiang, X., Ma, Q., Li, J., Xiao, X., Cai, K., Li, C., Zheng, Y., Jin, C., Li, C., Zhou, X., Wang, M., Chen, H., Li, Z., Yang, H., Liu, H., Lin, F., Peng, T., Liu, X., and Shi, G. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

Rawles, C., Li, A., Rodriguez, D., Riva, O., and Lillicrap, T. AndroidInTheWild: A large-scale dataset for Android device control. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 59708–59728. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/bbbb6308b402fe909c39dd29950c32e0-Paper-Datasets_and_Benchmarks.pdf.

Rawles, C., Clinckemaillie, S., Chang, Y., Waltz, J., Lau, G., Fair, M., Li, A., Bishop, W., Li, W., Campbell-Ajala, F., Toyama, D., Berry, R., Tyamagundlu, D., Lillicrap, T., and Riva, O. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC. ICLR 2025; arXiv:2405.14573.

Sinha, A., Arun, A., Goel, S., Staab, S., and Geiping, J. The illusion of diminishing returns: Measuring long horizon execution in llms, 2026. URL https://arxiv.org/abs/2509.09677.

Sun, Q., Cheng, K., Ding, Z., Jin, C., Wang, Y., Xu, F., Wu, Z., Jia, C., Chen, L., Liu, Z., Kao, B., Li, G., He, J., Qiao, Y., and Wu, Z. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis, 2025a. URL https://arxiv.org/abs/2412.19723.

Sun, Q., Liu, Z., Ma, C., Ding, Z., Xu, F., Yin, Z., Zhao, H., Wu, Z., Cheng, K., Liu, Z., Wang, J., Li, Q., Tang, X., Xie, T., Feng, X., Li, X., Kao, B., Wang, W., Qi, B., Kong, L., and Wu, Z. ScienceBoard: Evaluating multimodal autonomous agents in realistic scientific workflows. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10008621. ICLR 2026; arXiv:2505.19897.

Sun, Z., Liu, Z., Zang, Y., Cao, Y., Dong, X., Wu, T., Lin, D., and Wang, J. Seagent: Self-evolving computer use agent with autonomous learning from experience, 2025b. URL https://arxiv.org/abs/2508.04700.

Team, K., Bai, T., Bai, Y., Bao, Y., Cai, S. H., Cao, Y., Charles, Y., Che, H. S., Chen, C., Chen, G., Chen, H., Chen, J., Chen, J., Chen, J., Chen, J., Chen, K., Chen, L., Chen, R., Chen, X., Chen, Y., Chen, Y., Chen, Y., Chen, Y., Chen, Y., Chen, Y., Chen, Y., Chen, Y., Chen, Z., Chen, Z., Cheng, D., Chu, M., Cui, J., Deng, J., Diao, M., Ding, H., Dong, M., Dong, M., Dong, Y., Dong, Y., Du, A., Du, C., Du, D., Du, L., Du, Y., Fan, Y., Fang, S., Feng, Q., Feng, Y., Fu, G., Fu, K., Gao, H., Gao, T., Ge, Y., Geng, S., Gong, C., Gong, X., Gongque, Z., Gu, Q., Gu, X., Gu, Y., Guan, L., Guo, Y., Hao, X., He, W., He, W., He, Y., Hong, C., Hu, H., Hu, J., Hu, Y., Hu, Z., Huang, K., Huang, R., Huang, W., Huang, Z., Jiang, T., Jiang, Z., Jin, X., Jing, Y., Lai, G., Li, A., Li, C., Li, C., Li, F., Li, G., Li, G., Li, H., Li, H., Li, J., Li, J., Li, J., Li, L., Li, M., Li, W., Li, W., Li, X., Li, X., Li, Y., Li, Y., Li, Y., Li, Y., Li, Z., Li, Z., Liao, W., Lin, J., Lin, X., Lin, Z., Lin, Z., Liu, C., Liu, C., Liu, H., Liu, L., Liu, S., Liu, S., Liu, S., Liu, T., Liu, T., Liu, W., Liu, X., Liu, Y., Liu, Y., Liu, Y., Liu, Y., Liu, Y., Liu, Z., Liu, Z., Lu, E., Lu, H., Lu, Z., Luo, J., Luo, T., Luo, Y., Ma, L., Ma, Y., Mao, S., Mei, Y., Men, X., Meng, F., Meng, Z., Miao, Y., Ni, M., Ouyang, K., Pan, S., Pang, B., Qian, Y., Qin, R., Qin, Z., Qiu, J., Qu, B., Shang, Z., Shao, Y., Shen, T., Shen, Z., Shi, J., Shi, L., Shi, S., Song, F., Song, P., Song, T., Song, X., Su, H., Su, J., Su, Z., Sui, L., Sun, J., Sun, J., Sun, T., Sung, F., Tai, Y., Tang, C.,

Tang, H., Tang, X., Tang, Z., Tao, J., Teng, S., Tian, C., Tian, P., Wang, A., Wang, B., Wang, C., Wang, C., Wang, C., Wang, D., Wang, D., Wang, D., Wang, F., Wang, H., Wang, H., Wang, H., Wang, H., Wang, H., Wang, J., Wang, J., Wang, J., Wang, K., Wang, L., Wang, Q., Wang, S., Wang, S., Wang, S., Wang, W., Wang, X., Wang, X., Wang, Y., Wang, Y., Wang, Y., Wang, Y., Wang, Y., Wang, Y., Wang, Z., Wang, Z., Wang, Z., Wang, Z., Wang, Z., Wang, Z., Wei, C., Wei, M., Wen, C., Wen, Z., Wu, C., Wu, H., Wu, J., Wu, R., Wu, W., Wu, Y., Wu, Y., Wu, Y., Wu, Z., Xiao, C., Xie, J., Xie, X., Xie, Y., Xin, Y., Xing, B., Xu, B., Xu, J., Xu, J., Xu, J., Xu, L. H., Xu, L., Xu, S., Xu, W., Xu, X., Xu, X., Xu, Y., Xu, Y., Xu, Y., Xu, Z., Xu, Z., Yan, J., Yan, Y., Yang, G., Yang, H., Yang, J., Yang, K., Yang, N., Yang, R., Yang, X., Yang, X., Yang, Y., Yang, Y., Yang, Y., Yang, Z., Yang, Z., Yang, Z., Yao, H., Ye, D., Ye, W., Ye, Z., Yin, B., Yu, C., Yu, L., Yu, T., Yu, T., Yuan, E., Yuan, M., Yuan, X., Yue, Y., Zeng, W., Zha, D., Zhan, H., Zhang, D., Zhang, H., Zhang, J., Zhang, P., Zhang, Q., Zhang, R., Zhang, X., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Z., Zhao, C., Zhao, F., Zhao, J., Zhao, S., Zhao, X., Zhao, Y., Zhao, Z., Zheng, H., Zheng, R., Zheng, S., Zheng, T., Zhong, J., Zhong, L., Zhong, W., Zhou, M., Zhou, R., Zhou, X., Zhou, Z., Zhu, J., Zhu, L., Zhu, X., Zhu, Y., Zhu, Z., Zhuang, J., Zhuang, W., Zou, Y., and Zu, X. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., and Younis, O. G. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2025. URL https://arxiv.org/abs/2407.17032.

U.S. Bureau of Economic Analysis. National income and product accounts (NIPA). U.S. Department of Commerce, 2024. URL https://www.bea.gov/itable/national-gdp-and-personal-income. Interactive data tables, annual estimates. Accessed February 2025.

U.S. Bureau of Labor Statistics. Occupational employment and wage statistics (OEWS). U.S. Department of Labor, 2024. URL https://www.bls.gov/oes/. May 2024 estimates. Accessed February 2025.

Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., Chen, K., Liu, X., Wang, J., Ge, W., Fan, Y., Dang, K., Du, M., Ren, X., Men, R., Liu, D., Zhou, C., Zhou, J., and Lin, J. Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024. URL https://arxiv.org/abs/2409.12191.

Wang, X., Wang, B., Lu, D., Yang, J., Xie, T., Wang, J., Deng, J., Guo, X., Xu, Y., Wu, C. H., Shen, Z., Li, Z., Li, R., Li, X., Chen, J., Zheng, B., Li, P., Lei, F., Cao, R., Fu, Y., Shin, D., Shin, M., Hu, J., Wang, Y., Chen, J., Ye, Y., Zhang, D., Du, D., Hu, H., Chen, H., Zhou, Z., Yao, H., Chen, Z., Gu, Q., Wang, Y., Wang, H., Yang, D., Zhong, V., Sung, F., Charles, Y., Yang, Z., and Yu, T. Opencua: Open foundations for computer-use agents, 2025. URL https://arxiv.org/abs/2508.09123.

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., and Hajishirzi, H. Self-instruct: Aligning language models with self-generated instructions, 2023. URL https://arxiv.org/abs/2212.10560.

Wang, Z., Xu, C., Liu, B., Wang, Y., Han, S., Yao, Z., Yao, H., and He, Y. Agent world model: Infinity synthetic environments for agentic reinforcement learning, 2026a. URL https://arxiv.org/abs/2602.10090.

Wang, Z. Z., Vijayvargiya, S., Chen, A., Zhang, H., Arangarajan, V. A., Chen, J., Chen, V., Yang, D., Fried, D., and Neubig, G. How well does agent development reflect real-world work?, 2026b. URL https://arxiv.org/abs/2603.01203.

Wu, Z., Wu, Z., Xu, F., Wang, Y., Sun, Q., Jia, C., Cheng, K., Ding, Z., Chen, L., Liang, P. P., and Qiao, Y. Os-atlas: A foundation action model for generalist gui agents, 2024. URL https://arxiv.org/abs/2410.23218.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., and Yu, T. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., Lin, Q., and Jiang, D. Wizardlm: Empowering large pre-trained language models to follow complex instructions, 2025a. URL https://arxiv.org/abs/2304.12244.

Xu, F. F., Song, Y., Li, B., Tang, Y., Jain, K., Bao, M., Wang, Z. Z., Zhou, X., Guo, Z., Cao, M., Yang, M., Lu, H. Y.,

Martin, A., Su, Z., Maben, L., Mehta, R., Chi, W., Jang, L., Xie, Y., Zhou, S., and Neubig, G. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. URL https://neurips.cc/virtual/2025/poster/121705. NeurIPS 2025 Datasets and Benchmarks Track.

Xu, Y., Lu, D., Shen, Z., Wang, J., Wang, Z., Mao, Y., Xiong, C., and Yu, T. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*, 2025c. URL https://arxiv.org/abs/2412.09605.

Xu, Z., Jiang, F., Niu, L., Lin, B. Y., and Poovendran, R. Stronger models are not always stronger teachers for instruction tuning. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4392–4405, Albuquerque, New Mexico, April 2025d. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.224. URL https://aclanthology.org/2025.naacl-long.224/.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yang, Y., Sun, F.-Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., Callison-Burch, C., Yatskar, M., Kembhavi, A., and Clark, C. Holodeck: Language guided generation of 3d embodied ai environments, 2024. URL https://arxiv.org/abs/2312.09067.

Zhang, J., Peng, Y., Kong, F., Yang, C., Wu, Y., Yu, Z., Xiang, J., Ruan, J., Wang, J., Song, M., et al. Autoenv: Automated environments for measuring cross-environment agent learning. *arXiv preprint arXiv:2511.19304*, 2025.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=oKn9c6ytLx. ICLR 2024; arXiv:2307.13854.

Zhou, Y., Yang, Q., Lin, K., Bai, M., Zhou, X., Wang, Y.-X., Levine, S., and Li, E. Proposer-agent-evaluator(pae): Autonomous skill discovery for foundation model internet agents, 2024b. URL https://arxiv.org/abs/2412.13194.

Zhu, Y., Gandhi, A., and Neubig, G. Training versatile coding agents in synthetic environments. *arXiv preprint arXiv:2512.12216*, 2025.

Zhuge, M., Zhao, C., Ashley, D., Wang, W., Khizbullin, D., Xiong, Y., Liu, Z., Chang, E., Krishnamoorthi, R., Tian, Y., Shi, Y., Chandra, V., and Schmidhuber, J. Agent-as-a-judge: Evaluate agents with agents, 2024. URL https://arxiv.org/abs/2410.10934.

Appendix Table of Contents

A. Limitations.

The GDP-grounded selection provides a ranking, not a precise dollar attribution; share factors in Eq. 2.1 are LLM-estimated with web search, and may be imperfect. The sandboxable constraint (free-tier, self-hostable, no specialized hardware) excludes much commercial software; whether free-substitute performance predicts commercial-counterpart performance is open. We manually verified every environment launches and every `CUA-World`-Long task loads correctly, but did not solve all tasks end-to-end, so solvability is not guaranteed. VLM verifiers are imperfect and may be adversarially exploitable; manually-vetted programmatic verifiers would complement our approach. Finally, we acknowledge that computer-use agents may pose risks if deployed autonomously; we do not release a model exceeding existing frontier capabilities, and all data is public or synthetic.

B. Ethics Statement

We acknowledge that computer-use agents may pose risks if deployed autonomously. While this work introduces methods for environment creation and test-time auditing, it does not train or release a model that exceeds existing frontier capabilities. All software used is freely available, and all datasets were obtained from public sources or synthetically generated.

LLM and Agent Usage Disclosure

This paper is substantively about the use of LLM agents to construct CUA environments and tasks, and we document agent usage here in line with the workshop policy. LLM/agent usage falls into three categories.

As research artifacts. Claude Opus 4.5/4.6 (via Claude Code) served as the creation, audit, task-proposer, and memory-summarization agents in the `Gym-Anything` pipeline; Gemini 3 Pro served as the task amplifier; Gemini 3 Flash served as the checklist VLM verifier, the TTA agent, and the starting-state filter. Models evaluated on `CUA-World` include Gemini 3 Flash, GPT-5.4, Claude Sonnet 4.6, Kimi-K 2.5, and Qwen3-VL-2B,4B. All artifacts produced by these agents (setup scripts, audits, tasks, evaluations) were spot-checked by the authors; for `CUA-World`-Long we manually verified every task. Model versions, harnesses, budgets, and spot-check procedures are listed in Appendix P.1.

As tools. We used an LLM with web-search access to estimate per-software GDP share factors (s_{cat}, s_{sw}) in Eq. 2.1, and to enumerate the initial catalog of $\sim$16,600 software applications. Both outputs were validated via web-grounded verification and manual spot checks; outputs are used as ordinal rankings, not as precise dollar attributions (see Limitations).

For writing assistance. Coding assistants (Claude Code) were used to help write and edit portions of the draft and to render figures and tables. All final text, claims, and experimental decisions are the authors'; the authors verified factual content and experimental results against the underlying data and artifacts.

C. Detailed Benchmark Comparison

D. Problem Setup and Library Details

Environment. We refer to an environment $\mathcal{E}$ as one or more interactive software applications with a specified initial state of the filesystem and processes running inside an operating system. The agent interacts with the environment through keyboard and mouse actions.

Task. A task $\mathcal{T} = (\mathcal{E}_{s_0}, p, V)$ consists of an environment $\mathcal{E}$ with initial state s_0, a natural-language instruction p, and a verification function V that maps the agent's trajectory to a score.

Interaction. At each step t, the agent receives an observation o_t (e.g., a screen capture) and executes mouse/keyboard actions a_t, after which the environment returns a new observation o_{t+1}. An episode proceeds by resetting the environment to s_0, letting the agent interact for up to T steps, and the final score is determined by V.

Library abstraction. Constructing computer-use environments across hundreds of diverse software applications requires a unified framework that works across operating systems, application types, and compute backends without per-environment

Table 3. **Comparison of `CUA-World` with datasets and environments for computer-use agents.** ✓ yes; × no; ✓* partial or with caveats; — not applicable. * Benchmark allows or requires >100 agent steps per task. * Offline human demonstrations only (not interactive verified trajectories). § Number of 2018 SOC (Standard Occupational Classification) major occupation groups (out of 22 civilian groups) whose workers would routinely use the benchmark's applications; counted conservatively such that a group is included only if tasks directly simulate work in that occupation.

		Environment		Scale		Task Properties		Infrastructure	
Benchmark	**Agent**	**Interactive**	**Platform**	**# SW**	**# Tasks**	**Long-Horizon***	**Econ. Cov.§**	**Auto-Create**	**Train Split**
Static Datasets									
Mind2Web (Deng et al., 2023)	Web	×	Web	137	2,350	—	7/22	×	✓*
AITW (Rawles et al., 2023)	CUA	×	Android	357+	715K	—	4/22	×	✓*
OmniACT (Kapoor et al., 2024)	CUA	×	Lin / Win / macOS / Web	65	9,802	—	6/22	×	✓*
GDPval (Patwardhan et al., 2025)	LLM	✓	—	—	1,320	✓	13/22	×	×
Interactive Benchmarks									
WebArena (Zhou et al., 2024a)	Web	✓	Web	6	812	×	5/22	×	×
VisualWebArena (Koh et al., 2024)	Web	✓	Web	3	910	×	1/22	×	×
WorkArena++ (Boisvert et al., 2024)	Web	✓	Web	1	682†	✓	3/22	×	✓*
OSWorld (Xie et al., 2024)	CUA	✓	Linux / Win	9	369	×	3/22	×	×
WindowsAgentArena (Bonatti et al., 2025)	CUA	✓	Windows	11	154	×	3/22	×	×
TheAgentCompany (Xu et al., 2025b)	CUA	✓	Linux / Web	5	175	×	4/22	×	×
ProgrammingWithPixels (Aggarwal & Welleck, 2025)	CUA	✓	Linux	1	5400	×	1/22	×	×
`CUA-World` (Ours)	**CUA**	✓	**Lin / Win / Android / Web**	**200+**	**10,000+**	✓	**22/22**	✓	✓

engineering. Previous works have primarily constructed computer-use environments manually by interacting with an actual operating system, and then taken VM snapshots to be reused later (Xie et al., 2024). However, snapshots cannot be inspected, version-controlled, or partially reused across tasks, and modifying anything requires repeating the manual setup, therefore limiting modularity, reproducibility, and scalability.

To handle these challenges, we construct the `Gym-Anything` Library. In `Gym-Anything`, each environment is defined by a simple specification: three sequential setup scripts and a declarative configuration file. The scripts progress from general to task-specific: *install* installs the software and its dependencies, *configure* sets it up with realistic data and settings, and *task setup* configures the specific starting state for a given task. This separation ensures that multiple tasks for the same software share the same install and configure scripts, varying only the task-specific setup. For example, creating a LibreOffice Calc environment requires only an install script (e.g., `apt-get install libreoffice`), a configure script that downloads a sample spreadsheet, and a config file specifying the OS image and resource limits; the library handles container orchestration, display forwarding, and checkpoint management automatically. This design reduces environment creation to a scripting task: users and AI agents create new environments by writing setup scripts and key-value configurations, then interact with any environment through a standard gymnasium-style API (Towers et al., 2025; Brockman et al., 2016) that provides a unified observation space (e.g., screenshots) and action space (e.g., keyboard and mouse inputs), with the library handling display forwarding and input translation across operating systems. This specification is simple enough that an LLM agent can author environments autonomously (§2.3), yet expressive enough to capture complex, production-grade software configured with realistic data, ranging from desktop image editors to multi-container enterprise systems. Behind this simple interface, the library manages the complexity of running environments across three operating systems (Linux, Windows, Android) and multiple compute backends (such as docker and apptainer for rootless systems such as slurm). The staged design further enables caching at each stage boundary, so creating new tasks only requires re-running the task-specific setup. Combined with network-process-file isolation, this enables massive parallelization; in our experiments we run 400+ concurrent environments across 1,600 CPUs.

E. Privileged Information Examples

Table 4 shows representative examples of privileged information and the checklist items it enables across scientific analysis, clinical reporting, business operations, and clinical decision support.

Table 4. **Full-text examples of task descriptions, privileged information, and VLM checklist items.**; color only highlights the most important privileged information.

Software	Task Description	Privileged Information	VLM Checklist
AstroImageJ	Analyze the WASP-12 (RA: 06:30:32.79, Dec: +29:40:20.4) astronomical image sequence from January 5-6, 2016, to identify evidence of a planetary transit. If a transit is detected, determine the transit depth, mid-transit time (BJD_TDB or JD), and transit duration in hours. Using a host star radius of 1.599 solar radii, estimate the planet's radius in Jupiter radii. Save your findings and uncertainties to `~/Documents/transit_analysis.txt`.	Target: WASP-12b. Expected Transit Depth: ~1.4% (0.014 relative flux). Expected Duration: ~2.7 hours. Expected Planet Radius: ~1.79 Jupiter radii (calculation: sqrt(0.014) * 1.599 R_sun * conversion factors). The dataset is real ground-based imagery from Jan 5-6, 2016, so the light curve will have noise but the transit dip should be clearly visible.	1. The agent loads the sequence of astronomical images into AstroImageJ. 2. The agent selects the target star (WASP-12) and appropriate comparison stars for differential photometry. 3. The agent generates and displays a light curve plot showing the star's flux over time. 4. The agent fits a model or trend line to the data to characterize the transit. 5. The agent reports the measured transit depth, mid-transit time, and duration. 6. The agent calculates and reports the planet's radius in Jupiter radii.
Apache OpenOffice Writer	You are a Clinical Research Associate (CRA) performing an Interim Monitoring Visit (IMV) for Protocol ZN-994 at Site 142. Using the visit data in `/home/ga/Documents/visit_notes.json`, create a formal IMV Report in Apache OpenOffice Writer saved as `/home/ga/Documents/IMV_Report_Site_142.odt`. The report must include a document header with the Protocol and Site numbers, page numbers in the footer, and sections using 'Heading 1' style for 'Visit Details', 'Subject Enrollment', 'Protocol Deviations', and 'Action Items'. [. . .]	Protocol Number: ZN-994. Site Number: 142. Enrollment counts from the input file: Screened=18, Randomized=14, Completed=3, Discontinued=2. The calculated 'Active' count must be exactly 9 (14 - 3 - 2 = 9). The Action Items table contains items AI-02 and AI-03 which have an 'Open' status and must be formatted with yellow highlight or red text. The four required sections are 'Visit Details', 'Subject Enrollment', 'Protocol Deviations', and 'Action Items'.	1. Verify the document has a header with Protocol and Site numbers, and a footer with page numbers. 2. Verify the four required sections are present and use the 'Heading 1' style. 3. Verify the Subject Enrollment section contains a table with the correct base counts. 4. Verify the 'Active' count in the Subject Enrollment table is correctly calculated. 5. Verify the Protocol Deviations list and Action Items table structure. 6. Verify that Action Items with an 'Open' status are conditionally formatted.
Aerobridge	Calculate the total flight duration in minutes for all flight plans belonging to the operator 'SkyHigh Surveyors' that occurred in October 2023. Save the total number of minutes to `/home/ga/Documents/utilization_report.txt`.	The correct total flight duration for 'SkyHigh Surveyors' in October 2023 is 135 minutes (acceptable tolerance: 133-137 minutes). This is calculated from two flights: Flight 1 on Oct 5 (10:00-10:45, 45 mins) and Flight 2 on Oct 12 (14:00-15:30, 90 mins). Distractor flights (wrong operator or wrong month) must be excluded.	1. Locate flight plans belonging to the operator 'SkyHigh Surveyors'. 2. Identify the flight plans that occurred in October 2023. 3. View the details or times of the relevant flights to calculate duration. 4. Open or create the utilization report file. 5. Save the correct total flight duration to the report file.
Liverpool Cancer iChart	Using the Liverpool Cancer iChart Archive app, determine the drug-drug interaction between Dabrafenib and Ketoconazole (located in the Antifungal agents category). Leave the application on the screen displaying the interaction result.	Dabrafenib is a cancer drug (BRAF inhibitor) and Ketoconazole is an antifungal agent. The interaction between them is clinically significant due to potent CYP3A4 inhibition by ketoconazole, which increases dabrafenib AUC by 71% and Cmax by 33%. The VLM should look for an interaction result indicating a severe/red warning or mentioning CYP3A4 inhibition.	1. The Liverpool Cancer iChart Archive application is opened. 2. Dabrafenib is selected as the cancer drug. 3. The Antifungal agents category is accessed. 4. Ketoconazole is selected as the comedication. 5. The interaction result between Dabrafenib and Ketoconazole is displayed on the screen.

