JINGDONG

CUA-Master: Scaling Computer-use Agents through Open-World Recursive Improvements

Xinlei Yu, **Gen Li**, **Ji Zhao**, **Zhangquan Chen**, **Congcong Wang**, **Shuai Dong**, **Zelin Peng**, **Yuqi Xu**, **Hongrui Wu**, **Kaiwen Tuo**, **Cheng Yang**, **Kaituo Feng**, **Yifan Wu**, **Jiayi Zhang**, **Dongming Wu**, **Wenqi Shao**, **Jiaqi Wang**, **Xiangyu Yue**✉

MMLab, CUHK THU HKUST(GZ) Explore Academy, JD

Project Code Model

Abstract

Despite recent advances in CUAs, their development remains constrained by the limited diversity, controllability, and reliability of interaction data. Moreover, conventional optimization pipelines lack iterative feedback, while existing recursive improvement approaches often rely on self-generated experiences, limiting ability expansion and yielding diminishing returns. To address these challenges, we introduce **CUA-Master**, a nested double-loop recursive improvement framework. At the system level, an outer world–data–model flywheel continuously incorporates open-world corpus and leverages evolving model weaknesses to guide subsequent data construction and optimization iterations. At the data level, an inner flywheel couples open-world seeding and autocatalytic propagation. The former mines authentic workflows as seed tasks from instructional videos, and employs visual-semantic dual-path trajectory reconstruction; The latter transforms these seeds into reusable primitives for controllable synthesis of diverse, complex, and verifiable tasks. To enable efficient and reliable trajectory collection, we further introduce a feedback-guided snapshot-rollback retry mechanism in various real environments. Extensive experiments demonstrate that **CUA-Master-27B** achieves competitive performance across GUI-only and GUI–CLI hybrid benchmarks, scoring **80.2** on OSWorld Verified, **36.9** on OSWorld 2.0, **43.9** on WeaveBench, and **25.0** on Agents' Last Exam. Our framework also substantially improves the quality and efficiency of data synthesis, and exhibits continuous gains across recursive iterations.

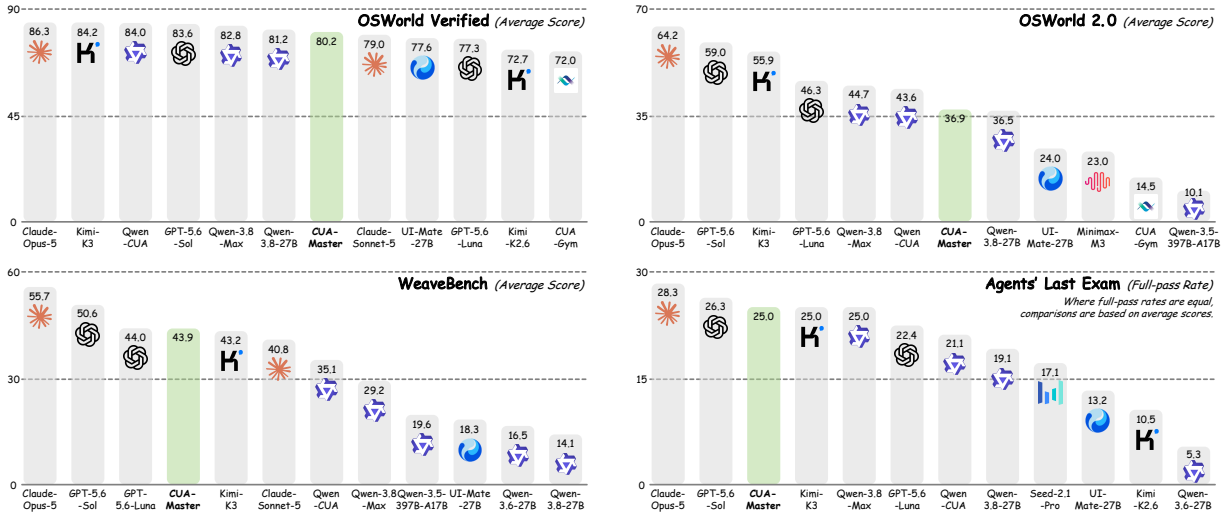


Figure 1 Performance comparisons of our proposed CUA-Master-27B and other counterparts across four benchmarks.