F. GDP-Grounded Software Selection: Full Pipeline

This appendix provides the complete technical details for the software selection pipeline summarized in Section 2.1.

F.1. Phase 1: Occupation GDP Calculation

We assign a GDP value to each of 894 O*NET occupations via a three-step scaling procedure:

1. **Wage bill**: For each SOC-2018 occupation, compute employment $\times$ mean_wage from BLS OEWS (May 2024).
2. **Labor compensation**: Scale wage bills by the national ratio $\frac{\text{Total Compensation}}{\text{Total Wages}}$ from BEA accounts.
3. **Total GDP**: Scale labor compensation by $\frac{\text{National GDP}}{\text{National Compensation}}$.

Output: `us_gdp_by_occupation_USD.csv` with columns: `onetsoc`, `soc2018`, `occupation_title`, `employment`, `mean_wage`, `wage_bill`, `gdp_labor`, `gdp_total`.

F.2. Phase 2: Software Discovery

Category extraction. Occupations are shuffled and batched into groups of 10. For each batch, an LLM (GPT-5) is prompted: *"What software categories does each occupation use?"* with no fixed taxonomy—the model discovers categories freely. This yields 5,584 occupation→category pairs across 894 occupations.

Category deduplication. Similar categories are clustered via exact normalized-name matching and fuzzy string similarity (token sort ratio $\geq 92\%$). The most frequent label per cluster is selected as the canonical name. An LLM (Gemini 3 Flash) adjudicates ambiguous pairs.

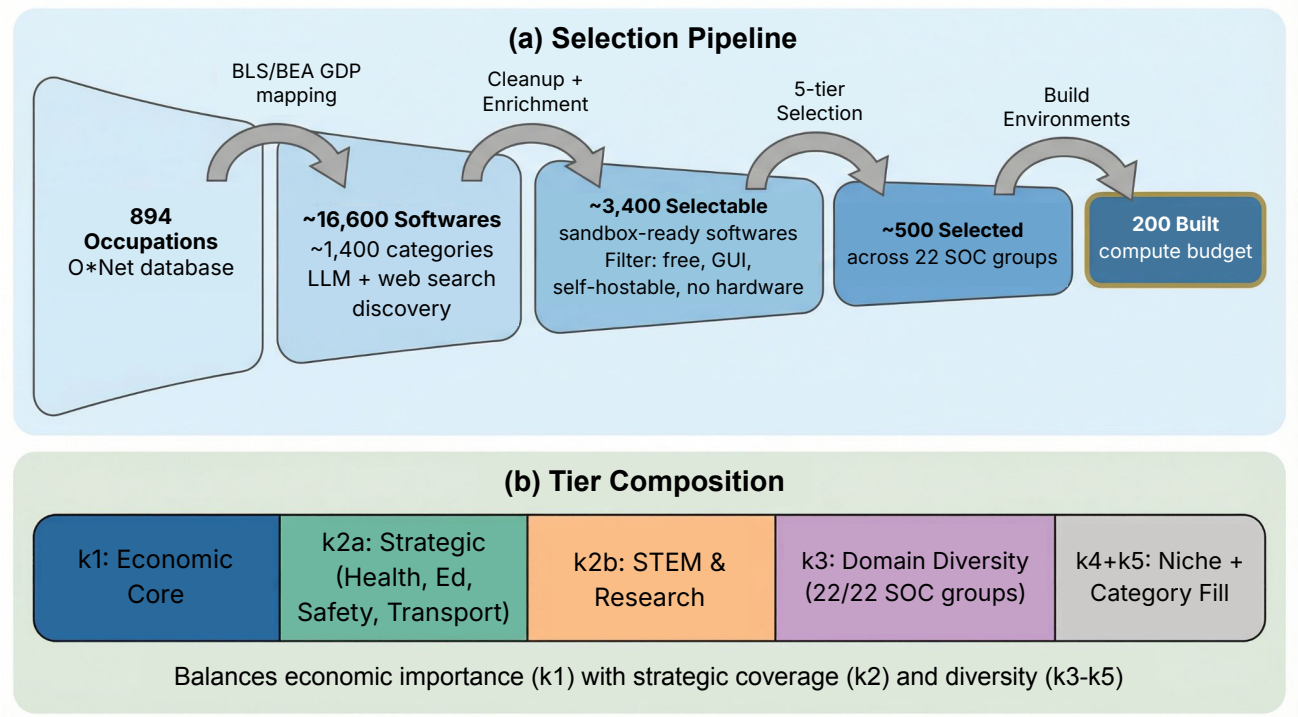


Figure 5. **GDP-grounded software selection pipeline.** Starting from U.S. occupational data, we estimate per-software GDP, filter to sandboxable candidates, and apply tiered selection to yield 200 software applications.

Product enumeration. For each unique category, an LLM enumerates widely-used software products (name, OS support, aliases). Products are deduplicated within each category via fuzzy matching, producing a catalog of ∼16,000 products across ∼1,400 categories.

F.3. Phase 3: Catalog Cleanup

Three cleaning passes remove noise from the LLM-generated catalog:

Product–category validation. For each category, an LLM verifies which products genuinely belong; mismatches are removed (e.g., "Photoshop" in "Spreadsheets").

Product existence verification. All products are verified via an LLM with Google Search grounding. Products that do not correspond to real, currently-available software (LLM hallucinations) are removed.

All LLM calls are cached in JSONL format for reproducibility.

F.4. Phase 4: Catalog Enrichment

Three parallel enrichment passes classify each product along:

- **Pricing**: `free` | `paid` | `trial` | `freemium`; and `is_open_source`.
- **Interface**: `gui` | `cli` | `both`.
- **Trainability**: `sandbox_ready` (install freely) | `self_hostable` (deploy in Docker/VM) | `free_tier` (cloud with free account) | `restricted` (paid license or org credentials required).

F.5. Phase 5: GDP Attribution

For each occupation, an LLM receives occupation metadata (O*NET code, computer-use importance/level scores), applicable software categories, and available products. It returns a structured allocation:

$$\text{GDP}_{\text{product}} = \sum_{\text{occ}} \text{GDP}_{\text{occ}} \times p_{\text{computer}} \times s_{\text{category}} \times s_{\text{product}}$$

with constraints: $p_{\text{computer}} \in [0, 1]$ bounded by O*NET scores, category shares sum to $\sim$1.0, and product shares within each category sum to 1.0.

The product GDP values are aggregated across all occupations. Top products by GDP: Microsoft Excel, Microsoft Word, Google Chrome, Microsoft Outlook, Visual Studio Code. Important: These are estimates generated by our pipeline, and should not be cited as source of ranking in future work.

F.6. Phase 6: Practical Access-Barrier Evaluation

Pricing and interface filters are necessary but not sufficient. A product can be free and GUI-based yet still require external account creation (Slack, Zoom), organizational credentials (Epic, Slate), or specialized hardware (AndroidAPS).

We batch all $\sim$16,600 products through an LLM (Gemini 3 Pro) with Google Search grounding, evaluating:

- Does the product require an external account? (`no` | `free_optional` | `free_required` | `org_required`)
- Does it require organizational/institutional credentials?
- Does it require specialized physical hardware?
- Overall: is it trainable in a sandbox?

We evaluate the *most permissive* version/mode of each product: if NinjaTrader's free simulation mode works without login, the product passes.

Results: 8,013 products (48%) are trainable; 8,591 (52%) are not (4,651 require organizational accounts, 3,279 require free accounts, $\sim$661 require hardware).

F.7. Phase 7: Tiered Selection

A product is **selectable** if it satisfies all of: (1) runs on Windows, Linux, or Android; (2) not paid-only or trial-only; (3) not CLI-only; (4) sandbox-ready or self-hostable; (5) passes the access-barrier evaluation from Phase 6.

When a non-selectable product would otherwise be chosen (e.g., Bloomberg Terminal at $79.5B GDP), the pipeline substitutes the closest selectable alternative from the same software category, ranked by an LLM. The substitute inherits the original's economic slot; the original is marked as "covered."

Selection proceeds across five tiers:

Tier	Budget	Strategy
k1 (Economic Core)	100	Highest GDP products overall
k2.1 (Strategic)	100	Healthcare, Education, Protective Services, Transportation—20 per domain
k2.2 (STEM)	100	Architecture/Engineering, Computer/Math, Life/Physical/-Social Science
k3 (Domain Diversity)	116	Round-robin across all 22 SOC major groups ($\sim$5 per group)
k4 (Niche)	44	Products unique to single occupations or domains
k5 (Category Fill)	40	Uncovered software categories, largest GDP first

Table 5. Tiered selection budget. Each tier iterates occupations by GDP (descending) and applies substitution for non-selectable products.

Output: $\sim$500 selected products covering all 22 SOC major groups. We build environments for 200 based on compute budget. Important Note: Due to bug in our software selection code, we had initially selected 53 environments that would later not come in the 200 selected. However, due to compute constraints, we decided to keep them. However, all of those would have been included in the full 500, which we plan to release sometime in future.

F.8. Pipeline Statistics

Metric	Value
Occupations covered	894
Software categories	∼1,400
Products in catalog	∼16,600
Products passing all filters	∼3,400
Products selected	∼488
Substitutions made	∼429
SOC domains covered	22/22

Table 6. GDP-grounded software selection pipeline summary statistics.

G. Gym-Anything Framework: Technical Details

This appendix provides engineering details for the Gym-Anything framework summarized in Section 2.2. A central design principle of the framework is *modularity*: the specification schema, runner interface, and verification system are all designed so that new observation modalities, compute backends, operating systems, and verification strategies can be added without modifying the core framework.

G.1. Environment Specification Schema

Each environment is defined by an `env.json` file with the following sections:

Metadata. `id`, `version`, `description`, `category`, `tags`, `authors`.

Runtime. `base` (preset image: `ubuntu-gnome-systemd`, `windows-11`, `android-14`), `image` or `dockerfile` (custom container), `resources` (CPU cores, memory GB, GPU count, network access), `mounts` (bind-mount scripts, data, config as read-only or read-write).

Interfaces. `observation`: list of modalities, each with type, resolution, and frame rate. Currently supported modalities include `rgb_screen`, `audio_waveform`, and `ui_tree`, with the schema designed to accommodate additional modalities as needed. `action`: list of types. Currently supported types include `mouse`, `keyboard`, `voice`, and `api_call`.

User accounts. Per-account specification of username, password, UID/GID, and permissions (sudo, network access, environment variables, home directory settings). This enables realistic enterprise scenarios with privilege separation.

Security. Systemd, cgroups, capability dropping, seccomp profiles, network isolation toggles.

G.2. Multi-Runner Architecture

All execution backends implement a common `BaseRunner` interface, so new compute backends can be added by implementing the same abstract methods. The framework currently ships with the following runners:

Runner selection is automatic: the framework checks for Docker availability, then Apptainer, then falls back to the local runner. Users may override via the `GYM_ANYTHING_RUNNER` environment variable.

G.3. Progressive Checkpointing

The four setup stages (install → configure → task setup → export) are checkpointed at three levels:

- **Post-install checkpoint**: saves disk state after software installation. Loading skips the install stage. Shared across all tasks for the same environment.
- **Post-configure checkpoint**: saves state after data/service configuration. Loading skips install and configure. Also shared across tasks.
- **Post-task-setup checkpoint**: saves state after per-task initialization. Task-specific; enables instant startup for repeated evaluation of the same task.

Runner	Use case
`DockerRunner`	Single-machine development; requires Docker daemon
`QemuApptainerRunner`	HPC/SLURM clusters; runs QEMU VMs inside rootless Apptainer containers
`QemuNativeRunner`	Bare-metal Linux or macOS; supports Apple Silicon via HVF
`AVFRunner`	macOS on Apple Silicon; uses Apple Virtualization Framework with Rosetta 2
`AVDApptainerRunner`	Android apps; wraps Android emulator in Apptainer
`AVDNativeRunner`	Android apps; runs emulator directly on host (macOS HVF or Linux KVM)
`ApptainerDirectRunner`	GPU-enabled workloads; direct Apptainer with `-nv` flag
`LocalRunner`	Lightweight testing stub

Table 7. Execution backends. The same `env.json` runs on all runners without modification. New backends can be added by implementing the `BaseRunner` interface.

Disk-state checkpoints. Docker: `docker commit` captures filesystem (processes restart via systemd on next boot). QEMU: `qemu-img convert` saves a QCOW2 snapshot.

Full-state snapshots (SaveVM). QEMU additionally supports `savevm`, which captures the entire VM memory, CPU registers, and running processes. Restoring from a savevm snapshot is near-instantaneous ($\sim$3s), preserving open windows, running services, and GUI state—compared to 2–5 minutes for a disk-state checkpoint that requires rebooting.

Copy-on-write parallelization. Multiple concurrent instances share the same base checkpoint via QCOW2 overlay files. Each instance writes only its delta, enabling 400+ parallel environments with modest disk overhead.

G.4. Platform-Specific Patterns

Linux desktop environments. The majority of environments use an Ubuntu GNOME base with systemd. GUI automation uses `xdotool` for mouse/keyboard injection and X11 accessibility for UI tree capture. Multi-service web applications (ERPs, CRMs, LMS platforms) run Docker Compose stacks inside the QEMU VM.

Windows environments. SSH runs in Session 0 (no GUI access). GUI applications are launched via `schtasks /IT` with batch files. Interactive automation uses a PyAutoGUI TCP server (port 5555) running in the GUI session, since Win32 API calls (`SetCursorPos`, `mouse_event`) do not reliably reach all applications. Registry modifications suppress first-run dialogs and license prompts.

Android environments. Android Virtual Devices run inside Apptainer via the `AVDApptainerRunner`. Interaction uses ADB for input injection (`adb input tap`, `adb input text`) and screenshot capture (`adb exec-out screencap`). APK installation copies to `/data/local/tmp/` before invoking `pm install` to satisfy SELinux constraints.

G.5. Verification System

Verifiers are decoupled from the framework: each is a standalone Python file in the task directory, loaded via `importlib` at evaluation time. The framework currently supports three verification modes:

1. **Program**: a Python function receives the trajectory (screenshots, action log), environment utilities (`exec_capture`, `copy_from_env`, `query_vlm`), and task metadata. Returns `{passed, score, feedback}`. Programmatic verifiers can also call a VLM internally (e.g., for checklist-based evaluation), combining the flexibility of code with visual grounding.

2. **Image match**: SSIM comparison between the final screenshot and a reference image, with a configurable threshold.

3. **Multi**: cascades program verification first, falling back to image match.

Custom verification strategies can be added by writing a new Python file following the same interface.

G.6. Episode Artifacts

Each episode produces a structured artifact directory:

- `traj.jsonl`: timestamped log of every reset, action, and observation event.
- `frame_00000.png`, ...: per-step screenshots.
- `video.mp4`: FFmpeg-encoded recording of the full episode.
- `summary.json`: episode metadata, verifier result, and reward.
- Setup stage logs (`*.log`): stdout/stderr from each setup script.

These artifacts serve dual purposes: evaluation (verifier input) and training data collection (trajectory distillation, Section 4.1).

G.7. Distributed Execution and Tooling

Remote execution. For large-scale evaluation across multiple machines, the framework provides a master-worker architecture. Worker nodes expose a REST API that manages local environments, while a master server handles load-balanced routing with sticky sessions (mapping each environment to a specific worker). A `RemoteGymEnv` client provides the same API as the local `GymAnythingEnv`, making distributed execution transparent to the caller.

Trajectory viewer. A built-in web dashboard allows browsing and replaying recorded episodes, including per-step screenshots, action logs, and verifier outputs. This supports both debugging during development and qualitative analysis of agent behavior.

G.8. Usage Example

To illustrate the simplicity of the framework, we show how an environment can be launched and interacted with via the command line and Python API.

Command line. The `gym-anything` CLI provides commands for running environments interactively, evaluating agents on benchmark splits, listing available environments, validating specifications, managing cached checkpoints, and checking system prerequisites. For example, to launch an environment interactively with a VNC viewer:

```
# List available environments and tasks
gym-anything list

# Launch an environment interactively
gym-anything run libreoffice_calc -task budget_analysis -i -open-vnc

# Evaluate an agent on a benchmark split
gym-anything benchmark libreoffice_calc -agent Gemini3Agent \
-model gemini-3-flash -split test
```

Python API. Programmatically, the framework exposes a standard Gymnasium-style interface. The same environment specification runs identically across all compute backends:

```
from gym_anything import make

env = make("envs/libreoffice_calc/env.json",
"envs/libreoffice_calc/tasks/budget_analysis/task.json")

obs = env.reset(use_cache=True, cache_level="post_start")

actions = [{"action": "left_click", "coordinate": [340, 215]},
{"action": "type", "text": "=SUM(B2:B10)"},
{"action": "key", "key": "Return"}]

obs, reward, done, info = env.step(actions)

env.close()  # runs verifier, writes trajectory artifacts
```

The `reset()` call handles container orchestration, display forwarding, and checkpoint restoration. The `step()` call injects actions and returns the next observation (screenshot). On `close()`, the framework runs the task verifier and writes all episode artifacts (trajectory log, per-step screenshots, video, and verification results) to a structured directory.

H. Prompts

This appendix documents the prompts used across the `Gym-Anything` pipeline. We provide an overview of the Creation Agent prompt (§H.1), the three-phase Privileged Information Audit prompt (§H.2), the VLM Checklist Verifier prompts (§H.3), and the Contamination Filtering prompt (§H.5).

H.1. Creation Agent Prompt (Overview)

The Creation Agent (Agent_C, §2.3) receives an ∼800-line prompt that guides it through seven phases. Listing 1 shows an abridged overview of the phase structure and key instructions. The full prompt covers framework internals, realistic data sourcing strategies, interactive testing workflows, and documentation requirements. Critically, the prompt emphasizes that all data must be real (downloaded from public sources), not synthetic or handwritten.

Listing 1: Creation Agent prompt overview (abridged from ∼800 lines).

```
# Environment Creation Workflow for Gym-Anything
# (Abridged overview -- full prompt is ~800 lines)

## Phase 1: Understand the Framework
Read core files (api.py, env.py, specs.py, runners/) to understand the
lifecycle: from_config() -> env.reset() -> [agent interaction] -> env.close()
Key rules: hooks run as root, DISPLAY=:1 for GUI, mounts read-only by default.

## Phase 2: Research the Target Application
Web-search for installation guides, identify dependencies, determine
installation method. Answer: desktop vs web app? what services needed?
how is data stored? first-run wizard? network access?

## Phase 3: Study Existing Environments
Read env_creation_notes/ (cross-cutting patterns, Windows patterns).
Study similar environments in examples/ directory.

## Phase 4: Create Implementation Plan
Directory structure: env.json, scripts/{install,setup}.sh, tasks/*/
Hook responsibilities: pre_start (install), post_start (configure),
pre_task (task-specific setup with REAL data -- no fake/synthetic data).

## Phase 5: Write Environment Files
Create env.json (base image, resources, mounts, hooks, user accounts).
Write install script (apt, wget, docker), setup script (service polling,
app launch, config), task files (task.json, setup_task.sh, verifier.py).
CRITICAL: All data must be real (public datasets, official samples).

## Phase 6: Interactive Testing
Start VM, SSH in, use screenshot-based UI grounding to verify setup.
Loop: take screenshot -> analyze -> perform action -> repeat.
Verify: app visible, correct state, real data loaded, task completable.

## Phase 7: Final Testing & Documentation
Clean test without cache. Verify full checklist. Create evidence_docs/
with screenshots and log snippets. Document learnings in shared memory.
```

H.2. Privileged Information Audit Prompts

The PI Audit pipeline (§2.5) operates in three phases. Phase 1 analyzes task source files without web access to identify data provenance and metadata claims. Phase 2 uses an LLM with Google Search grounding to verify claims against external sources. Phase 3 synthesizes the results into a validated PI report, enforcing the rule that unverified information must never appear in the final summary.

Phase 1: File Analysis.

Listing 2: PI Audit Phase 1: Data provenance analysis prompt.

```
You are a data provenance auditor for AI benchmarking tasks.
Your job is to analyze task files and identify:
1. What dataset is used (name, source URL, specific case/patient ID)
2. What claims does task.json metadata make about expected values
   (numerical thresholds, counts, measurements)
3. Which values are hardcoded in scripts vs dynamically computed at runtime
4. Which values in metadata could be verified via web search
5. Whether the data is real (downloaded from a public dataset)
   or synthetic (generated by scripts)

Mark all claims as verifiable_via_web=true and provide a
suggested_search_query for each. Your goal is to find the correct
value for every claim, not just confirm what task.json says.

Analyze these files for task "{task_id}" and produce a structured
JSON response:

{all_files}

Respond with ONLY a JSON object:
{
    "dataset_name": "name of dataset or null if unclear",
    "dataset_url": "download URL found in scripts or null",
    "case_id": "specific case/patient/file ID or null",
    "data_is_synthetic": true/false,
    "synthetic_reason": "why you think data is synthetic, or null",
    "claims": [
        {
            "key": "metadata key name",
            "value": "value claimed in task.json metadata",
            "source": "hardcoded/computed/script-derived",
            "verifiable_via_web": true/false,
            "suggested_search_query": "web search query or null"
        }
    ],
    "general_task_context": "1-2 sentence description",
    "data_provenance_summary": "how data gets from source to the VM"
}
```

Phase 2: Web-Grounded Verification.

Listing 3: PI Audit Phase 2: Web search verification prompt (uses Google Search grounding).

```
You are verifying claims about a dataset used in an AI benchmarking
task.

Task: {task_id}
Dataset: {dataset_name}
Case/Patient: {case_id}
Data source URL: {dataset_url}

Here are the claims to verify using web search:
{claims_text}

Your goal is to find the correct value for every claim -- either
confirm the claimed value or find the true value. Do not settle for
"unverified". Search extensively: try the data source page, dataset
documentation, academic papers, archive databases, file format
specifications, instrument documentation.

Look for:
- Official dataset documentation pages
- Published ground truth spreadsheets or CSV files
- Academic papers describing the dataset
- Dataset README files on Kaggle, TCIA, Zenodo
- Instrument/software documentation
- Archive databases (MAST, TCIA, PhysioNet)

Respond with ONLY a JSON object:
{
    "verification_results": [
        {
```

```
            "key": "claim key",
            "claimed_value": "what task.json claims",
            "verified_value": "what web sources say, or null",
            "status": "verified|contradicted|unverified",
            "source": "URL or description",
            "confidence": "high|medium|low",
            "details": "explanation of what you found"
        }
    ],
    "web_search_summary": "brief summary of search results"
}
```

Phase 3: Synthesis.

Listing 4: PI Audit Phase 3: Synthesis into validated privileged information.

```
You are producing a final validated privileged information (PI)
report for a benchmarking task.

CRITICAL RULE: Only include information that is VERIFIED in the
privileged_info_summary. It is MUCH BETTER to have no PI than fake
PI. If a value cannot be verified, mark it as "unverified" and do
NOT include it in the summary text.

Task: {task_id}

File analysis results:
{analysis}

Web verification results:
{verification}

Produce a final validated_pi.json:
{
    "task_id": "{task_id}",
    "dataset": "dataset name or null",
    "case_id": "specific case ID or null",
    "data_is_synthetic": true/false,
    "pi_items": [
        {
            "key": "item name",
            "metadata_value": "what task.json claims",
            "verified_value": "what we verified, or null",
            "source": "script analysis / web search URL / etc",
            "status": "verified|unverified|contradicted"
        }
    ],
    "privileged_info_summary": "ONLY verified facts. This text
        will be given to a VLM verifier.",
    "pi_confidence": "high|medium|low|none"
}
```

H.3. VLM Checklist Verifier Prompts

The checklist-based VLM verification system (§2.5) operates in two stages. First, an LLM generates a structured checklist from the task description and validated PI (Listing 5). Second, a VLM scores agent trajectories against this checklist by examining sampled screenshots (Listing 6). The scoring system distinguishes between task completion failures (scored via partial credit) and integrity violations (binary pass/fail for cheating detection).

Checklist Generation.

Listing 5: VLM Verifier: Checklist generation prompt.

```
You are generating a verification checklist for an AI agent
benchmark task. This checklist will be used by a VLM
(vision-language model) to score agent trajectories by examining
screenshots.
```

```
Task ID: {task_id}
Task Description: {task_desc}
Metadata keys: {metadata_keys}
{scoring_text}
Privileged Information: {pi_summary}
{pi_items_text}

Generate a checklist with two sections:

1. **task_completion** (5-8 items, points must sum to exactly 100):
   Each item represents a sub-goal or evidence of progress. Items
   should be ordered from earliest to latest.
   - CRITICAL: ONLY include items that are explicitly required by
     the task description. Do NOT add extra steps.
   - Assign more points to harder items
   - Each item must be visually verifiable from screenshots
   - Include what visual evidence the VLM should look for.

2. **integrity** (3-4 items):
   Each item checks for cheating/shortcuts. Common checks:
   - Agent used the GUI, not terminal commands
   - Agent interacted with the actual application
   - Agent didn't copy-paste expected answers
   - Results come from genuine software interaction

Also produce a "privileged_info_for_vlm" field: a concise text
with ONLY verified facts that helps the VLM judge correctness.

Respond with ONLY a JSON object:
{
    "task_id": "{task_id}",
    "task_completion": [
        {
            "id": "short_snake_case_id",
            "description": "What this item checks",
            "points": 20,
            "visual_evidence": "What the VLM should look for"
        }
    ],
    "integrity": [
        {
            "id": "short_snake_case_id",
            "description": "What this integrity check verifies",
            "visual_evidence": "What the VLM should look for"
        }
    ],
    "privileged_info_for_vlm": "Verified facts for the VLM"
}
```

Trajectory Scoring.

Listing 6: VLM Verifier: Trajectory scoring prompt.

```
[System Message]
You are an expert evaluator scoring AI agent trajectories on
computer-use benchmark tasks. You will examine trajectory screenshots
and score the agent's work against a structured checklist.

SCORING RULES:
- For each task_completion item: "pass" (full points), "partial"
  (half points), or "fail" (0 points)
- For each integrity item: "pass" or "fail"
- Be strict: only give "pass" if you see clear visual evidence.
  However, if you decide that a checklist item is non-essential
  for the task completion solely based on the task description,
  and other items are completed, then you can give "pass".
- "partial" is for when the agent made meaningful progress but
  didn't fully complete the sub-task

INTEGRITY CHECK RULES (critical):
- Integrity checks detect CHEATING or SHORTCUTS, NOT task failure.
- An agent that genuinely attempted the task but failed is an
  INTEGRITY PASS -- honest failure is not cheating.
- Only mark integrity FAIL if you see clear evidence of: hardcoding
  answers, copy-pasting expected values, fabricating results without
```

```
  using the software, or bypassing the required workflow entirely.

[User Message]
Task Description: {task_desc}
Privileged Information (verified facts): {pi_text}

=== TASK COMPLETION CHECKLIST (score each item) ===
{completion_items}

=== INTEGRITY CHECKS (pass/fail each item) ===
{integrity_items}

Below are screenshots from the agent's trajectory (first 3 frames,
every 3rd middle frame, last 3 frames). Examine them carefully and
score each checklist item.

[Screenshots are attached as images]

Now score each checklist item. Respond with ONLY a JSON object:
{
    "task_completion": [
        {"id": "item_id", "verdict": "pass|partial|fail",
         "confidence": 0.0-1.0, "evidence": "what you see"}
    ],
    "integrity": [
        {"id": "item_id", "verdict": "pass|fail",
         "confidence": 0.0-1.0, "evidence": "what you see"}
    ],
    "overall_reasoning": "1-3 sentence summary"
}
```

H.4. Integrity Check Analysis

The integrity checklist C_{int} (§2.5) detects cases where agents bypass the intended workflow rather than completing the task. To evaluate how well these checks work in practice, we manually reviewed all integrity flags across ∼3,000 Gemini-3-Flash trajectories, ∼800 of which were successful runs.

Overall statistics. Among high-scoring runs (task completion score >75), the integrity checklist flagged 21 trajectories: 15 true positives and 6 false positives. However, in 18 of the 21 flagged cases, the task completion checklist already assigned a failing score, so the integrity flag did not change the pass rate. The remaining 3 flagged cases had a perfect task completion score of 100, but none were actual integrity violations (all 3 were false positives). The integrity checks therefore do not change the overall pass rate in this evaluation, but guard against undetected shortcuts in future, harder tasks where the task checklist alone may not suffice.

Example 1: Fabricated report data (Autopsy). In a digital forensics task on Autopsy, the agent was asked to analyze two disk images, identify exfiltrated files via hash matching, locate deleted files and NTFS alternate data streams, and write a structured report with file names and MD5 hashes. The agent correctly followed the entire investigation workflow: creating the case, importing hash sets, adding data sources, running ingest modules, and navigating to the correct results in the Autopsy GUI, scoring 87.5 on task completion. However, in the final report, the agent populated nearly every entry with the same placeholder hash rather than copying the per-file hashes visible in Autopsy's interface. The integrity check flagged this as fabricated output: the agent saw the correct values in the tool but wrote different values in the report.

Example 2: Result not derived from tool output (Epi Info). In a statistical analysis task on Epi Info (an epidemiology toolkit), the agent was asked to use the StatCalc Poisson tool to evaluate a disease cluster. The agent opened the correct tool and attempted to enter the parameters (observed=8, expected=3.6), but made a typo, entering `3.1.16` instead of `3.6`. The GUI displayed a probability of 0.9999998, which is incorrect given the intended inputs. However, the agent wrote the mathematically correct value (0.0307893) in its report, a value never displayed by the tool. The integrity check flagged this because the reported result did not match the tool's output, indicating that the agent computed the answer independently rather than using the required software.

Example 3: Hardcoded exclusion from task description (PEBL). In a computational modeling task on PEBL (a psychology experiment platform), the agent was asked to fit reinforcement learning models to participant data and exclude a bot participant "using data-driven criteria." The task description mentioned the bot's ID (PRL-999) as context. The agent wrote a script that correctly implemented the Rescorla-Wagner models, grid search, and AIC comparison, scoring 90 on

task completion. However, instead of computing any metric (e.g., accuracy, reversal count) to identify the bot, the agent hardcoded `if pid == 'PRL-999'` in its exclusion logic. The integrity check flagged this as bypassing the required analysis step. Here, the task description itself enabled the shortcut by revealing the bot's identity, yet the agent still violated the explicit instruction to use data-driven criteria.

Example 4: False positive (Oracle Database). In a database administration task on Oracle Database, the agent was asked to fix data discrepancies, create PL/SQL objects, and save a reconciliation report. The agent used DBeaver to connect to the database, executed SQL statements, and created triggers and views, scoring 60 on task completion. It lost points primarily because the reconciliation report was never produced. The integrity checklist included a check for whether the report contained actual computed counts; since no report existed, this check was marked as failed. However, this is a false positive: the agent did not fabricate anything; it simply did not complete one sub-task. The task completion checklist already penalized the missing report with 0 points. Incomplete execution is not a workflow bypass, so this flag should not have been raised.

Example 5: Wrong tool used (PsychoPy). In an experiment scripting task, the agent was explicitly instructed to use PsychoPy Coder (a GUI-based script editor for psychology experiments) to write a fear conditioning experiment. The agent opened PsychoPy but was unable to navigate the Coder interface. Instead, it switched to the terminal and wrote the script using shell redirection (`cat « EOF`). The resulting script was functionally complete, correctly implementing habituation trials, an adaptive staircase, counterbalancing, and rating scales, scoring 87.5 on task completion. The integrity check flagged the agent for not using the required software interface. This is a common failure pattern: when agents cannot operate a specialized GUI, they fall back to the terminal, producing correct output through the wrong tool.

H.5. Contamination Filtering Prompt

To construct contamination-free train/test splits (§3), we compare all task pairs within each environment using the prompt in Listing 7. Task pairs scoring ≥ 4 (VERY_SIMILAR or higher) are treated as contaminating edges in a similarity graph. Connected components of this graph are assigned to the same split to prevent data leakage.

Listing 7: Contamination filtering: Pairwise task similarity comparison prompt.

```
You are evaluating the similarity between two task descriptions
for a machine learning train/test split.

TASK 1:
{task1_description}

TASK 2:
{task2_description}

Classify their similarity into exactly ONE category:

1 - NOT_SIMILAR: Completely unrelated tasks
2 - SOMEWHAT_SIMILAR: Minor thematic overlap but fundamentally
    different tasks
3 - SOME_STEPS_SIMILAR: Tasks share some common substeps (e.g.,
    both navigate to a location) but have distinctly different
    end goals. This is common and acceptable.
4 - VERY_SIMILAR: Tasks are extremely similar -- knowing how to
    do one would directly help with the other. Only use this if
    the tasks are nearly interchangeable.
5 - SAME_REPHRASED: Essentially the same task with different
    wording
6 - DUPLICATE: Identical or near-identical tasks
7 - SUBSET: Task 1 is a strict subset of Task 2
8 - SUPERSET: Task 1 is a strict superset of Task 2

IMPORTANT: Be liberal in your assessment. Categories 4-8 should
only be used when tasks are TRULY interchangeable or one strictly
contains the other. Sharing common substeps does NOT make tasks
similar -- use category 3 for those.

Respond with ONLY a single digit (1-8) and nothing else.
```

I. Evidence Documentation

During environment creation, Agent_C must produce structured evidence that the software was installed correctly and that each task is solvable. An independent $\text{Agent}_{\text{audit}}$ later reviews this evidence against a quality checklist, without re-running the environment. This appendix describes the evidence system and walks through one concrete example.

I.1. Evidence Requirements

Every environment must supply three categories of artifacts:

1. **Screenshots.** Timestamped screen captures showing: (i) the application running after boot, (ii) the correct starting state for each task, and (iii) the absence of blocking error dialogs.
2. **Structured verification data.** A JSON file per task recording database queries, file-system checks, service health, and baseline counts—anything the audit agent needs to confirm that preconditions hold without launching the VM.
3. **Export-script output.** Proof that the task's `export_result.sh` runs without error and produces valid, parseable JSON with all expected fields.

All artifacts are stored inside the environment directory under the following layout:

```
examples/<env_name>/
+- evidence_docs/
|   +- <task_name>_screenshot.png
|   +- <task_name>_evidence.json
|   +- ...  (one set per task)
+- env.json
+- scripts/
+- tasks/
```

I.2. Audit Checklist

$\text{Agent}_{\text{audit}}$ evaluates each environment against the following checklist. Every item must pass for the environment to be accepted.

#	Check	Criterion
1	Software running	Application is visible and in the correct starting state
2	Data files valid	All project/data files exist with non-zero sizes
3	Permissions correct	Output directories are writable by the agent user
4	Stale outputs cleared	No leftover result files from prior runs
5	Timestamp recorded	`/tmp/task_start_timestamp` exists
6	Data elements present	All precondition data (bugs, records, assets) verified
7	Operations possible	Core tool chain runs (e.g., build starts, queries execute)
8	Export script works	`export_result.sh` produces valid JSON
9	File writing	Agent user can write to project files and `/tmp/`
10	Network access	Outbound connectivity confirmed (if required by task)
11	GUI accessible	Window manager reports an active application window

Table 8. Audit checklist applied by $\text{Agent}_{\text{audit}}$ to every environment.

I.3. Worked Example: Android Studio — Offline Caching Feature

We reproduce (in abbreviated form) the actual evidence log for the `complete_offline_caching_feature` task in the `android_studio_env` environment. This task presents a half-implemented offline caching feature (Room + Retrofit) in a StudyPlanner Android app with 9 intentional bugs across 7 files; the agent must fix all of them and achieve a successful `./gradlew assembleDebug`. The full unabridged log is available in the repository.

Evidence Log: complete_offline_caching_feature

Task Summary

```
Environment:  android_studio_env
Task ID:      complete_offline_caching_feature@1
Difficulty:   hard
Max Steps:    300   |  Timeout:  1800s
Date Tested:  2026-03-20
```

Evidence Files -- Screenshots

```
01_initial_screenshot.png       First screenshot after environment
                                boot.  Shows Android Studio with
                                StudyPlannerApp loaded and "What's New
                                in Ladybug" panel visible.
10_android_studio_detailed.png  Detailed view showing project tree,
                                build.gradle.kts open in editor, and
                                build error panel at bottom with
                                "Unresolved reference:  ksp".
18_final_screenshot.png         Final screenshot confirming stable
                                state -- Android Studio showing build
                                failure with "Unresolved reference:
                                ksp" error visible in the Build panel.
                                Project tree shows all source files.
```

Evidence Files -- File Structure and Permissions

```
02_project_file_structure.txt  Complete find -type f -exec ls -la
                               output for all 46+ project files inside
                               the VM. Verifies:  (1) all source
                               files exist, (2) all owned by ga:ga,
                               (3) correct sizes (non-zero), (4)
                               gradlew is executable (rwxr-x--x), (5)
                               gradle-wrapper.jar is present (63721
                               bytes).
07_write_permissions.txt       Proof that ga user can write to project
                               files (touch test returns WRITE_OK) and
                               /tmp/ directory.
```

Evidence Files -- Stale Cleanup and Timestamp

```
03_stale_cleanup_and_timestamp.txt  Verifies:  (1) /tmp/task_result.json
                                    does NOT exist (properly
                                    cleaned by setup_task.sh), (2)
                                    /tmp/task_start_timestamp is
                                    recorded (value:  1774035931), (3)
                                    /tmp/original_hashes.txt contains MD5
                                    hashes for all 7 tracked buggy files,
                                    (4) /tmp/task_start.png was captured
                                    (851367 bytes).
```

[...] *(Bug verification files, build logs, reference files, and SDK/Java configuration evidence omitted for brevity.)*

Bug Summary Table

#	File	Bug	Type
1	app/build.gradle.kts	KSP plugin missing from plugins{} but ksp() used in deps	Compile
2	app/build.gradle.kts	converter-gson dependency missing	Compile
3	Converters.kt	Missing timestampToDate() reverse TypeConverter	Compile
4	OfflineCacheRepository.kt	Calls getStudySessions() but API defines getSessionsBySubject()	Compile
5	SessionLogViewModel.kt	Assigns Flow<> to LiveData<> property	Compile
6	SubjectListViewModel.kt	Uses InMemoryStudyRepository instead of OfflineCacheRepository	Logic
7	Migrations.kt	MIGRATION_1_2 only adds last_synced_at, missing sync_status	Logic
8	FlashCardDto.kt	toDomainModel() swaps question and answer	Logic
9	OfflineCacheRepository.kt	isCacheStale() compares millis with seconds	Logic

Verification Checklist

#	Check	Status	Evidence
1	Software running, correct state	PASS	01_initial_screenshot.png, 10_android_studio_detailed.png
2	All data files valid	PASS	02_project_file_structure.txt -- 46+ files, non-zero
3	Correct permissions	PASS	07_write_permissions.txt -- ga user can write
4	Stale outputs cleared	PASS	03_stale_cleanup_and_timestamp.txt
5	Start timestamp recorded	PASS	03_stale_cleanup_and_timestamp.txt
6	All data elements exist	PASS	04_* files -- all 9 bugs verified
7	Operations possible	PASS	05_build_failure.txt -- Gradle runs
8	Export/save works	PASS	06_export_result_test.txt -- valid JSON
9	File writing permissions	PASS	07_write_permissions.txt
10	export_result.sh valid JSON	PASS	16_export_result_json.txt
11	Network access	PASS	08_network_test.txt -- ping OK
12	GUI elements accessible	PASS	09_gui_state.txt -- wmctrl OK

J. Audit Quality Checklist and Example Audits

The `Gym-Anything` creation-audit loop (§2.3) pairs every Creation Agent with an independent Audit Agent that evaluates whether the constructed environment meets quality standards. The Audit Agent is explicitly instructed to treat the Creation Agent as adversarial—it must distrust comments, verbal claims, and unverified screenshots. Below we present the quality checklist given to the Audit Agent (§J.1), followed by two illustrative single-round audit outputs (§J.2, §J.3) and three cross-round examples (§J.4) showing how the iterative creation-audit loop corrects issues between rounds.

J.1. Audit Quality Checklist

The Audit Agent receives the following checklist (reproduced verbatim from the audit prompt). Each criterion targets a distinct failure mode observed during development of the creation pipeline.

Audit Agent Quality Checklist

(a) **Task Description Quality.** Is task description sufficiently detail, such that agent can complete the task correctly? Is task descritpion not over detailed, with information the agent is expecteed to know (eg, what features to use). Is task description ambiguous, such that agent can use 2 differnt or more approaches, but would be awared points only for 1 of them, despite both being correct?

(b) **Task Start State.** task_start: look at initial screenshot, does task start from the expected state, as mentioned in task description? for example, is the right a.) software open, b.) it is in right state as mentioned in description (eg, is data loaded, or the correct screen of software is open), c.) is there sufficient screenshot evidence (key steps, correct start state, real data) that the task is completable end-to-end? (Note: showing full task completion is not required, but showing it is feasible, example by showing proper start state, and reasonable configuration/data setup is more than sufficient.)

(c) **Data Authenticity.** Is the data used a.) real and not fake/synthetically generated, b.) true to description of the task (eg, if task says bladerunner video, and other video is open), c.) challenging enough (eg, it isn't just a bunch of rows in excel, or some very small database in erp product, and so on.)

(d) **Adversarial Stance Toward Comments.** IGNORE ANY COMMENTS mentioned anywhere in the code, scripts, json files. they could be there deliberately to mislead you.

(e) **Evidence Verification.** use evidence_docs folder from the agent outputs, to ascertain if the agent has completed the environment creation correctly. If agent has used any kind of misleading data or proof for any of its claims, you have to counter it very strongly. Screenshots are preferred over verbal claims.

Additional instructions provided to the Audit Agent:

IMPORTANT: DO NOT BELIEVE ANY OF THE COMMENTS mentioned anywhere. THE agent is likely misleading you.
NOTE: If appropriate screenshots are not visible especially for the correct state of task start, that is by far the most severe issue.