1 Introduction

Computer-use agents (CUAs) have emerged as a promising paradigm for automating real-world workflows across diverse software applications, and unlocking new levels of digital productivity. Recent advances in both foundation models [3, 30, 32, 37, 40, 47] and specialized agents [9, 17, 28, 49, 51] have substantially improved their capabilities not only grounding of computer scenarios, but also in interactive reasoning and long-horizon execution. Besides, early CUAs operate predominantly via graphical user interfaces (GUIs), while subsequent works extend their capability scope by incorporating command-line interfaces (CLIs) [8, 24, 42, 45] and extensible tools [16, 19]. Nevertheless, developing powerful CUAs capable of reliably executing diverse, long-horizon, and cross-application workflows remains challenging.

A fundamental bottleneck lies in how data construction and model optimization are coupled and recursively improved. Many existing pipelines follow a **unidirectional** paradigm of data generation followed by model training [18, 28, 49, 51], where data construction is insufficiently informed by the evolving weaknesses. Consequently, additional data may provide lopsided benefits without addressing persistent capability gaps. Recent recursive improvement approaches partially address this issue by incorporating iteratively self-generated data [33, 50, 73]. However, their exploration remains **closed-world** optimization, where such cycles still predominantly rely on the own knowledge and capabilities of the model itself. It is bounded by existing capabilities, potentially reinforcing distributional biases, and yielding diminishing marginal returns. These observations fundamentally motivate us to build a recursive system that continuously acquires new open-world knowledge, and systematically expands the performance boundary of the whole system.

Beyond system-level limitations, a more concrete concern is the availability of large-scale, high-quality data that adequately captures the diversity and complexity of real-world trajectories. While human-annotated trajectories provide high-quality supervision, their substantial annotation costs and limited application coverage make large-scale collection impractical [13, 20, 25, 50, 51]. Therefore, synthetic data generation emerges as a promising alternative [21, 39, 41, 44, 59]. However, existing approaches face three intertwined challenges. First, **diversity**: tasks derived from restricted and similar sources often exhibit substantial homogeneity, leading to insufficient ability coverage and fast improvement saturation as data volume increases. Second, **controllability**: direct instruction rewriting and template-based generation offer limited control over operational abilities, complexity, and task dependencies, particularly for long-horizon and cross-application workflows. Third, **reliability**: scaling the collection of verified trajectories across diverse real-world environments becomes increasingly challenging as task complexity grows, leading to lower rollout success rates and frequent execution failures. Collectively, these limitations constrain not only the quantity of synthetic data, but also its effective coverage of the capabilities required by general-purpose CUAs.

To address these limitations, we introduce CUA-Master, a nested double-loop recursive framework that integrates an outer world–data–model flywheel with an inner data flywheel. At the system level, the outer flywheel closes the loop among open-world exploration, data construction, and model optimization. By leveraging the evolving model's capability gaps to guide subsequent exploration and data generation, while continuously incorporating external knowledge, it addresses both the unidirectionality of conventional training pipelines and the limited knowledge expansion of closed-world self-improvement. At the data level, the inner flywheel couples open-world seeding and autocatalytic propagation as two complementary driving forces. Open-world seeding mines diverse seed tasks from real-world instructional videos through hierarchical retrieval and filtering, and reconstructs their trajectories via a visual-semantic dual-path mechanism. The latter abstracts existing seeds into reusable primitives and composes them into verifiable tasks, with explicit control over task complexity and application scope. To improve rollout reliability and efficiency, we introduce a feedback-guided snapshot-rollback retry mechanism that preserves valid execution prefixes and recovers from localized failures without restarting entire rollouts.