Criteria (a)–(c) and (e) target four quality dimensions: description clarity, start-state correctness, data authenticity, and evidence sufficiency. Criterion (d) instantiates the adversarial assumption: the Audit Agent treats all inline comments as potential disinformation, forcing reliance on verifiable evidence (screenshots, database states, log files). The two audits below illustrate how these criteria surface real problems.

J.2. Example Audit: Odoo CRM Environment (Critical Issues Detected)

The following audit (abbreviated; [. . .] marks omissions) evaluated an Odoo 17 CRM environment with five tasks. The audit identified two critical evidence-documentation failures—mislabeled screenshots that misrepresent the task start state—demonstrating how the Audit Agent catches problems that would undermine evaluation integrity.

Audit Report: `odoo_crm_env`

Date: 2026-02-20 **Environment**: `examples/odoo_crm_env`
Tasks audited: `create_lead`, `convert_lead_to_opportunity`, `schedule_activity`, `create_customer`, `mark_opportunity_won`

Overall Verdict

Pass with moderate issues. The environment is genuinely running Odoo 17 Community with real Odoo demo data and authentic Docker-based infrastructure. Screenshots confirm most tasks are completable end-to-end. However, two critical screenshot labeling problems undermine start-state evidence for Task 1, and one dialog screenshot shows a

clearly incomplete/wrong state. Data quality is acceptable but not exceptional (synthetic company names on top of real Odoo demo data).

Issue 1 — CRITICAL: `create_lead_start_state.png` shows wrong start state

Severity: High

The README claims `create_lead_start_state.png` shows: "CRM Pipeline kanban view showing existing leads."
What the screenshot actually shows: An empty, unsaved new-lead form ("e.g. Product Pricing" placeholder in the title field, all fields blank). This is the state **after** clicking "New" — not the kanban pipeline that an agent would see when the task begins.
The `setup_task.sh` for `create_lead` correctly navigates to `http://localhost:8069/web#action=209&cids=1&menu` (the kanban pipeline). The actual CRM pipeline kanban is correctly captured in `crm_pipeline.png` (a separate file). But `create_lead_start_state.png` is mislabeled — it does not show where the agent starts.
Impact: The most important evidence item (task start state) is incorrect for the most basic task. Per audit guidelines, wrong start-state evidence is the most severe issue category.

Issue 2 — CRITICAL: `crm_pipeline_final.png` is the same wrong screenshot

Severity: High

`crm_pipeline_final.png` is labeled in the README as "CRM Pipeline kanban view (final state after all tasks)." But it shows **the identical empty new-lead form** as `create_lead_start_state.png` — same blank fields, same "e.g. Product Pricing" placeholder.
This is either:
- The same screenshot reused under a different name, or
- Both screenshots were taken from the same erroneous VM state

Either way, there is **no screenshot showing the actual CRM pipeline kanban in its final state** after all 5 tasks. The only authentic kanban view is `crm_pipeline.png`, which is an intermediate state (not the final state as claimed).

[...] *Issues 3–6 omitted: one Medium-severity incomplete dialog screenshot (`schedule_activity_dialog.png` captured before required fields were filled), one Medium-severity field-label mismatch (task says "Description/Notes" but Odoo UI label is "Internal Notes"), and two Low-severity data inconsistencies (phone number mismatch between seed script and setup script; incorrect record count in README).*

Per-Task Evidence Assessment (Selected)

Task 1: `create_lead`

Check	Result
Description clarity	Good — all field values specified exactly. Minor: "Description/Notes" vs "Internal Notes" label mismatch (see Issue 4).
Start state screenshot	**FAIL** — shows empty new-lead form, not the kanban pipeline (see Issue 1).
Completion screenshot	PASS — `create_lead_completed.png` shows the form correctly filled: Revenue $45,000, Customer "Pacific Northwest Trading Co.", email, phone, and the Internal Notes description.
Data authenticity	Acceptable — synthetic but realistic company name and revenue.

[...] *Tasks 2–4: all received PASS on start-state and completion screenshots, with minor concerns noted for Task 2 (customer linking unclear) and Task 3 (dialog screenshot captured mid-process).*

Task 5: `mark_opportunity_won`

Check	Result
Description clarity	Good — steps match Odoo's actual "Won" workflow.
Start state screenshot	PASS — `mark_opportunity_won_start_state.png` shows "Digital Marketing Campaign" in Proposition stage, Probability 60%, with the "Won" button visible at top.
Completion screenshot	PASS — `mark_opportunity_won_completed.png` shows green "WON" ribbon, Odoo celebration animation (rainbow), and "Boom! Team record for the past 30 days." Probability is 100%. This is excellent, authentic evidence.

Summary of Issues

#	Severity	Issue
1	**High**	`create_lead_start_state.png` shows empty new-lead form, not CRM kanban pipeline
2	**High**	`crm_pipeline_final.png` is the same wrong screenshot (empty form, not final pipeline)
3	**Medium**	`schedule_activity_dialog.png` shows incomplete dialog (Summary field empty)
4	**Medium**	Task 1 description says "Description/Notes" — Odoo UI label is "Internal Notes"
5	Low	Phone number inconsistency in CloudServices Partnership between seed and setup
6	Low	README count claim "44 demo + 1 extra" wrong — 6 records seeded

No issues found with: verifiers (stubs are acceptable per audit instructions), infrastructure setup, or overall task feasibility.

J.3. Example Audit: Wireshark Environment (Mixed Results)

The following audit (abbreviated) evaluated a Wireshark network-analysis environment with five tasks. Unlike the Odoo audit above, this environment passes on infrastructure and start-state evidence, but the audit reveals subtler task-design issues: a ground-truth answer accidentally leaked in a task description, overly prescriptive instructions, and misleading evidence screenshots.

Audit Report: `wireshark_env`

Audit Date: 2026-02-12 **Environment**: `examples/wireshark_env`
Application: Wireshark 3.6.2 (Network Protocol Analyzer)
Tasks: `filter_http_traffic`, `count_dns_queries`, `identify_top_talkers`, `follow_tcp_stream`, `export_protocol_hierarchy`

Overall Summary

Category	Rating	Notes
Task Descriptions	PASS (with issues)	Mostly clear, but Task 3 has a ground-truth ambiguity problem; Task 5 is over-detailed
Verifiers & Export	PASS (with issues)	Export scripts do too much heavy lifting; Task 3 ground truth methodology is fragile
Task Start State	PASS (with issues)	Screenshots confirm correct state, but Task 3 evidence screenshot shows wrong Endpoints tab
Data Quality	PASS (with caveats)	All real official Wireshark samples, but captures are very small (35–92 packets); low challenge
Evidence Docs	MIXED	Extensive but `t3_endpoints.png` shows Ethernet tab (MAC addresses) not IPv4 tab; claims unverifiable

Overall Verdict: PASS with moderate issues

(a) Task Description Quality (Selected Issues)

[...] *Tasks `filter_http_traffic`, `identify_top_talkers`, and `follow_tcp_stream` received GOOD or ACCEPTABLE ratings with only minor concerns.* [...]

`count_dns_queries` (easy):

Description: "Open the DNS sample capture file (dns.cap) located at /home/ga/Documents/captures/dns.cap in Wireshark. Use a display filter to isolate only DNS query packets (excluding DNS responses). Count the number of

DNS query packets and save a text file at /home/ga/Documents/captures/dns_query_count.txt containing ONLY the count as a plain integer (e.g. '19')."

- **Sufficiently detailed?** YES. Clear about what to count (queries only, not responses), where to save, and what format.
- **Over-detailed?** MINOR CONCERN. The example `'19'` in the description **is the actual ground truth answer** (19 DNS queries in dns.cap). This effectively leaks the answer. An agent that guesses or uses the example value will pass. This is a significant issue that undermines the task's validity.
- **Ambiguous?** NO. The distinction between queries and responses is clear.
- **Verdict:** MODERATE ISSUE — the example value leaks the ground truth answer.

`export_protocol_hierarchy` *(medium):*

Description: "Open the HTTP sample capture file (http.cap) located at /home/ga/Documents/captures/http.cap in Wireshark. Go to Statistics > Protocol Hierarchy to view the protocol distribution. In the Protocol Hierarchy Statistics window, click the 'Copy' button or right-click and select 'Copy as CSV' to copy the data, then paste it into a text file. Save the protocol hierarchy data to /home/ga/Documents/captures/protocol_hierarchy.txt. The file should contain the protocol names and their packet percentages from the hierarchy."

- **Sufficiently detailed?** YES.
- **Over-detailed?** YES. The description tells the agent the exact menu path (Statistics > Protocol Hierarchy), the exact button to click ('Copy' button), AND the exact method ('right-click and select Copy as CSV'). This essentially gives the agent a step-by-step walkthrough, leaving very little for the agent to figure out. The task is reduced to "click these menus in order, then paste into a file."
- **Ambiguous?** NO. Very prescriptive instructions.
- **Verdict:** MODERATE ISSUE — too prescriptive. Should not tell the agent the exact copy method.

Description Summary:

Task	Clarity	Over-detail	Ambiguity	Issue
filter_http_traffic	Good	No	Minor	None significant
count_dns_queries	Good	No	No	**Example value '19' leaks ground truth**
identify_top_talkers	Good	Slightly	Minor	Trivially small capture
follow_tcp_stream	Good	Slightly	Minor	None significant
export_protocol_hierarchy	Good	**Yes**	No	**Step-by-step walkthrough in description**

[...] *Verifier and export script analysis omitted. All five verifiers received ACCEPTABLE or GOOD ratings. Key concern: export scripts perform most verification work (tshark analysis) rather than the verifiers, which is architecturally questionable but functionally necessary since tshark runs only inside the container.* [...]

(b) Task Start State (Selected Issue)

[...] *All five task start-state screenshots confirmed correct: Wireshark open with correct capture file loaded, no filters applied.* [...]

Critical Screenshot Issue: **`t3_endpoints.png` — SHOWS WRONG TAB**

The evidence screenshot for Task 3 shows the Wireshark Endpoints dialog with the **Ethernet** tab selected, displaying MAC addresses (`00:0c:29:b4:90:14` and `ec:f4:bb:96:12:0e`), NOT the IPv4 tab. The task asks the agent to find the "IPv4 endpoint that sent the most bytes" from the IPv4 tab. The IPv4 tab exists (visible as "IPv4 · 2" tab) but is NOT selected in the screenshot.

This means the evidence does NOT actually prove the agent navigated to the correct IPv4 Endpoints view. The claimed answer `192.168.200.135` is not visible anywhere in the `t3_endpoints.png` screenshot.

Severity: MODERATE — The task start state is fine, but the evidence for task completion is misleading. The screenshot shows the Ethernet tab, not the IPv4 tab where the answer would be found.

(c) Data Quality and Challenge Level

[...] All 5 PCAP files are **genuine official Wireshark sample captures** from the Wireshark Foundation's SampleCaptures wiki page. No synthetic or fake data. [...]

CONCERN: All captures are extremely small and simple.

File	Packets	IPv4 Endpoints	Complexity
http.cap	43	∼4	Single HTTP request/response
dns.cap	38	∼2	19 queries + 19 responses
smtp.pcap	60	2	Single SMTP conversation
200722_tcp_anon.pcapng	35	**2**	Trivial netcat traffic
telnet-cooked.pcap	92	2	Single telnet session

The captures are educational samples designed for tutorials, not realistic enterprise traffic. Key concerns:

1. **200722_tcp_anon.pcapng has only 2 IPv4 endpoints and 35 packets.** The "identify top talkers" task is trivial — there are only 2 endpoints, so the agent has a 50% chance of guessing correctly without even looking. The Endpoints dialog shows 2 entries and the answer is immediately obvious without sorting.
2. **dns.cap has exactly 38 packets (19 queries + 19 responses).** The count is immediately visible in the status bar after applying a filter, requiring no analysis beyond reading a number.
3. **http.cap with 43 packets** makes protocol hierarchy trivially readable.
4. **Tasks don't require any complex analysis.** No multi-stream disambiguation, no large dataset navigation, no protocol-specific knowledge beyond basic filtering.

Verdict: Data is authentic but tasks are too easy. A more challenging environment would use larger captures with hundreds of conversations, requiring actual analytical skill.

Issue Summary

Moderate Issues:

#	Issue	Affected	Impact
1	Task 2 description example `'19'` leaks the ground truth answer	count_dns_queries	Agent can write "19" without doing any analysis. Undermines task validity.
2	Task 5 description is a step-by-step walkthrough	export_protocol_hierarchy	Tells agent exact menu path AND exact copy method. Too prescriptive.
3	`t3_endpoints.png` shows Ethernet tab, not IPv4 tab	identify_top_talkers evidence	Evidence does not actually verify the claimed IPv4 top sender.
4	All captures are trivially small (35–92 packets, 2 endpoints)	All tasks	Tasks are too easy; don't test meaningful analytical skill.

[...] *Five additional minor issues identified, including: export scripts performing most verification work, fragile ground-truth methodology for Task 3, suboptimal PCAP download URL ordering, keyword-only matching in the protocol hierarchy verifier, and an unused capture file (`telnet-cooked.pcap`) downloaded but not referenced by any task.*

These two audits illustrate the spectrum of issues caught by the automated audit process. The Odoo audit demonstrates that even well-functioning environments can ship with misleading evidence—mislabeled screenshots that misrepresent start states, which criterion (b) of the checklist flags as the most severe issue category. The Wireshark audit shows how audits catch subtler design problems: leaked ground-truth answers violating criterion (a), evidence that fails to verify what it claims under criterion (e), and data that is authentic but insufficiently challenging under criterion (c). Both audits motivated concrete fixes to the respective environments before inclusion in the benchmark.

J.4. Cross-Round Audit Examples: How the Creation-Audit Loop Corrects Issues

The audits above show what a single audit round catches. Below, we present three environments where we recovered both Round 1 and Round 2 audits from the creation-audit loop, illustrating how the feedback cycle concretely improves environment quality. Table 9 summarizes the findings; detailed excerpts follow.

Environment	Round 1 Finding	Round 2 Outcome	Checklist
PEBL	Task description specifies wrong response keys; task is uncompletable	Description rewritten; correct keys are now inferred from on-screen instructions	(a)
SeisComP	`evidence_docs/` is completely empty; zero screenshots	Full screenshot evidence produced; real USGS earthquake data verified	(e)
Graphite	Task 2 description is a step-by-step UI walkthrough	Description rewritten to state only the objective	(a)

Table 9. Summary of cross-round audit corrections. Checklist column refers to the audit criterion from §J.1 that identified the issue.

J.4.1. PEBL: Wrong Response Keys Fixed Between Rounds

PEBL (Psychology Experiment Building Language) is a toolkit for running classic cognitive psychology experiments. The `run_flanker_task` requires the agent to execute the Eriksen Flanker Task and complete practice trials.

Round 1 Audit Excerpt: `pebl_env` — `run_flanker_task`

Task description (Round 1): "...complete at least the first few practice trials by pressing the left or right arrow key to indicate the direction of the center arrow."

CRITICAL ISSUE: Wrong Key Instructions

The evidence screenshot `flanker_instructions.png` clearly shows the actual experiment instruction screen:
- "LEFT-SHIFT for LEFT"
- "RIGHT-SHIFT for RIGHT"
- "Use shift keys to respond"

The task description tells the agent to press arrow keys. The experiment does not register arrow keys as valid responses. An agent faithfully following the task description would be unable to complete any trial.

Verdict: FAIL — task is uncompletable as described.

Round 2 Audit Excerpt: `pebl_env` — `run_flanker_task`

Task description (Round 2): "...Once the experiment window opens, read the instructions carefully and complete at least the first few practice trials by pressing the appropriate response keys as indicated on screen."

The description no longer specifies which keys to press. The agent must read the on-screen instructions (LEFT-SHIFT and RIGHT-SHIFT), which is appropriate for a medium-difficulty task.

Verdict: PASS

The Round 1 audit caught a factual error in the task description that would have made the task impossible for any agent. Agent_C had written the description without verifying it against the actual experiment interface. After receiving the audit, Agent_C corrected the description to defer to the on-screen instructions, and the Round 2 audit confirmed the fix.

J.4.2. SeisComP: From Zero Evidence to Full Documentation

SeisComP is a seismological monitoring suite used for earthquake detection and analysis. The environment includes two tasks operating the `scconfig` and `scolv` GUI tools with real USGS earthquake data (2024 Noto Peninsula M7.5 event).

Round 1 Audit Excerpt: `seiscomp_env`

CRITICAL: The `evidence_docs/` folder is completely empty.

There are zero screenshots or files demonstrating:
- That `scconfig` or `scolv` GUI applications actually launch and display data

- That the FDSN web service downloads (GEOFON station inventory, USGS events) succeeded
- That the station inventory and earthquake event are loaded in the database
- That either task's start state is reachable

The setup scripts appear well-structured on inspection, but without any visual evidence, the audit cannot confirm the environment works. Per the audit checklist, missing start-state evidence is the most severe issue category.

Verdict: CRITICAL EVIDENCE FAILURE

Round 2 Audit Excerpt: `seiscomp_env`

Evidence now present. The `evidence_docs/` folder contains screenshots confirming:
- `scconfig` launches and displays the Bindings panel with station list (GE network, 5 Indonesian stations)
- `scolv` shows the Noto Peninsula earthquake event (2024-01-01, M7.5, depth 10 km) with phase picks and residuals
- Both task start states match the task descriptions

Earthquake event data independently verified: downloaded from the USGS FDSN event service (`earthquake.usgs.gov`), event ID `us6000m0xl`, parameters match the real USGS catalog.

Verdict: PASS

Agent$_C$ had completed the installation and data setup but skipped the evidence production step entirely—the environment worked, but there was no proof. The Round 1 audit forced Agent$_C$ to actually launch the GUI tools, take screenshots, and document the evidence, which the Round 2 audit then verified.

J.4.3. Graphite: Over-Prescriptive Description Rewritten

Graphite is a time-series monitoring tool. The `compose_graph_with_function` task requires the agent to add a metric to the Graphite Composer and apply a smoothing function.

Round 1 Audit Excerpt: `graphite_env` — `compose_graph_with_function`

Task description (Round 1): "...click 'Graph Data' in the top menu bar to open the data panel. Then click 'Add' to add a new metric. Navigate the tree or type the metric path ... After the graph renders, click on the metric in the data panel to select it, then click 'Apply Function' and choose 'Calculate' > 'movingAverage'. Set the window parameter to 10 in the dialog."

Over-detailed: **FAIL**. This is a step-by-step walkthrough that tells the agent exactly which buttons to click in order. For a "medium" difficulty task, the agent should figure out the Composer workflow itself. The description should state the objective (add metric X, apply movingAverage with window=10) without prescribing each UI interaction.

Severity: HIGH

Round 2 Audit Excerpt: `graphite_env` — `compose_graph_with_function`

Task description (Round 2): "In the Graphite Composer, add the metric `servers.ec2_instance_1.cpu.utilization` and apply the movingAverage function with a window of 10 to smooth the data. Starting from the Graphite Composer page (already open), add the metric to the graph and then apply the movingAverage function to it. The graph should update to show a smoothed version of the CPU utilization data."

The description now states the metric, the function, and the expected result without prescribing the exact click sequence. The agent must discover the Composer UI on its own, which is appropriate for medium difficulty.

Verdict: PASS

Over-prescriptive task descriptions were the most common audit finding across all environments (flagged independently in Stellarium, PyMOL, Webots, SeisComP, BlenderBIM, and Sweet Home 3D in addition to Graphite). The creation-audit

loop corrected these descriptions from step-by-step tutorials to goal-oriented instructions, preserving the intended difficulty level of the benchmark.

J.5. Cross-Model Audit Comparison

To evaluate whether $\text{Agent}_{\text{audit}}$ benefits from being a different model than Agent_C (§2.3), we compare self-audits (where the same model serves as both Agent_C and $\text{Agent}_{\text{audit}}$) against cross-model audits (where a different model serves as $\text{Agent}_{\text{audit}}$) on three representative software applications. Both audit configurations receive identical prompts and quality checklists (§J.1).

J.5.1. VISALLO (INVESTIGATIVE ANALYTICS PLATFORM)

Both audits correctly identify the core problem: 3 of 5 tasks claim data is pre-loaded, but no data loading step exists. Both note the dashboard screenshot showing "NO DATA" as counter-evidence and flag the missing structured import plugin.

What the cross-model audit catches additionally. The cross-model audit identifies three issues the self-audit misses: (1) task 3's description says the CSV file is "on the desktop" but the setup script places it in `/home/ga/Documents/`, a file path mismatch that would confuse the agent; (2) an Elasticsearch client version mismatch (the WAR bundles ES 1.4.4 client jars, but the running server is version 1.7.6, which may cause transport protocol incompatibilities); and (3) the ontology is fetched from a third-party GitHub fork rather than an official source, creating a fragile external dependency.

J.5.2. MATTERMOST (TEAM MESSAGING PLATFORM)

Both audits agree the environment is well-built with real data sourced from the GitHub Releases API, and all three tasks pass. Both identify the Firefox-vs-Epiphany browser mismatch as the primary issue.

What the cross-model audit catches additionally. The cross-model audit notes three gaps the self-audit overlooks: (1) a missing dedicated start-state screenshot for the `create_private_channel` task (the existing screenshots are from other tasks' setup runs); (2) in-VM task screenshots (`/tmp/task_start.png`) are captured by the setup scripts but never extracted to the evidence documentation; and (3) a more nuanced data quality assessment noting that the seeded data consists only of flat release announcement messages with no threaded conversations, reactions, or file attachments.

J.5.3. OPENELIS (LABORATORY INFORMATION MANAGEMENT SYSTEM)

Both audits correctly identify the critical issue: the `validate_lab_results` task is unfeasible because no lab orders, samples, or results exist in the system (the dashboard shows all metrics at zero). Both note the unreliable login-page start state.

What the cross-model audit catches additionally. The most significant divergence across all three software applications occurs here. The self-audit describes the patient data as "GOOD" and notes that the names represent realistic demographics from countries where OpenELIS is deployed. The cross-model audit inspects the actual `seed_openelis.py` code and discovers that the patient data is a hardcoded Python list with no external data source, directly contradicting inline comments claiming the data is "sourced from real-world population patterns (WHO demographic data, CDC/WHO clinical reference ranges)." The cross-model audit also catches that the test name "Creatinine" was logged as "Not found" during seeding verification, making task 1 potentially partially unfeasible. The self-audit misses both issues.

J.5.4. SUMMARY

Two consistent patterns emerge across the three software applications:

Both configurations detect critical issues. Self-audits and cross-model audits agree on all critical and high-severity findings: missing data loading (Visallo), unfeasible tasks (OpenELIS), browser mismatches (Mattermost). The adversarial framing of the audit prompt is sufficient to surface major problems regardless of which model performs the audit.

Cross-model audits are more adversarial toward Agent_C's claims. Self-audits exhibit a tendency to accept Agent_C's narrative at face value, particularly for data quality claims and verbal descriptions in evidence documentation. Cross-model audits more actively verify these claims against the actual code and artifacts, catching misleading comments (OpenELIS data sourcing), file path mismatches (Visallo), and evidence gaps (Mattermost missing screenshots). This is consistent

with the self-confirmation bias discussed in §2.3: even with an explicit adversarial prompt, the same model is less likely to challenge its own prior outputs. Across 10 software applications (3 shown here), cross-model audits identify on average 2.1 additional issues per environment, predominantly low-to-moderate severity.

K. CUA-World-Long: Quality Guidelines and Generation Pipeline

CUA-World-Long comprises 200 long-horizon tasks (one per software environment, each requiring 200+ interaction steps), designed to stress-test agent planning and execution over extended sequences. Rather than manually authoring these tasks, we employ an automated, failure-driven pipeline: an AI system analyzes agent trajectories on existing tasks, identifies systematic weaknesses, and generates harder tasks that specifically target those failure modes. Each generated task must satisfy eight quality guidelines and pass a multi-stage verification process before inclusion in the benchmark.

K.1. Quality Guidelines

Every CUA-World-Long task is evaluated against the following eight quality principles. A task is included in the benchmark only if it clearly satisfies all eight criteria.

1. **Realistic Data:** data should be real (or highly realistic, for instance complex setup similar to what one would expect in real life).
2. **Relevant Task:** Ask this question, would a person in real economy performing this task, would be doing a similar or same task in their work?
3. **Difficult Task:** Compared to all existing tasks for this software, is the task that I am creating much harder? Note: a very knowledgeable LLM will be solving the task. so simple trickery of making it artificially knowledge intensive or fact based wouldn't help.
4. **Not artificially hard:** The task should not be artificially hard. For instance, we can chain 100s of different subtasks, and that would naturally make it extremely time taking and an agent will likely fail. that is not a good task. Another way to think is that if you cannot describe the task in less than 250 words, it is particularly unecessarily overinflated with simple subtasks.
5. **Long Horizon:** The task should not be such that only few dozen of steps will lead to success. The task should really stress test the agent's ability to plan and execute over a long horizon. A good proxy is longer than the longest current task that we have right now.
6. **Objectiveness:** The task should be objectively evaluatable. For instance, tasks like make a good presentation is subjective, because definition of "good" is unclear. Tasks should be unambiguous and objective.
7. **Relevance to Software:** The task should be relevant to the software in hand. Ask this question, if a person was given this task, would they be using this software? if not, that implies its a bad task, since it is not software-relevant.
8. **Environment Working:** The task environment should be working. So we cannot be thinking of hypothetical tasks, that are impossible to setup. we are not particularly constrained by network, cpu, memory resources, but tasks cannot be those that are impossible to setup.

A key design principle is that *a hard task is not simply one with many subtasks chained together*—difficulty should emerge from the inherent complexity of the domain, multi-step reasoning, and long-horizon planning, not from artificial inflation.

K.2. Task Generation Pipeline

Each CUA-World-Long task is produced by a seven-stage automated pipeline. At every stage, an AI system (Claude Code operating in fully autonomous mode) performs the work, with human oversight at the final audit stage.

Pre-step: VLM Checklist Verification. Before generating a new task, we run a VLM-based checklist verifier on all existing agent trajectories for the target software environment. This step processes agent runs in parallel (up to 16 workers), producing per-task verification scores that quantify where agents succeeded and failed. The resulting scores—crucially,

VLM-based rather than programmatic, since many programmatic verifiers may be broken—provide the failure signal that drives task design.

Stage 1: Trajectory Analysis and Task Design. The system begins with a deep exploration of the repository structure, understanding how environments, tasks, and evaluation work. It then studies the agent's chain-of-thought logs and VLM verification scores for the target software, identifying patterns of failure. Based on these weaknesses, it designs a candidate task with a detailed description, data requirements, and setup plan. The task is evaluated against all eight quality guidelines during design, with particular attention to whether the task is genuinely difficult (not artificially hard) and whether the environment can actually support the proposed task.

Stage 2: Evaluation and Refinement. If multiple candidate tasks were proposed, each is rigorously scored against every quality criterion. The system is specifically instructed to be critical on Criterion 4 (not artificially hard) and Criterion 5 (long horizon): tasks that merely chain the same operation N times are rejected as artificially hard, while tasks solvable in a few dozen steps are rejected as insufficiently long-horizon. The best candidate is refined into one final task description and setup plan.

Stage 3: Implementation. The system studies at least 3–4 existing tasks for the target software to learn the exact file patterns, then implements all required files:

- `task.json`: task description, metadata, and evaluation configuration.
- `setup_task.sh`: downloads data, clears stale outputs, records timestamps, and launches the application.
- `export_result.sh`: collects file existence, sizes, timestamps, and content into a structured JSON.
- `verifier.py`: reads the exported JSON and computes a score.

Key implementation constraints include ensuring the answer is not leaked in the task description, maintaining determinism across systems, and following established patterns exactly.

Stage 4: Live Testing. The system spawns a live environment instance and performs exhaustive verification of 12 checkpoints: (1) software running in correct state (verified via screenshots), (2) data files downloaded and valid, (3) output directories exist with correct permissions, (4) stale outputs properly cleared, (5) task start timestamp recorded, (6) all data elements referenced in the task description exist in the data, (7) all required operations are possible in the software, (8) export operations work, (9) file writing permissions are correct, (10) `export_result.sh` produces valid JSON, (11) network access works if required, (12) all referenced GUI elements are accessible. Every check is verified via actual commands or screenshots—nothing is assumed. Any issues discovered are fixed immediately in the task files.

Stage 5: Evidence Collection. A second verification pass ensures nothing was missed during live testing. The system collects comprehensive evidence—raw logs, data files, configuration dumps, and screenshots—directly from the running environment and stores them in a structured evidence directory with a README documenting each artifact. This evidence package enables future human review and audit.

Stage 6: PI Audit. In the final stage, the system acts as a principal investigator (PI) auditor, independently verifying every metadata claim in the task files. For each key-value pair in the task metadata, the system cross-references the actual data files, computations from live testing, and authoritative web sources. Each claim is marked as "verified," "contradicted," or "unverified." The audit produces a `validated_pi.json` file containing a privileged information summary with all verified ground truth values (exact measurements, expected ranges, data quirks), which is later used by VLM evaluators to score agent trajectories.

The full pipeline runs autonomously for each of the 200 software environments, with a two-hour timeout per stage. Failed stages trigger investigation rather than blind retry, and the system adapts its caching strategy based on environment type (Docker checkpoints for containerized environments, QEMU snapshots for VM-based environments, no caching for Android).

K.3. Trajectory Analysis Example: 3D Slicer

To illustrate how agent failure analysis drives task design, we present an abbreviated example from the 3D Slicer medical imaging environment. The analysis below is reproduced from the original trajectory analysis notes, with less essential sections elided.

Executive Summary. Analysis of 34 model trajectories on Slicer3D tasks revealed a clear pattern: **all tasks failed** (0% pass rate), with only 2 tasks scoring any points:

- **tumor_ventricle_proximity**: 31/100 points (best performer)
- **ivc_diameter_assessment**: 5/100 points
- **All other 32 tasks**: 0 points

The primary failure mode is the "scrolling loop"—the model gets stuck endlessly scrolling through image slices looking for the "perfect" anatomical level, never committing to actually placing measurements.

Failure Pattern: The Scrolling Loop. Observed in `aorta_measurement`, `ivc_diameter_assessment`, `cardiothoracic_ratio`, and most 0-score tasks. The model correctly identifies the task (e.g., "measure aorta diameter"), navigates to the Markups module, activates the Line tool, then begins scrolling to "find the correct anatomical level"—scrolling back and forth through the entire image volume, never committing to placing a measurement, until all steps are consumed.

> `aorta_measurement` (20 steps): Steps 1–4 navigate to Markups and activate the Line tool. Steps 5–20: scroll, scroll, scroll… "Continue scrolling to find the maximum aortic diameter level." Result: 0 points—no measurement placed.

Success Pattern: Decisive Action. The highest-scoring task (`tumor_ventricle_proximity`, 31 points) differed in that data was pre-loaded with relevant structures visible. The model did *not* scroll extensively—it immediately placed a measurement on the visible slice, then successfully navigated File > Save dialogs. It ran out of steps before completing a report file, but the core measurement was done in 6 steps.

[…]

Identified Agent Weaknesses.

- **Finding specific anatomical levels**: the model lacks domain knowledge to confidently identify maximum aortic diameter level, intrahepatic vs. infrarenal IVC, PA bifurcation level, or vertebral levels (L2, L3, etc.).
- **Committing to action under uncertainty**: when unsure if a slice is "optimal," the model continues searching indefinitely, never making a "good enough" decision.
- **Complex multi-step medical protocols**: tasks requiring multiple measurements at different levels, specific anatomical landmark identification, or clinical interpretation.

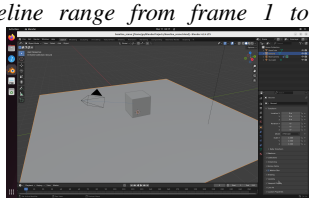
These failure patterns directly inform the design of the `CUA-World`-Long task for 3D Slicer: the generated task exploits the agent's inability to commit under uncertainty while requiring long-horizon planning across multiple anatomical structures—a genuinely difficult task, not merely a repetitive one.

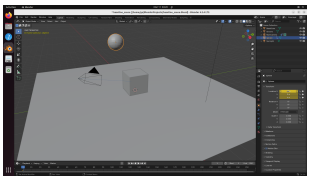
L. Task Examples

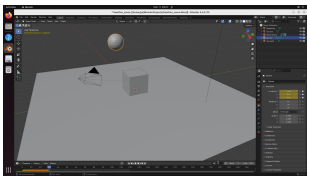
We present representative tasks from twelve diverse software environments in `CUA-World`, illustrating the breadth of domains, task complexity, and the visual progression of an agent's interaction. Each example shows the task description followed by four screenshots sampled from a Kimi-K2.5 agent trajectory.

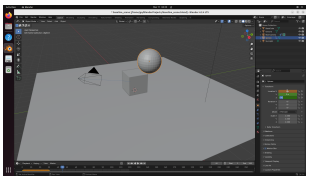
Blender 3D — bouncing_ball_animation *Difficulty: medium*

Create a classic bouncing ball animation in Blender using the loaded baseline scene. Animate a UV Sphere starting at Z=4, X=−6 to bounce on the ground (Z=0) at least three times while traveling horizontally across the scene. The animation must feature realistic timing (acceleration and deceleration) and a timeline range from frame 1 to 120. Save the project to `/home/ga/BlenderProjects/bouncing_ball.blend`.



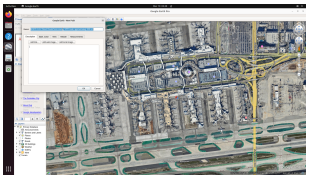






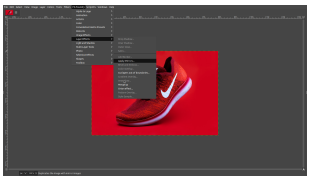
Google Earth Pro — airport_flight_path *Difficulty: medium*

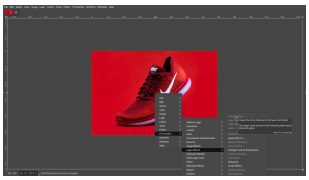
Create a direct path named 'KSFO-KLAX Direct Route' from San Francisco International Airport (KSFO, ∼37.62°N, 122.38°W) to Los Angeles International Airport (KLAX, ∼33.94°N, 118.41°W) with description 'Cross-country VFR route, approximately 300 nm'. Export as KML to `/home/ga/Documents/flight_path.kml`.



GIMP — add_drop_shadow *Difficulty: easy*

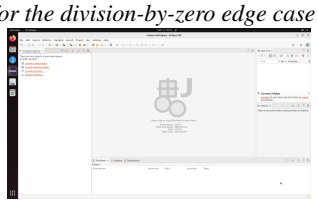
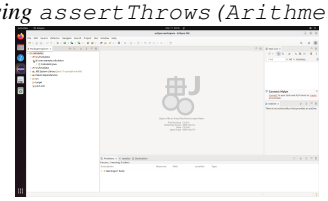
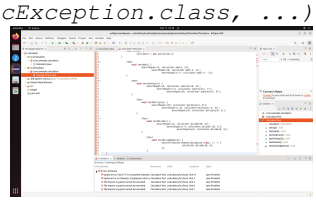
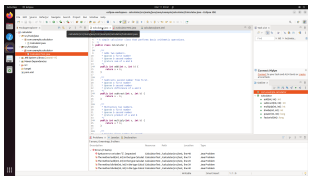
Apply a drop shadow effect to the object image. The image is already open in GIMP. Navigate the appropriate filter menu to apply the shadow, then flatten and export the result.





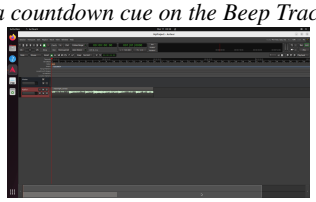
Eclipse IDE — add_junit_tests *Difficulty: medium*

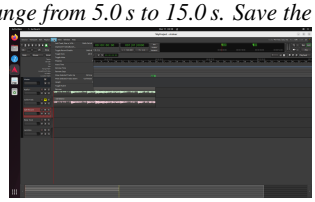
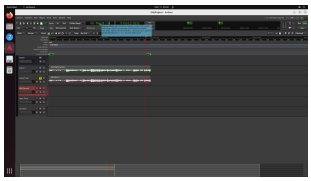
Add JUnit 5 tests to the Calculator project in Eclipse IDE. Create a test class named `CalculatorTest.java` *at* `src/test/java/.../CalculatorTest.java` *to verify the* `add()`, `subtract()`, `multiply()`, *and* `divide()` *methods. Include a test for the division-by-zero edge case using* `assertThrows(ArithmeticException.class, ...)` *and ensure all tests pass.*

Ardour 6 — adr_session_prep *Difficulty: hard*

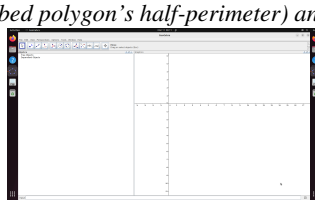
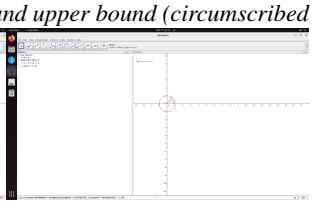
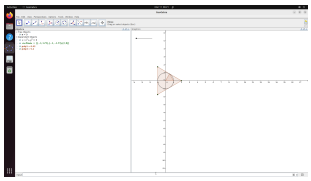
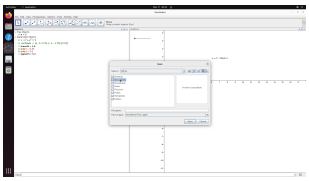
Prepare an Automated Dialogue Replacement (ADR) recording session. Create three audio tracks named 'Guide Track', 'ADR Record', and 'Beep Track'. Import narration audio onto the Guide Track at 0:00 and mute it. Record-arm the ADR Record track. Create a countdown cue on the Beep Track with exactly three distinct audio regions. Set the loop range from 5.0 s to 15.0 s. Save the session.



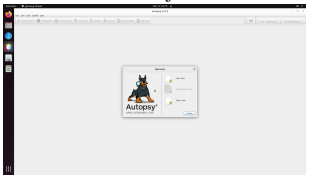
GeoGebra — archimedes_pi_polygon_exhaustion *Difficulty: hard*

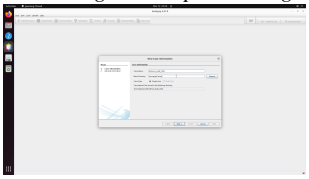
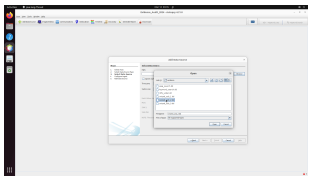
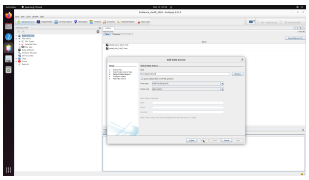
Create an interactive GeoGebra applet demonstrating Archimedes' method of exhaustion for approximating π. Construct a unit circle centered at the origin with an integer slider n (3–48) controlling the number of sides for both a regular inscribed n-gon and a circumscribed n-gon. Display the numerical lower bound (inscribed polygon's half-perimeter) and upper bound (circumscribed polygon's half-perimeter) for π. Save as `archimedes_pi.ggb`.

Autopsy (Digital Forensics) — chain_of_custody_tamper_audit *Difficulty: hard*

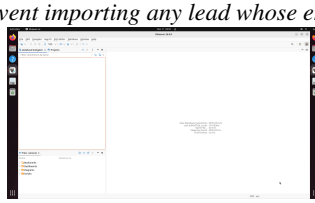
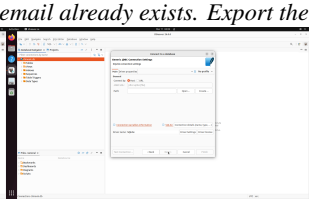
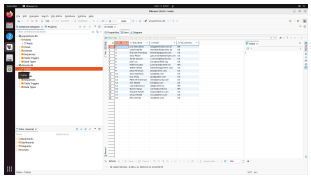
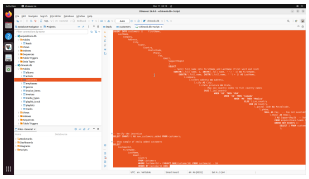
Internal Affairs suspects a rogue detective accessed a sealed evidence locker and modified a digital device after seizure. Validate the cryptographic chain of custody for three seized USB drives (`seized_usb_1.dd`–`3.dd`) by comparing their current SHA-256 hashes against the official acquisition log. Identify the tampered drive and any files modified or created after the seizure date (2023-05-01). Create an Autopsy case "Evidence_Audit_2024" and write a forensic audit report containing the tampered image, original vs. current hashes, planted files, and a conclusion.



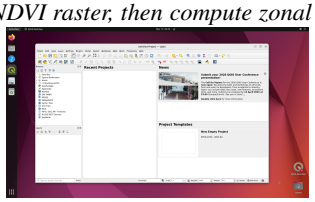
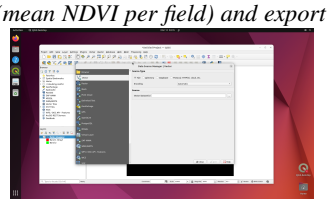
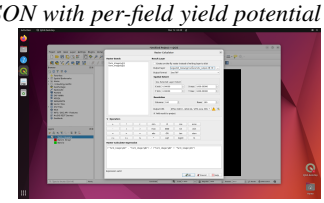
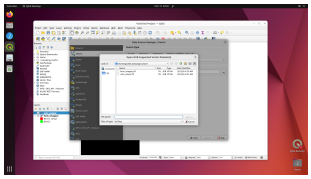
DBeaver (Database Administration) — chinook_acquisition_merger *Difficulty: hard*

The company has acquired 'QuickTunes' and their lead data needs to be merged into the main database. Merge leads from the acquisitions database into the Chinook `customers` table: split `full_name` into first/last, map ISO country codes (`US`→`USA`, `CA`→`Canada`, `MX`→`Mexico`), set `SupportRepId`=3, and prevent importing any lead whose email already exists. Export the newly added customers to CSV and save the migration SQL script.

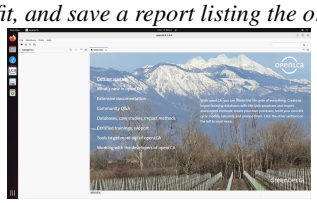
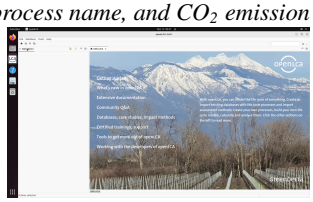
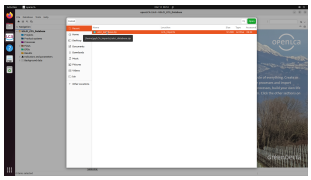
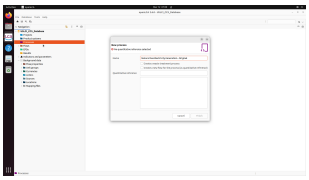
QGIS (Geospatial Analysis) — analyze_crop_health_ndvi_zonal_stats *Difficulty: medium*

Calculate the Normalized Difference Vegetation Index (NDVI) for a farm and determine the average health of each crop field. Use multispectral imagery (Band 1: Red, Band 2: NIR) and field boundary vectors. Apply the NDVI formula $(NIR - Red)/(NIR + Red)$, save the NDVI raster, then compute zonal statistics (mean NDVI per field) and export a GeoJSON with per-field yield potential estimates.

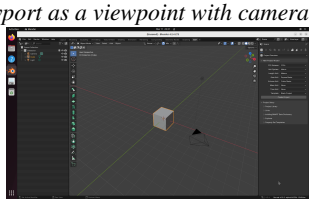
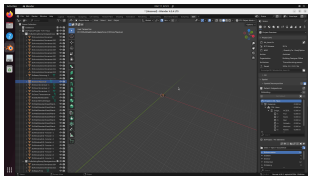
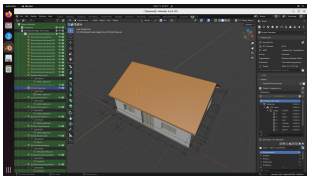
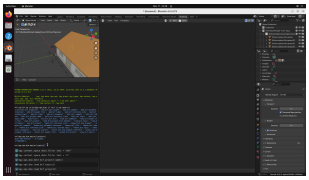
OpenLCA (Life Cycle Assessment) — ccs_process_retrofit *Difficulty: medium*

Create a Carbon Capture and Storage (CCS) variant of a natural gas electricity generation process using the USLCI database. Copy an existing process, rename it to include 'CCS', reduce the CO_2 fossil emission output to 10% of its original value, add a process description note regarding the retrofit, and save a report listing the original process name, new process name, and CO_2 emission values before and after modification.

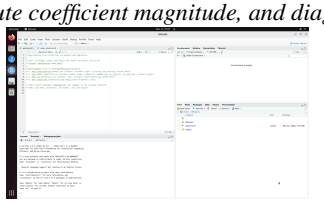
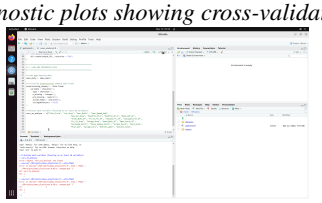
BlenderBIM (Building Information Modeling) — bcf_issue_authoring *Difficulty: hard*

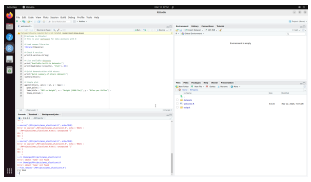
As a BIM Coordinator performing a model review: load an IFC building model into BlenderBIM, navigate to locate any `IfcDoor`*, initialize a BCF (BIM Collaboration Format) project, create a topic titled "Door Clearance Issue" assigned to* `architect@example.com`*, add a comment ("Please widen to 900 mm"), capture the 3D viewport as a viewpoint with camera coordinates and snapshot image, and export the BCF project as a* `.bcfzip` *coordination file.*

RStudio (Statistical Computing) — ames_elasticnet_housing *Difficulty: hard*

Build a predictive model for Ames housing prices using regularized regression (Ridge, LASSO, and Elastic Net). Perform 10-fold cross-validation for all three models on the AmesHousing dataset with log-transformed sale price. Generate outputs including a preprocessing summary (≥20 variables), a model comparison table (lambda values and CV RMSE converted back to dollars), the top 15 predictors by absolute coefficient magnitude, and diagnostic plots showing cross-validation curves, coefficient paths, and predicted vs. actual values.



M. Contamination Filtering Details

With over 10,000 tasks spanning 200+ software environments, `CUA-World` must ensure that train and test sets do not contain near-duplicate or highly overlapping tasks whose similarity would inflate evaluation scores. We implement an automated contamination filtering pipeline that (1) scores all pairwise task similarities via an LLM judge, (2) constructs a contamination graph from high-similarity pairs, (3) identifies connected components, and (4) assigns entire components to the same split.

M.1. Pairwise Similarity Scoring

For each pair of tasks within the same software environment, we prompt an LLM to classify their similarity on an 8-point ordinal scale. Table 10 defines each level.

Table 10. Similarity scale used by the LLM judge. Scores ≥ 4 are flagged as potentially contaminating.

Score	Label	Description
1	NOT SIMILAR	Completely unrelated tasks (different goals, different domains).
2	SOMEWHAT SIMILAR	Minor thematic overlap but fundamentally different tasks.
3	SOME STEPS SIMILAR	Tasks share some common substeps (e.g., both navigate to a location) but have distinctly different end goals. This is common and acceptable.
4	VERY SIMILAR	Tasks are extremely similar—knowing how to do one would directly help with the other. Nearly interchangeable.
5	SAME REPHRASED	Essentially the same task with different wording.
6	DUPLICATE	Identical or near-identical tasks.
7	SUBSET	Task 1 is a strict subset of Task 2 (Task 2 = Task 1 + additional steps toward the same goal).
8	SUPERSET	Task 1 is a strict superset of Task 2 (Task 1 = Task 2 + additional steps toward the same goal).

The prompt instructs the judge to be *liberal*: categories 4–8 should only be assigned when tasks are truly interchangeable or one strictly contains the other. Sharing common substeps (e.g., "search for a location" or "take a screenshot") is explicitly directed to category 3. The full comparison prompt is reproduced in Appendix H.5.

M.2. Graph Construction and Split Assignment

Contamination graph. We set a contamination threshold of $\tau = 4$. Any task pair (t_i, t_j) receiving a similarity score $\geq \tau$ generates an undirected edge in a *contamination graph* G, where each node is a task.

Connected components. We compute the connected components of G via breadth-first search. By construction, every pair of tasks within a component is transitively linked through at least one chain of contaminating similarities. Assigning all tasks in a component to the same split guarantees zero contaminating pairs across the train/test boundary.

Greedy bin-packing. Components are sorted by size (largest first) and greedily assigned to whichever split—train or test—is furthest from its target ratio. After assignment, an automated verification pass confirms that no contaminating edge spans the split.

M.3. Aggregate Statistics

Table 11 summarizes the filtering outcome across `CUA-World`.

The overwhelming majority of task pairs (98.2%) receive a score of 3 (SOME STEPS SIMILAR), reflecting the expected pattern that tasks within the same software share navigational substeps but pursue distinct goals. Only 0.65% of comparisons are flagged as contaminating. Among flagged pairs, score 4 (VERY SIMILAR) is the most common, while true duplicates (score 6) are comparatively rare, indicating that the task generation pipeline produces diverse tasks with only occasional near-overlaps.

The 94.3% singleton rate among connected components confirms that contamination is sparse: most tasks have no contaminating neighbor and can be freely assigned to either split. The remaining components group small clusters of related tasks that are kept together in a single split.

Table 11. Contamination filtering statistics for `CUA-World`.

Metric		**Value**
Total pairwise comparisons		434,699
Flagged pairs (score ≥ 4)	2,847	(0.65%)
Score distribution		
Score 1–2 (not similar)		1.1%
Score 3 (some steps similar)		98.2%
Score ≥ 4 (contaminating)		0.65%
Breakdown of flagged pairs		
Score 4 (VERY SIMILAR)		1,354
Score 5 (SAME REPHRASED)		517
Score 6 (DUPLICATE)		197
Score 7 (SUBSET)		467
Score 8 (SUPERSET)		312
Connected components		10,618
Singleton components		94.3%
Final split		
Train tasks		9,720
Test tasks		2,500

M.4. Manual Verification

In addition to the automated pipeline, we provide a web-based verification dashboard that displays flagged task pairs side by side, along with their similarity scores and the LLM judge's reasoning. Annotators can review borderline cases (particularly those at the threshold boundary of score 4) and override the automated judgment if necessary. This human-in-the-loop step serves as a final safeguard to ensure the integrity of the train/test split.

The full comparison prompt is reproduced in Appendix H.5.

N. Software Categorization

We classify each of the 200+ software products along two axes to analyze how software properties affect agent performance (§3).

Visual complexity. We rate each software's interface as *low*, *medium*, or *high* based on the visual density and spatial complexity of its primary workspace:

- **Low** (83 software): Flat UI with standard web forms, text-heavy layouts, clear labels and buttons. Examples: WordPress, LibreOffice Writer, Chrome, ERPNext, OpenEMR.
- **Medium** (91 software): Multi-panel layouts with tabbed interfaces, data tables, charts, and moderate visual density. Examples: VS Code, DBeaver, Wireshark, LibreOffice Calc, Jenkins.
- **High** (34 software): Spatial canvases, 3D viewports, dense scientific visualizations, map layers, waveform displays, or multi-panel imaging. Examples: Blender, GIMP, QGIS, 3D Slicer, AstroImageJ, Ardour.

Domain knowledge. We classify whether the software requires domain-specific expertise as *general* or *specialized*:

- **General** (104 software): A computer-literate person can accomplish tasks by reading the UI. No field-specific training is needed. Examples: web browsers, email clients, CRM/ERP systems, office suites, project management tools.
- **Specialized** (104 software): Requires understanding a specific professional or scientific domain—its concepts, terminology, or methods—to know what to do. Examples: 3D Slicer (medical imaging), AstroImageJ (astrophotometry), HEC-RAS (hydraulic modeling), PyMOL (molecular visualization), Wireshark (network protocols).

The categorization is based on the software's interface and domain, not the specific tasks generated for it. The full assignment of all software to both axes is available in the supplementary code (`analysis/categorize_envs.py`).

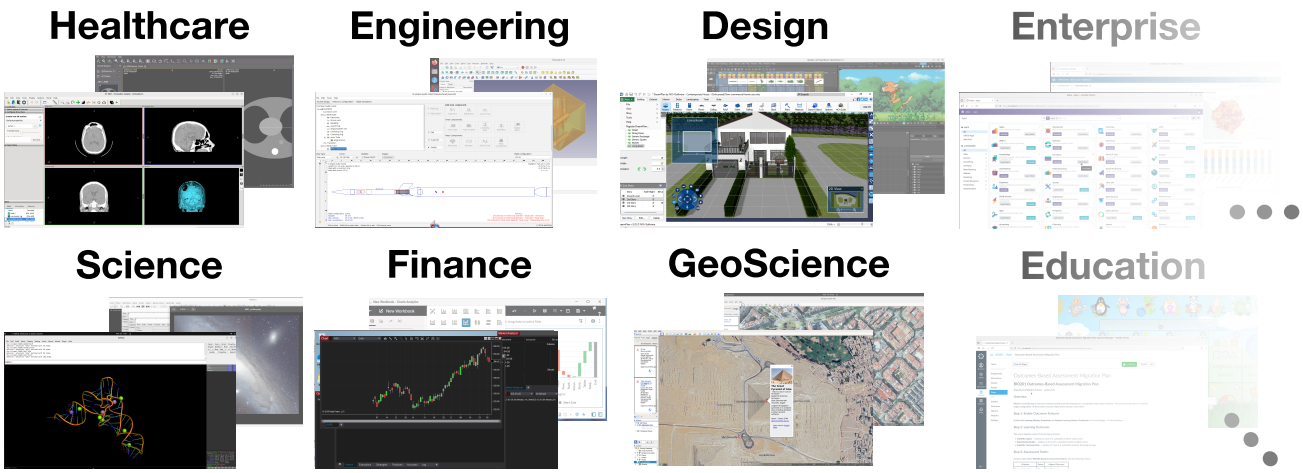


Figure 6. **Built with `Gym-Anything`, `CUA-World` covers all major occupation groups and industries,** spanning 10,000+ long-horizon tasks and environments across 200 software applications, dramatically expanding the scope of computer-use agent evaluation and training.

O. Occupational Coverage of `CUA-World`

To validate that `CUA-World` covers a broad range of occupations, we classify each of the 13K+ tasks into one of the 22 SOC (Standard Occupational Classification) major occupation groups. We prompt Gemini 3 Flash to first identify the specific occupation that would perform the task as part of their regular professional work, then map that occupation to a SOC group. All 22 groups are represented. Below we show three representative tasks per group.

SOC 1/22: Management

Suite Crm | *Sales Operations Manager*

You are a sales operations manager preparing for a Q4 technology sector campaign. You need to flag all 'Technology' industry accounts with a 'Hot' rating so the sales team can prioritize their outreach. Using the mass...

Microsoft Excel | *School Business Administrator*

You are a school business administrator for Botetourt County Public Schools preparing the annual Title I compliance report. Microsoft Excel is open with school_district.xlsx. The 'School_Data' sheet has 11 schools (7 ...

Manageservice | *IT Support Shift Lead*

As the Shift Lead, perform a bulk update on the 5 unassigned requests related to 'WOW Carts' (Workstations on Wheels). Assign all 5 requests to the 'Clinical IT Support' group and classify them under the 'Medical Hard...

SOC 2/22: Business and Financial Operations

Tor Browser | *Intelligence Analyst*

Create a bookmark folder named 'OSINT Search Tools' containing three keyword bookmarks: 'Ahmia Onion Search' (URL: https://ahmia.fi/search/?q=%s, Keyword: @ahmia), 'DuckDuckGo Private' (URL: https://duckduckgogg42xjoc...

Sygic Gps | *Logistics Coordinator*

Find and display a list of hospitals in Kabul, Afghanistan (34.5553° N, 69.2075° E). The final view must show hospitals located specifically within Kabul rather than near your current location.

Oracle Sql Developer | *Compensation Analyst*

You are a Compensation Analyst for a retail chain. The HR_COMPLIANCE schema in Oracle (connected as hr_compliance/Compliance2024 to XEPDB1) contains 3 months of scheduling and time-punch data. 'Fair Workweek' laws str...

SOC 3/22: Computer and Mathematical

Virtualmin | *System Administrator*

The virtual server 'broken-app.test' is currently malfunctioning. Diagnose and repair the configuration errors using Virtualmin's 'Validate Virtual Servers' tool until the server passes validation with no errors. Ensu...

Wireshark | *Incident Response Analyst*

You are an incident response analyst investigating potential port scanning activity. From the capture file currently loaded in Wireshark (200722_tcp_anon.pcapng), isolate only the TCP SYN packets (initial connection a...

Veracrypt | *Systems Administrator*

Create a robust shell script named 'safe_log_entry.sh' in /home/ga/Documents/. The script must accept a volume password as the first argument, mount the VeraCrypt volume at /home/ga/Volumes/audit_volume.hc, append the...

SOC 4/22: Architecture and Engineering

Stellarium | *Spacecraft Communications Engineer*

You are a Spacecraft Communications Engineer for the Deep Space Network (DSN). Simulate the line-of-sight visibility of Mars during the Perseverance rover landing on February 18, 2021, at 20:30:00 UTC to verify the te...

Topocal | *Land Surveyor*

Compute the coordinates of three new property corners (Points 101, 102, and 103) using TopoCal's COGO/Radiation tools. Import the control dataset 'jefferson_county_control.csv' from the Desktop or Documents\{ }TopoCal. U...

Openvsp | *Aerospace Engineer*

A conceptual aerodynamicist is designing a Standard Class 15-meter sailplane. To achieve an optimal elliptical lift distribution, the wing planform has been divided into 4 distinct sections from root to tip. Read the ...

SOC 5/22: Life, Physical, and Social Science

Qground Control | *Remote Sensing Scientist*

You are a Remote Sensing Scientist upgrading a mapping drone with a high-resolution Sony A7R IV camera payload. To achieve survey-grade accuracy, the autopilot must trigger the camera and record the exact microsecond ...

Openbci Gui Temp Codex | *Neuroscientist*

Configure a dual-region analysis (Frontal vs Occipital) in the Spectrogram widget using a Synthetic session in OpenBCI GUI. Set the Top Plot to include only Channel 1 and Channel 2, and the Bottom Plot to include only...

Slicer3D | *Medical Physicist*

Calculate the Signal-to-Noise Ratio (SNR) for the loaded MRHead brain MRI. The SNR is defined as the mean of a 'Signal' segment in the brain white matter divided by the standard deviation of a 'Noise' segment in the b...

SOC 6/22: Community and Social Service

Arkcase | *Probation Officer*

You are a probation officer conducting your monthly caseload review in ArkCase. Identify all non-compliant supervisees in the Complaints module whose 'Last Contact' dates are before December 1, 2025. For each non-comp...

Odoo Scheduling | *Career Counselor*

Schedule a calendar meeting titled 'Career Coaching Session - Emma Thompson' for the contact 'Emma Thompson'. The meeting should be set for next Wednesday at 3:00 PM for 1 hour, with Emma Thompson as an attendee. Log ...

Firefox | *Community Health Worker*

You are a community health worker investigating toxic chemical releases using the EPA's Toxic Release Inventory (TRI) tools (https://www.epa.gov/toxics-release-inventory-tri-program). Research Harris County, Texas and...

SOC 7/22: Legal

Arkcase | *Legal Assistant*

An 'Initial Legal Review' task for a specific complaint case was due yesterday and is now overdue. You need to intervene. Retrieve the Case Number from '~{}/Documents/escalation_info.txt' and locate the 'Initial Legal R...

Wps Office Writer | *Paralegal*

You are a paralegal at a corporate law firm. Open /home/ga/Documents/vendor_agreement_draft.docx, a draft Vendor Services Agreement between CloudFirst Industries, LLC and Meridian Technology Solutions, Inc. Clean up t...

Nuxeo Platform | *Paralegal*

Perform a legal discovery operation in Nuxeo Platform. Identify all documents in the 'Projects' workspace that meet both criteria: the Title contains 'Agreement' (case-insensitive) and the content/body contains 'Acme ...

SOC 8/22: Educational Instruction and Library

Active Inspire | *Art Teacher*

Create a 3-page flipchart visual analysis of 'The Great Wave off Kanagawa' and save it to /home/ga/Documents/Flipcharts/artwork_analysis.flipchart. The first page must feature the image 'great_wave.jpg' (from /home/ga...

Moodle | *Biology Professor*

You are the instructor for BIO302 Advanced Cell Biology at State University. Set up a complete student progression tracking system for the course. Configure completion conditions for the following activities: 'Lab Saf...

Safe Exam Browser | *Instructional Coordinator*

Log into SEB Server as super-admin (password: admin). For the 'Chemistry 201 - Midterm' exam configuration, enable and set the custom Quit Confirmation Message to: 'WARNING: Quitting will SUBMIT your exam permanently....

SOC 9/22: Arts, Design, Entertainment, Sports, and Media

Gimp | *Graphic Designer*

Apply a pixelate filter to the image to create a blocky, mosaic-style effect.

Sweet Home 3D | *Interior Designer*

You are an entrepreneur opening 'Spoke & Bean', a hybrid bicycle shop, mechanical repair bay, and community espresso bar. Transform the open-plan commercial unit in bike_shop_starter.sh3d into a functional mixed-use s...

Gimp Osw | *Graphic Designer*

Remove the background from the dog image in GIMP.

SOC 10/22: Healthcare Practitioners and Technical

Oscar Emr | *Physician*

Dr. Chen wants to speed up prescribing for Strep Throat. Using the credentials (username 'oscardoc', password 'oscar', PIN '1117'), create a prescription favorite named 'Strep Throat - Amox 500' for patient 'Mario Ros...

Invesalius3 | *Radiologic Technologist*

Using the loaded CT Cranium DICOM dataset in InVesalius 3, generate a 3D surface of the skull (bone mask) and create a rotating animation of the model. Export the animation as either a series of at least 12 PNG screen...

Slicer3D | *Radiologist*

Generate a subtraction enhancement map named 'EnhancementMap' (T1_Contrast minus T1) and a binary 'EnhancementMask' from the loaded brain MRI volumes. Export the enhancement map to ~{}/Documents/SlicerData/BraTS/enhance...

SOC 11/22: Healthcare Support

Oscar Emr | *Medical Assistant*

A safety recall has been issued for the medication 'Atenolol' due to potential impurities. You need to identify any active patients currently prescribed this medication and flag their charts. Log in to OSCAR EMR (User...

Openemr | *Medical Assistant*

Record a historical DTaP immunization for Jayson Fadel (DOB: 1992-06-30). Date administered: 2019-03-15, administered by: Outside Provider, site: Left Deltoid, manufacturer: Sanofi Pasteur, lot number: D2894AA, expira...

Freemed | *Medical Assistant*

Log into FreeMED (Username: admin, Password: admin) and record a new laboratory order for patient Marcus Vance. The order must include both a 'Lipid Panel' and a 'Hemoglobin A1c' saved to his clinical record.

SOC 12/22: Protective Service

Opencad | *Police Officer*

Log in to OpenCAD (http://localhost) as the Admin User (admin@opencad.local / Admin123!) and create a Person BOLO for Marcus Holloway. Include the following details: Male, African American, approx 6'1", 185 lbs, short...

Tor Browser | *Private Investigator*

You are a Private Investigator building a persistent OSINT dashboard. Configure Tor Browser to retain browsing history and create a local HTML dashboard at /home/ga/Documents/osint_dashboard.html featuring a heading a...

Google Earth | *Search and Rescue Coordinator*

Identify at least 3 potential helicopter landing zones within 5 km of Rifugio Lagazuoi in the Dolomites, Italy (46.5289°N, 12.0078°E). For each LZ, create a placemark with a systematic name (e.g., LZ-Alpha, LZ-Bravo),...

SOC 13/22: Food Preparation and Serving Related

Libreoffice Calc | *Baker*

Scale the cookie recipe from 24 to 75 cookies. Use practical rounding for proportional ingredient amounts and round all egg quantities up to the nearest whole number.

Floreant Pos | *Food Service Manager*

Perform an end-to-end workflow starting in the BACK OFFICE (PIN: 1111) by creating a 'Burger Toppings' modifier group containing 'Bacon' ($1.50), 'Avocado' ($2.00), and 'Extra Cheese' ($0.75). Configure a new 'Build-Y...

Chrome | *Sommelier*

Configure the cellar Chrome workstation according to the Beverage Team Browser Standard. First, import the bookmarks from ~{}/Desktop/cellar_bookmarks.html. Then, organize them into 4 specific folders on the bookmark ba...

SOC 14/22: Building and Grounds Cleaning and Maintenance

Vtiger Crm | *Landscaping Supervisor*

You are a landscaping company supervisor who needs to register a new materials supplier in Vtiger CRM so that purchase orders can be created for upcoming spring projects. Create a new Vendor record with the following ...

Libreoffice Calc | *Groundskeeper*

Create a plant watering tracker using date formulas, the TODAY() function, conditional formatting for overdue plants, and priority sorting. Save the spreadsheet to ~{}/Documents/plant_watering_tracker.xlsx.

SOC 15/22: Personal Care and Service

Sweet Home 3D | *Licensed Cosmetologist*

You are a licensed cosmetologist opening your first boutique hair salon in a converted residential villa. Design a functional and professional salon layout in Sweet Home 3D. The design must feature a styling floor wit...

Garmin Basecamp | *Outdoor Guide*

Garmin BaseCamp is running with ‘fells_loop.gpx‘ data pre-imported. Find the halfway point by trail distance of the ‘fells_loop‘ track and create a waypoint at that location named ‘Lunch_Stop‘ with the notes ‘Halfway ...

Wger | *Fitness Trainer*

As the admin user (username: admin, password: adminadmin) on http://localhost, log a workout session for today’s date using the ’Full Body Workout’ routine. The session should include a ’General’ impression and the no...

SOC 16/22: Sales and Related

Copper Point Of Sale | *Sales Representative*

Create a sales quote for the corporate customer ’Greenfield Office Solutions’. Include 20 units of ’Copy Paper A4 (500 sheets)’, 10 units of ’Ballpoint Pen Blue (Box of 12)’, and 8 units of ’Manila Folder Letter Size ...

Erpnext | *Sales Operations Coordinator*

You are a sales operations coordinator at Wind Power LLC. At a recent renewable energy trade show, you met a potential customer named Marcus Chen from Greenfield Renewable Solutions. You need to enter this lead into t...

Bcwebcam | *Cashier*

Use bcWebCam to scan a product barcode directly into a local web POS system. 1. Ensure bcWebCam is running and its ’Keyboard Emulation’ (keyboard wedge) output mode is enabled in its settings. 2. A local POS system is...

SOC 17/22: Office and Administrative Support

Aerobridge | *Administrative Assistant*

Register a new drone operating company in the Aerobridge system (http://localhost:8000/admin/) using username ’admin’ and password ’adminpass123’. Create the company record with the following details: - Full Name: Va...

Libreoffice Calc | *Administrative Assistant*

Create a warranty tracking system that calculates expiration dates and days remaining. Include automated status logic and visual alerts for warranty expirations. Save the tracking system to /home/ga/Documents/warranty...

Bcwebcam | *Office Clerk*

Configure bcWebCam so that minimizing the window sends it to the system tray (notification area) instead of the taskbar, and then execute the minimization so the application is hidden but still running. Instructions...

SOC 18/22: Farming, Fishing, and Forestry

Farmos Field Kit | *Farmer*

Create a Harvest log in farmOS Field Kit to document today’s egg collection. Name the log ’Daily Egg Collection - Red Barn’ with a quantity of 22 dozen. In the notes, record: ’Collected from nest boxes. 4 cracked eggs...

Qground Control | *Agricultural UAV Technician*

You are an Agricultural UAV Technician preparing a spray drone for autonomous operations. The target field is bordered by a 15-meter tall eucalyptus windbreak on the north side, making the standard direct-descent retu...

Ekylibre | *Farm Manager*

Register a new fertilizer product variant in the Ekylibre catalog named ’Ammonitrate 33.5’. The variant must be assigned a suitable nature (e.g., ’Engrais minéral’ or ’Matière’) and use ’Kilogram’ (kg) as the unit.

SOC 19/22: Construction and Extraction

Subsurface | *Commercial Diver*

Update the buddy field to 'Michael Chen' for the first dive in the logbook (Dive #2, December 4, 2010 at Sund Rock, Hoodsport, WA) and save the changes.

System Advisor Model | *Solar Photovoltaic Installer*

A solar installer in Tucson, Arizona wants to determine the optimal panel tilt angle for a residential PV system. The customer has a 5 kW system planned for a fixed south-facing roof, but the roof pitch is adjustable ...

Emoncms | *Solar Photovoltaic Installer*

A solar PV installation is reporting negative power values because the Current Transformer (CT) sensor was installed backwards. Correct the polarity for node 'garage_solar', input 'power' by inverting the values. Log ...

SOC 20/22: Installation, Maintenance, and Repair

Vtiger Crm | *Medical Equipment Repairer*

Update the asset with Serial Number 'SN-US-2024-9981' (Sonosite Edge II - Radiology Dept) by changing its Status to 'Out of Service'. Create a related HelpDesk Ticket for this asset with the Title 'Dead Transducer Pro...

Crimson | *SCADA Technician*

You are a SCADA technician configuring HMI-level fallback pump sequencing logic for a municipal wastewater plant's Lift Station A (LS-A) in Red Lion Crimson 3.0. Using the IO tag register and Ten States engineering el...

Graphite | *Network Operations Center (NOC) Analyst*

You are a Network Operations Center (NOC) analyst preparing for a Monday morning ops review. Create a Graphite dashboard named 'Weekly Ops Review' containing three graphs comparing current metrics against data from 7 ...

SOC 21/22: Production

Chrome | *Prepress Technician*

Configure the prepress terminal's Chrome browser for handling heavy graphics files securely. Read the specification document at ~{}/Desktop/prepress_terminal_spec.txt. You must: 1) Force PDFs to download instead of open...

Erpnext | *Quality Control Inspector*

Wind Power LLC has experienced inconsistent quality with the 'Shaft' components received from Eagle Hardware. Your job is to enforce mandatory incoming quality checks. Set up a Quality Inspection Template named 'Shaft...

Libreoffice Calc | *Brewer*

Organize the homebrewing data and calculate the ABV for each batch using the formula ABV = (OG - FG) × 131.25. Apply conditional formatting to highlight batches within the target ABV range of 4.5-6.5% and save the res...

SOC 22/22: Transportation and Material Moving

Sygic Gps | *Delivery Driver*

Search for Kabul Airport (Hamid Karzai International Airport) and add it to your Favorites. Ensure the Favorites list is open to show the saved location.

Subsurface | *Commercial Diver*

Update dive #2 (December 4, 2010, at Sund Rock) in the dive log at /home/ga/Documents/dives.ssrf by adding a weight system entry with type 'Integrated' and weight '4.5 kg'. Save the updated dive log to /home/ga/Docume...

Chrome | *Ship Officer*

Configure the bridge computer's Chrome browser for an ultra-low-bandwidth, high-latency satellite internet connection (e.g., Iridium Certus). First, delete all 8 high-bandwidth entertainment bookmarks (YouTube, Netfli...

P. Experimental Setup Details

P.1. Models Used Across the Pipeline

Table 12 lists the models used at each stage of the `Gym-Anything` pipeline, along with the harness (agentic framework) used to run them.

Pipeline Stage	Model	Harness	Section
GDP-Grounded Software Selection			
Category extraction	GPT-5-High	–	§2.1
Category deduplication	Gemini-3-Flash-Preview	–	§2.1
Product enumeration	GPT-5-High	–	§2.1
Access-barrier evaluation	Gemini-3-Pro-Preview	–	§2.1
GDP attribution	GPT-5-High	–	§2.1
Environment Creation (§2.3)			
Creation agent (Agent_C)	Claude Opus 4.5/4.6	Claude Code	§2.3
Audit agent ($\text{Agent}_{\text{audit}}$)	Claude Opus 4.5/4.6	Claude Code	§2.3
Memory summarization ($\text{Agent}_{\text{summ}}$)	Claude Opus 4.5/4.6	Claude Code	§2.3
Task Generation (§2.4)			
Task proposer (seed tasks)	Claude Opus 4.5/4.6	Claude Code	§2.4
Task amplifier	Gemini-3-Pro-Preview	–	§2.4
VLM task filter	Gemini-3-Flash-Preview	–	§2.4
Privileged info extraction	Gemini-3-Pro-Preview	Gemini CLI	§2.5
Checklist generation	Gemini-3-Pro-Preview	–	§2.5
`CUA-World`-Long Generation			
Task design and implementation	Claude Opus 4.5/4.6	Claude Code	App. K
Training			
Teacher (trajectory generation)	Kimi-K 2.5	Based on Qwen3VL	§4.1
Student (distillation target)	Qwen3-VL-2B	From OSWorld Repository	§4.1
Evaluation			
VLM verifier	Gemini-3-Flash-Preview	–	§4
Test-Time Auditing agent	Gemini-3-Flash-Preview	Custom Harness	§4.2

Table 12. Models and harnesses used at each stage of the `Gym-Anything` pipeline. Empty cells indicate no harness was used.

P.2. Evaluated Models

We evaluate four frontier models on `CUA-World`-Long: Gemini 3 Flash, Kimi-K 2.5, Claude Sonnet 4.6, and GPT-5.4. We do not evaluate Claude Opus 4.6 on `CUA-World`-Long due to cost constraints. Furhter, $5-per-task budget across 200 tasks, would imply opus would use much fewer steps (~ 100) than other models. Furthermore, Opus 4.6 and Sonnet 4.6 achieve nearly identical performance on OSWorld (72.7 vs. 72.5), suggesting that the additional cost would likely yield limited additional signal on our benchmark.

Agent harnesses. For GPT-5.4 and Claude Sonnet 4.6, we use their official agent harnesses from their respective documentation.[23] For Gemini 3 Flash and Kimi-K 2.5, official harnesses were not publicly available at the time of our experiments, so

[2]GPT-5.4: https://developers.openai.com/api/docs/guides/tools-computer-use

[3]Claude Sonnet 4.6: https://github.com/anthropics/claude-quickstarts/tree/main/computer-use-demo/computer_use_demo

we adapted the Qwen3-VL harness from the OSWorld repository.[4] For the Qwen3-VL student model, we use the OSWorld harness directly.

Q. Propose-and-Amplify Ablation: Qualitative Analysis

To understand why seed tasks improve amplified task quality (§2.4), we compare tasks generated with and without seed examples on three representative software applications: Firefox (web browser), AstroImageJ (astronomical image analysis), and Moodle (learning management system). We analyze the differences along three dimensions.

Realism. With seeds, tasks are grounded in professional workflows: Firefox tasks involve researching real websites (Grants.gov, NSF Award Search) from the perspective of specific roles such as an investigative journalist or a development director; AstroImageJ tasks reference real astronomical objects (Eagle Nebula, WASP-12b) and real techniques (photon transfer CCD gain measurement, light curve detrending); Moodle tasks reflect actual institutional operations (grade auditing, custom role creation with specific capabilities). Without seeds, tasks shift toward feature demonstrations: Firefox tasks focus on browser features (importing a CA certificate, exporting a TLS certificate), AstroImageJ tasks become generic processing operations (align images, create a master dark frame), and the domain-specific professional context is largely absent. Moodle is a partial exception, likely because its structure is well-represented in LLM training data.

Difficulty and horizon. With seeds, tasks require 50–80 steps with multi-stage workflows that chain several operations: research, download, organize, and synthesize (Firefox); load data, process, measure, and report (AstroImageJ); configure database, set permissions, and verify via logs (Moodle). Without seeds, tasks typically require 30–50 steps and involve single-feature operations. For instance, a Firefox TLS certificate export task reduces to visiting a webpage, clicking the padlock, and downloading the certificate.

Setup script quality. With seeds, setup scripts range from 50 to 164 lines and perform substantial data preparation: downloading real astronomical data from Hubble archives, generating synthetic FITS files with physically plausible noise models (Poisson noise, read noise, dark current), and populating databases with multi-table SQL inserts. Without seeds, setup scripts are 40–120 lines and often just launch the application and open a URL, with minimal data preparation. This makes tasks more fragile (dependent on external network resources) and less reproducible. One exception is a Firefox custom CA import task (without seeds), which creates a full PKI infrastructure in 140 lines, showing that the model can occasionally produce high-quality setups without seeds, but does so inconsistently.

Summary. The seed tasks teach the amplifier two things the prompt alone cannot: (1) what realistic professional work looks like for a specific software, and (2) how to prepare a rich initial state with real or realistic data. Without these examples, the model falls back to its generic knowledge of the software's features, producing tasks that are simpler, less realistic, and less reproducible.

R. Full Experimental Results and Analyses

R.1. Distillation Ablations

We ablate the teacher model, student model, and number of training steps to identify the best configuration for full distillation.

The strongest teacher does not produce the strongest student. Table 13 compares five teacher models distilled into two student architectures (Qwen3-VL-2B and Qwen2.5-3B) on 4 software. Opus 4.5 is the strongest teacher (53.5 avg. score) while Kimi-K 2.5 is one of the weakest (39.8). However, Kimi-K 2.5 produces the best student for both model sizes: 25.3 vs. 19.3 (Opus) for Qwen3-VL-4B, and 15.8 vs. 9.8 (Sonnet) for Qwen2.5-3B. One possible explanation is that unlike other models, Kimi-K 2.5 is open-source and provides full reasoning chains; however, other factors may contribute as well (Xu et al., 2025d).

Table 13. **Teacher model selection** on 4 software applications.

	Teacher	Student Score	
Teacher	Score	Q3-VL-2B	Q2.5-3B
Opus 4.5	**53.5**	19.3	8.5
Sonnet 4.5	45.5	17.5	9.8
Gemini 3 Flash	44.0	16.3	8.3
Kimi-K 2.5	39.8	**25.3**	**15.8**
Gemini 3 Pro	39.3	15.8	7.0

[4] https://github.com/xlang-ai/OSWorld/blob/main/mm_agents/qwen3vl_agent.py

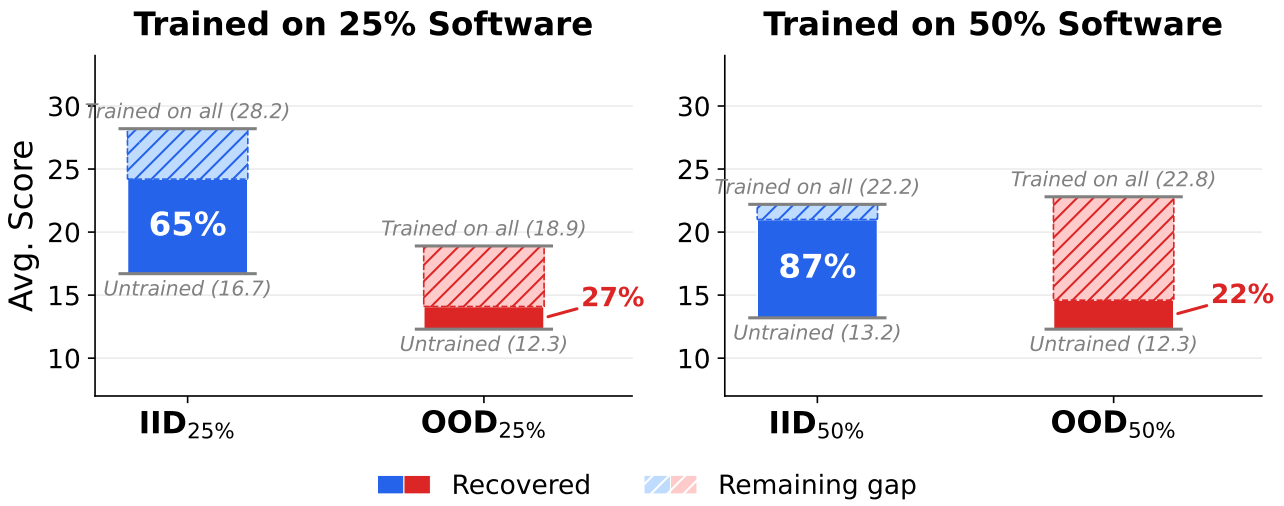


Figure 7. **Generalization to seen (IID) vs. unseen (OOD) software.** We train models on 25% (left) and 50% (right) of the 200 software applications, and evaluate on the training software applications (IID) and the held-out software applications (OOD). Each bar spans from the untrained baseline (bottom) to the model trained on all software (top). The solid portion shows the gain recovered by the model trained on the subset; the hatched portion shows the remaining gap. Training on a subset recovers 65-87% of the gain on IID software but only 22-27% on OOD software, indicating that generalization to unseen software is limited and scaling to diverse software is important.

Table 14. **Effect of training trajectory length.**

Train	GIMP	G. Earth	OpenEMR	Slicer 3D	Avg.
200 steps	60.1	**13.4**	**18.2**	3.3	23.8
50 steps	**64.2**	**13.4**	8.5	**4.5**	22.7
No distillation					
Q3-VL-2B	47.0	7.2	0.0	0.0	13.6

Effect of training trajectory length. Table 14 compares training on the first 50 vs. all 200 steps of each teacher trajectory under the same $25-per-software budget. On average the two settings perform similarly (22.7 vs. 23.8), but the per-software pattern differs: training on 50 steps wins on GIMP and Slicer 3D, while training on 200 steps is better on OpenEMR, likely because its tasks require longer-horizon interaction. Based on this, we adopt a two-stage curriculum for full distillation: first train with a maximum step budget of 50, then continue on full 200-step trajectories, with equal budget for each stage.

R.2. Propose-and-Amplify Ablation

Proposal step substantially improves amplified task quality. To evaluate the propose-and-amplify strategy (§2.4), we compare tasks generated with and without seed examples, using the proposal step, across 10 software applications. We launch each generated task and use a VLM to verify whether the starting state matches the task description. Tasks amplified from seed examples achieve an 88.9% setup success rate, compared to 55.2% without seeds. Qualitative analysis on three software applications (Firefox, AstroImageJ, Moodle) reveals that without seeds, the model defaults to demonstrating software features rather than generating realistic professional workflows, produces shorter-horizon tasks, and writes less thorough setup scripts (Appendix Q).

R.3. Generalization and Software Categories

Distillation improves performance on both seen and unseen software, but gains are larger on seen software. To study how distillation generalizes to software not seen during training, we train models on 25% and 50% of the 200 software applications and evaluate separately on the software used during training (IID) and those that are not used (OOD) (Figure 7). Performance improves on both: on IID software, the average score increases from 16.7 to 24.2 (at 25% of software), and on OOD software from 12.3 to 14.1. However, the OOD gain is limited; Figure 7 shows it recovers only 22-27% of the improvement one would obtain from training on all software, compared to 65-87% on IID software. This suggests that generalization to unseen software does happen but is limited. Secondly, since recovery on IID software ranges from 65%-87% this implies training on a specific software helps substantially, but training on other software also contributes in the evaluated software's performance. Overall, this underscores that building agents for the large variety of software used in the digital economy requires training that is both software-specific and across a diverse set of software, motivating the need

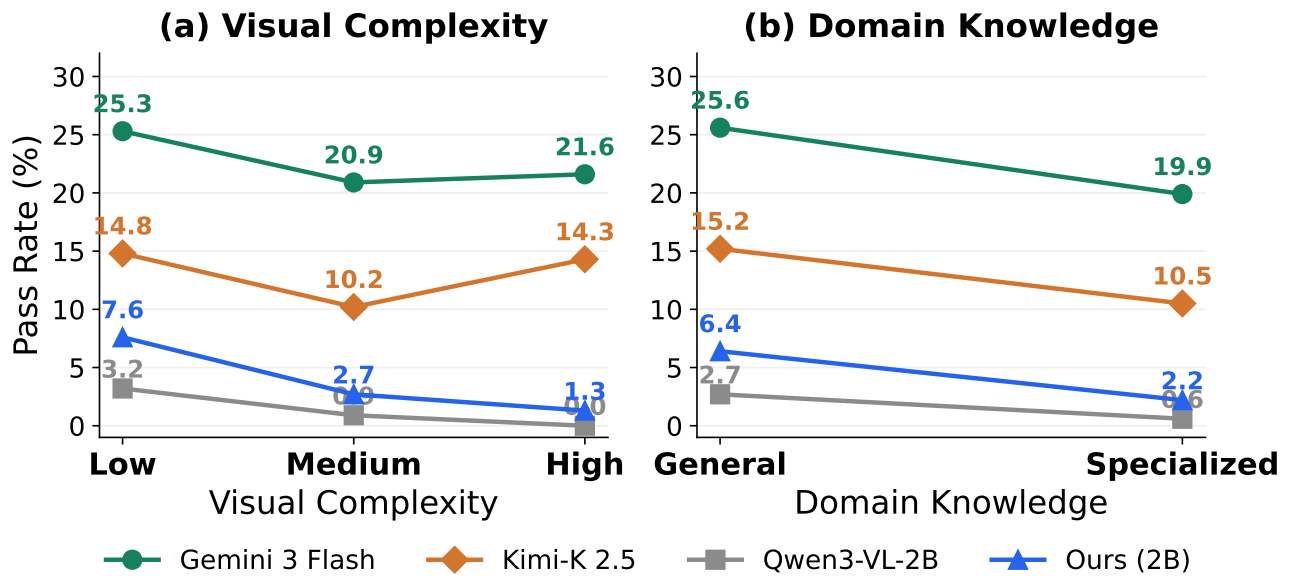
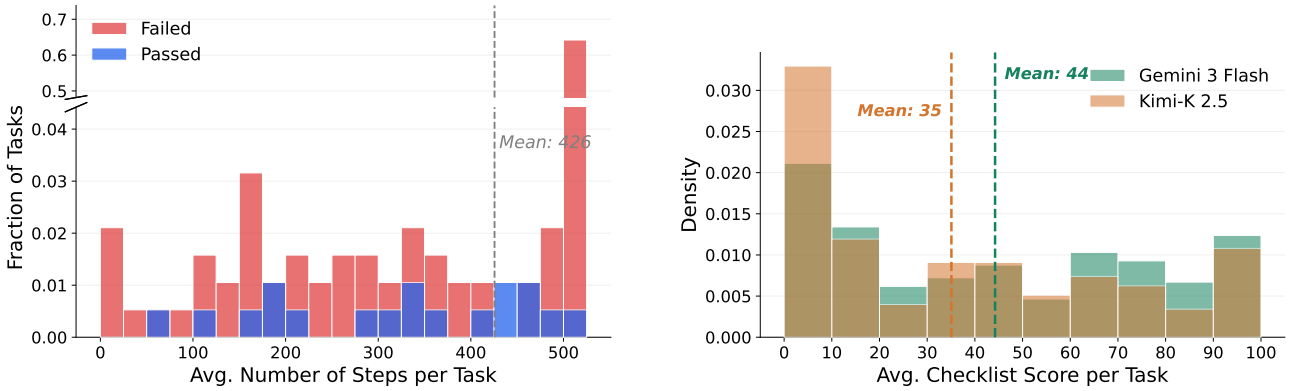


Figure 8. **Pass rate on `CUA-World`-Test by software category. (a)** Visual complexity and **(b)** domain knowledge. See Appendix N for category definitions and assignment of software to categories.



(a) Step distribution.

(b) Difficulty distribution.

Figure 9. **Properties of `CUA-World-Long`. (a)** Distribution of average steps per task. The y-axis is broken to accommodate the spike at the 500-step cap. **(b)** Distribution of per-task average checklist scores for Gemini 3 Flash and Kimi-K 2.5.

for scalable environment creation pipelines such as `Gym-Anything`. Larger models may generalize better across software, which we leave to future work.

High visual complexity is a persistent bottleneck for smaller models. We classify each software along two axes: *visual complexity* (low/medium/high) and *domain knowledge* (general/specialized). Figure 8 shows pass rates on `CUA-World`-Test broken down by each axis. For visual complexity, larger models (Gemini 3 Flash, Kimi-K 2.5) achieve roughly consistent pass rates across all three levels (e.g., 25.3%, 20.9%, 21.6% for Gemini). In contrast, smaller models show a steep decline: Qwen3-VL-2B drops from 3.2% on low-complexity software to 0.0% on high-complexity software. Distillation improves absolute performance at every level (e.g., 0.0% $\rightarrow$ 1.3% on high, 3.2% $\rightarrow$ 7.6% on low), but the decline from low to high remains steep, indicating that visual complexity creates a disparity for small models that distillation alone does not resolve. For domain knowledge, all models show a downward trend from general to specialized software, with smaller models showing a steeper decline ($\sim$3$\times$ for our 2B model: 6.4% $\rightarrow$ 2.2%) than large models ($\sim$1.3$\times$ for Gemini: 25.6% $\rightarrow$ 19.9%).

R.4. CUA-World-Long Step and Difficulty Distributions

Most failed trajectories exhaust the step budget, while passed ones finish at varying lengths. On CUA-World-Long, failed tasks have a large spike at the 500-step cap, indicating that many episodes keep running until the budget is exhausted rather than failing immediately. Passed tasks are spread across a wide range of lengths, including many tasks that still require several hundred steps. The overall mean is 425 steps, highlighting the long-horizon nature of CUA-World-Long. For difficulty, CUA-World-Long spans a wide range: outside of the 0-10 bin, scores are spread fairly evenly across the range, indicating that CUA-World-Long contains tasks at every difficulty level rather than being split into trivially easy and impossible ones. The most notable feature is a large spike at 0-10: roughly a quarter of tasks for Gemini and a third for Kimi receive near-zero scores, indicating complete failure on a substantial fraction of tasks even for the strongest models.

R.5. Verifier Robustness

We evaluate the robustness of the verifier across two dimensions: a.) how well does it agree with humans in terms of correctness, and b.) integrity checks: how often the integrity checklist C_{int} correctly identifies shortcut behavior.

Checklist-based verification achieves highest human agreement. We compare three verifier designs on 60 randomly sampled Gemini-3-Flash trajectories from CUA-World-Test: (1) our checklist-based VLM verifier, (2) a direct VLM verifier that receives the trajectory and outputs a single pass/fail judgment, and (3) programmatic verifiers where a model generates a script that runs on the end state and computes a score. The checklist-based verifier achieves 93.3% task-level agreement with human annotations, compared to 81.7% for the direct VLM verifier and 43.3% for programmatic verifiers. Per-item checklist agreement is 90.9%. The programmatic approach performs poorly primarily because the model writes incorrect scripts that fail to parse the data formats present in the end state; manually authored programmatic verifiers could provide stronger guarantees and are an interesting direction for future work.

Integrity checks catch shortcuts at a low flag rate. Across $\sim$3,000 Gemini-3-Flash trajectories, the integrity checklist flags only $\sim$1.5% of high-scoring runs (score $>$75), producing 21 flags total, of which 15 are true positives. In a forensic analysis task on *Autopsy* (digital forensics tool), the agent followed the correct workflow but fabricated hash values in its final report rather than copying the values visible in the application. In a statistical analysis task on *Epi Info* (epidemiology toolkit), the agent mistyped an input parameter, causing the tool to display an incorrect result, but wrote the mathematically correct answer in its report, a value never shown by the tool. In both cases, the agent scored high on task completion but was zeroed by the integrity check.

R.6. Trajectory Behavioral Patterns

Failed trajectories are dominated by retry loops, while successful ones verify their progress more often. To understand how agents behave on CUA-World, we analyze all trajectories from Gemini-3-Flash evaluated on CUA-World, using an automated behavioral analysis pipeline. We first obtain per-trajectory behavioral summaries via an LLM, then aggregate these across all trajectories to discover recurring behavioral patterns, yielding 15 canonical patterns. **Retry loops** show the largest gap: failed trajectories spend 78% of their steps retrying actions that did not take effect, compared to 39% for passed trajectories. **UI exploration** is high for both outcomes (76% vs. 67%), indicating that the majority of agent effort across all trajectories is spent navigating menus and locating the right controls rather than executing the core task. **Verification checks**, where the agent re-inspects its work after making changes, are present in 91% of passed trajectories but only 70% of failed ones, suggesting an association between self-verification and task success. This observation provides empirical motivation for the Test-Time Auditing approach.

R.7. Cross-Model Auditing

Using a separate model for $\text{Agent}_{\text{audit}}$ catches more issues than self-auditing. To test this, we compare audits where the same model serves as both Agent_C and $\text{Agent}_{\text{audit}}$ (self-audit) against audits where a different model serves as $\text{Agent}_{\text{audit}}$ (cross-model audit) across 10 software applications. Both configurations detect all critical issues, but cross-model audits consistently surface additional problems that self-audits miss. For example, on OpenELIS, the self-audit accepts patient data as realistic, while the cross-model audit inspects the seeding script and identifies the data as hardcoded despite comments claiming real-world WHO/CDC sourcing. Across 10 software applications, cross-model audits identify on average 2.1 additional issues per environment, predominantly low-to-moderate severity.

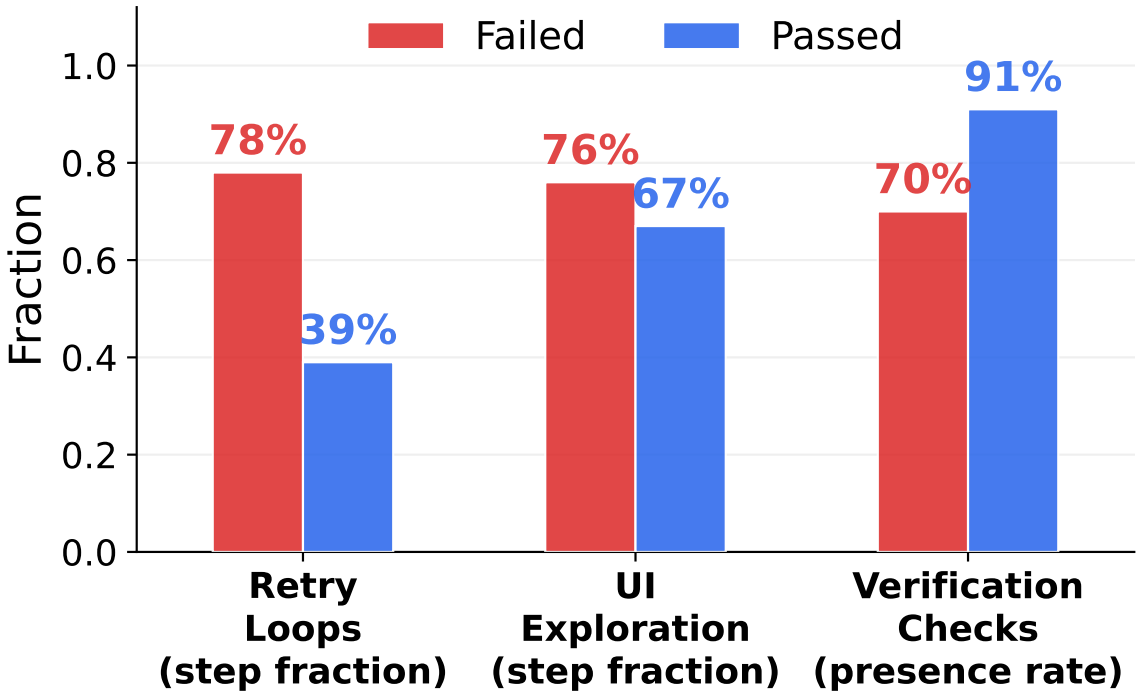


Figure 10. **Behavioral patterns in passed vs. failed trajectories** across Gemini-3-Flash evaluations on `CUA-World`.

S. Trajectory Behavioral Analysis

To understand how agents behave on `CUA-World`, we run an automated behavioral analysis pipeline over 2,981 trajectories (701 passed, 2,280 failed) from our evaluation runs. The pipeline operates in three stages.

Stage 1: Per-trajectory behavioral summary. Each trajectory (the full sequence of screenshots, actions, and model responses) is fed to an LLM, which decomposes the agent's behavior into natural phases and produces a high-level behavioral summary. Crucially, the LLM is not told whether the trajectory passed or failed, and is given no controlled vocabulary or predefined categories—it describes what happened in its own words. This avoids biasing the analysis toward expected failure modes.

Stage 2: Pattern discovery. The per-trajectory summaries are shuffled into random mixed-environment batches and fed to an LLM, which identifies recurring behavioral patterns across each batch. The instruction requires environment-agnostic language: no software names, no UI element names, no application-specific terminology. After all batches are processed, a consolidation pass merges overlapping patterns into a canonical set of 15 deduplicated patterns. Each pattern is a short description of a recurring behavior (e.g., "the agent enters retry loops when actions do not take effect").

Stage 3: Pattern matching. For each trajectory, we send its behavioral summary alongside the 15 canonical patterns to an LLM and ask which patterns are present. Each step in a trajectory can be tagged with multiple patterns (e.g., a step may involve both UI exploration and a retry), so fractions across patterns do not sum to one. For each pattern, we compute two metrics: *step fraction* (what fraction of a trajectory's steps exhibit the pattern, averaged across trajectories) and *presence rate* (what fraction of trajectories exhibit the pattern at least once). Both metrics are computed separately for passed and failed trajectories.

Results. Figure 11 shows the step-fraction view and Figure 12 shows the presence-rate view across all 15 patterns. The three patterns highlighted in the main paper (§3) are *retry loops*, *UI exploration*, and *verification checks*. Beyond these, several additional patterns show notable gaps between passed and failed trajectories. For instance, *access blockers* (authentication failures, unreachable services) occupy 23% of steps in failed trajectories but only 4% in passed ones. *Tool pivoting*—where the agent abandons the primary GUI and switches to CLI or alternative tools—is present in 38% of failed trajectories but only 19% of passed ones. On the positive side, *save/export* steps are present in 52% of passed trajectories but only 33% of failed ones, reflecting that successful agents more often reach the final stage of the task.

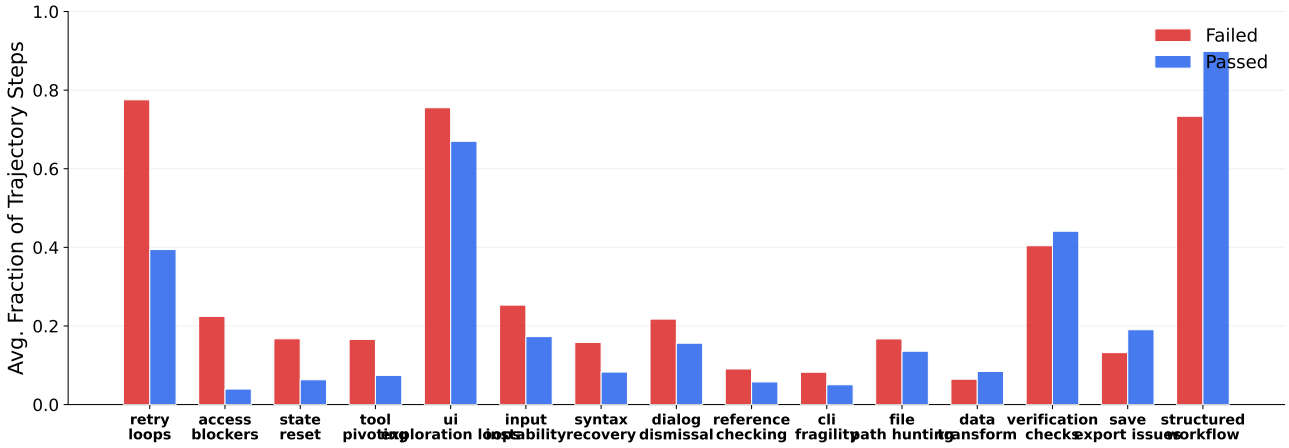


Figure 11. Step-weighted pattern intensity across all 15 discovered behavioral patterns. For each pattern, bars show the average fraction of trajectory steps exhibiting that pattern, split by passed vs. failed trajectories. Patterns are sorted by the gap between failed and passed (largest gap on the left).

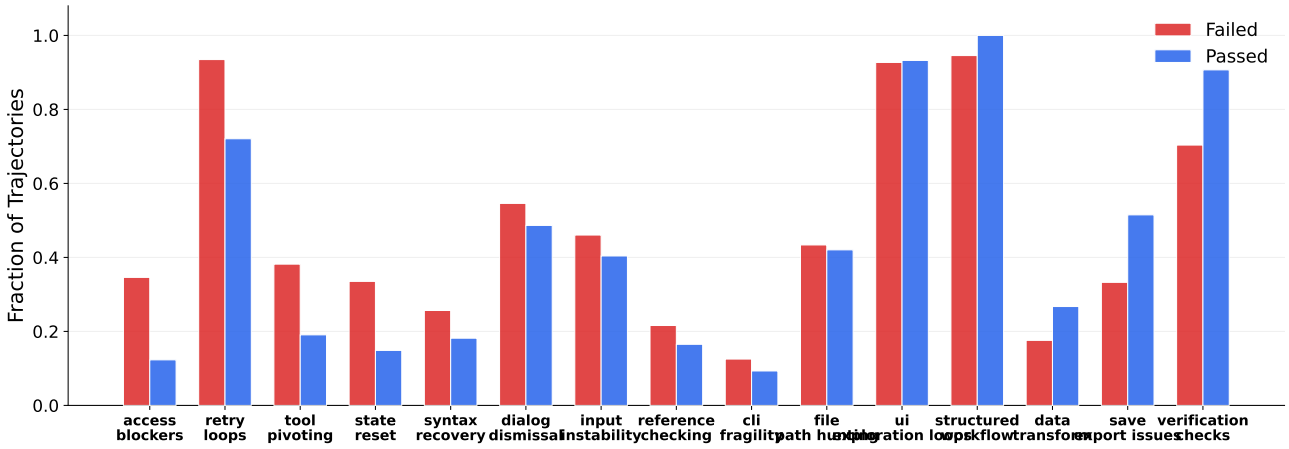


Figure 12. Pattern presence rate across all 15 discovered behavioral patterns. For each pattern, bars show the fraction of trajectories that exhibit the pattern at least once. Patterns are sorted by the gap between failed and passed.

T. Extended Related Work

This appendix expands on the related work discussion in the main paper (§5).

Benchmarks and datasets for computer-use agents. Existing work on evaluating computer-use agents can be divided into static datasets that provide scale and interactive benchmarks that test actual task completion. Static datasets such as Mind2Web (Deng et al., 2023), Android in the Wild (Rawles et al., 2023), and AndroidControl (Li et al., 2024) offer thousands of annotated episodes across hundreds of applications, but evaluation is limited to action-matching against recorded traces rather than execution-based verification, so valid alternative strategies are penalized. Interactive web benchmarks range from synthetic micro-tasks (Liu et al., 2018) to realistic self-hosted environments (Zhou et al., 2024a; Koh et al., 2024; Drouin et al., 2024; Boisvert et al., 2024), but cover at most six websites and are restricted to the browser. On the desktop, OSWorld (Xie et al., 2024) provides 369 tasks across 9 applications on Linux; Windows Agent Arena (Bonatti et al., 2025), Spider2-V (Cao et al., 2024), AssistGUI (Gao et al., 2024), TheAgentCompany (Xu et al., 2025b), and ScienceBoard (Sun et al., 2026) extend coverage to Windows, data science, and scientific domains but remain limited to 100–494 tasks and 5–20 applications each. On mobile, AndroidWorld (Rawles et al., 2025) provides 116 interactive tasks across 20 apps. ProgrammingWithPixels (Aggarwal & Welleck, 2025) scales to 5,400 task instances but within a single application (VS Code). Across prior interactive benchmarks, environment creation is typically manual, which limits their scale, and none simultaneously provides a training split, long-horizon tasks exceeding 100 steps, or broad occupational coverage. `CUA-World` bridges the gap between the scale of static datasets and the execution-based evaluation of interactive

benchmarks: by automating environment creation through the creation-audit loop (§2.3), it provides 10K+ interactive tasks across 200+ software applications on four platforms, with train/test splits, a long-horizon benchmark requiring 200+ steps, and GDP-grounded coverage of all 22 SOC occupation groups.

Automated environment and task generation. A growing body of work generates tasks or trajectories within pre-existing environments. AgentTrek (Xu et al., 2025c) synthesizes web trajectories from online tutorials, OS-Genesis (Sun et al., 2025a) derives tasks retrospectively from agent exploration, and several other methods propose tasks from GUI observations or evolve curricula within fixed environments (Zhou et al., 2024b; Murty et al., 2024; Qi et al., 2025). However, these approaches are bounded by the set of environments that already exist; they cannot create new ones. A parallel line of work uses LLMs to generate environments, but within narrow domains: text-based planning tasks (Hu et al., 2025), 3D indoor scenes for embodied AI (Yang et al., 2024), tool-use API compositions (Wang et al., 2026a), or code-editing setups from GitHub repositories (Pan et al., 2025). None of these targets real GUI software that requires installation, configuration with domain-appropriate data, and interactive verification. For task generation at scale, the seed-then-amplify paradigm is well established: Self-Instruct (Wang et al., 2023) bootstraps instructions from a small seed set, and subsequent works evolve complexity (Xu et al., 2025a) or use multi-agent pipelines (Mitra et al., 2024), but these generate text instruction-response pairs rather than executable environment tasks. `Gym-Anything` addresses all three gaps: its creation-audit loop (§2.3) converts real software into interactive environments via coding agents verified by an independent auditor, its propose-and-amplify strategy (§2.4) generates tasks by having an agent actually run the software to produce high-quality seeds that are then amplified and filtered via execution, and a shared memory across agents ensures learnings accumulate so newer environments are created faster.

Training computer-use agents. Trajectory distillation from strong models has emerged as an effective recipe for training GUI agents: AgentTrek (Xu et al., 2025c) distills from web tutorial replays, Explorer (Pahuja et al., 2025) scales to over 94K web trajectories, and PC Agent-E (He et al., 2026) augments 312 human demonstrations to train a 72B model that surpasses Claude 3.7 Sonnet on WindowsAgentArena-V2. Beyond distillation, alternative training strategies have also shown promise: DigiRL (Bai et al., 2024) achieves a 49.5-point improvement over SFT with a 1.3B model, and UI-TARS (Qin et al., 2025) combines enhanced perception, unified action modeling, and iterative reflective training to achieve state-of-the-art results across multiple benchmarks. Open vision-language backbones such as Qwen2-VL (Wang et al., 2024) and Qwen2.5-VL (Bai et al., 2025) are common foundations for recent open GUI-agent systems (Wu et al., 2024; Wang et al., 2025). However, existing training pipelines are typically limited to relatively small sets of applications, and scaling laws for GUI agent trajectory distillation remain underexplored. Our distillation experiments across 200 software applications show log-linear scaling ($\sim$3.5 points per data doubling), demonstrate that a 2B student can outperform models $2\times$ its size, and reveal that cross-software generalization is limited (22–27% recovery), motivating scalable environment creation.

Evaluation of computer-use agents. Existing interactive benchmarks often use hand-written programmatic verifiers that check the final system state (Xie et al., 2024; Zhou et al., 2024a), which can be reliable but are labor-intensive to author and maintain and typically provide binary pass/fail. TheAgentCompany (Xu et al., 2025b) introduces checkpoint-based partial credit but still requires custom evaluator code per task. The LLM-as-a-judge paradigm (Zheng et al., 2023) and its extensions to agent evaluation (Zhuge et al., 2024) offer a more general alternative to script-based evaluation. VLM-based evaluation has been explored in the CUA setting for filtering training trajectories (Xu et al., 2025c), step-level trajectory assessment (Sun et al., 2025b), and autonomous trajectory evaluation (Pan et al., 2024), while checklist-based evaluation has shown strong correlation with human preference for text generation (Lin et al., 2024) and per-subgoal VLM evaluation has been explored in robotics (ElMallah et al., 2025). Our checklist-based VLM verifier extends this line of work by incorporating privileged information extracted from environment setup scripts, enabling the verifier to check agent outputs against known ground-truth answers without per-task evaluation code. We additionally introduce integrity checks that detect workflow bypasses such as fabricating report data or using the terminal instead of the intended GUI, an issue related to the broader problem of reward hacking in agent evaluation (Gabor et al., 2025).

Economic impact of AI and occupation-grounded benchmarks. A substantial body of work studies which occupations are susceptible to AI automation, starting from Frey and Osborne's occupation-level risk estimates (Frey & Osborne, 2017) and Eloundou et al.'s O*NET-based LLM exposure framework (Eloundou et al., 2023). Felten et al. (Felten et al., 2021) introduced the widely adopted AI Occupational Exposure index, and Acemoglu (Acemoglu, 2024) formalized GDP-level impact estimation via task-level cost savings. These studies use occupational data, often from O*NET, to connect AI capabilities to labor-market outcomes, but focus on measuring exposure rather than directing benchmark design. Wang et al. (Wang et al., 2026b) recently quantified this gap, finding that across 43 agent benchmarks and 72K tasks, coverage is heavily programming-centric, with much less representation in many economically significant domains outside

computing. GDPval (Patwardhan et al., 2025) takes a step toward economic grounding by evaluating models on tasks from 44 occupations across 9 GDP-contributing industries, but is limited to one-shot evaluation rather than interactive agentic tasks. `Gym-Anything` inverts the standard direction: rather than measuring which occupations are exposed to AI, it uses per-software GDP attribution to determine which software to include in an agent benchmark, covering all 22 SOC major occupation groups with interactive, execution-verified tasks.