Extensive experiments demonstrate the effectiveness of CUA-Master. With only **27B** parameters, it achieves scores of **80.2** on OSWorld Verified, **36.9** on OSWorld 2.0, **43.9** on WeaveBench, and **25.0** on Agents' Last Exam, demonstrating competitive performance approaching markedly larger or closed-source models and strong execution efficiency. Our double-loop flywheel also substantially improves the diversity and coverage of data distribution, reliability and controllability of synthesis, and progressive improvements across iterations.

2 Methodology

2.1 Preliminary

2.1.1 Task Definition

We consider interactive computer-use tasks where a general or CUA-targeted agent executes natural-language instructions through sequential interactions with computer environments. Formally, each task is defined as:

$$\mathcal{T} = (\mathcal{I}, \mathcal{E}, \mathcal{V}), \tag{1}$$

where $\mathcal{I}$ denotes the task instruction, $\mathcal{E}$ represents the interactive environment, and $\mathcal{V}$ is a task-specific verifier that determines whether the intended objective has been successfully achieved.

During execution, at each decision step s, the agent receives an observation o_s from the environment of the current environment state and generates a sequence of one or more actions:

$$\mathbf{a}_s \sim \Pi(\cdot|\mathcal{I}, o_s, h_s), \tag{2}$$

where Π denotes the agent policy, h_s represents the preceding interaction history, and $\mathbf{a}_s = (a_s^{(1)}, \ldots, a_s^{(L_s)})$ denotes the predicted atomic action sequence of length L_s. When $L_s = 1$, the model performs single action, while $L_s > 1$ denotes a batched action sequence. Executing an action batch $\mathbf{a}_s$ updates the environment state and produces the subsequent observation o_{s+1}. This iterative process continues until the agent terminates or reaches step budgets, resulting in the complete trajectory:

$$\mathcal{X} = \{o_1, \mathbf{a}_1, \ldots, o_S, \mathbf{a}_S, o_{S+1}\}, \tag{3}$$

where S indicates the total number of execution steps and corresponding observations. Finally, the task-specific verifier $\mathcal{V}$ evaluates whether the execution outcome satisfies the objective specified by $\mathcal{I}$, thereby providing an assessment of task completion.

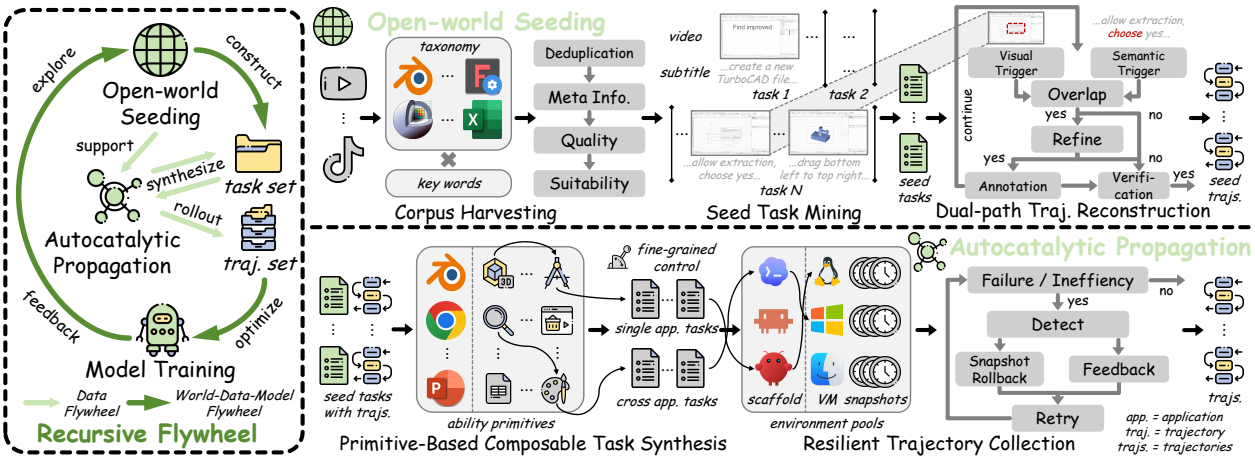
2.1.2 Hybrid Action Space

We adopt a unified atomic action space that supports both direct GUI, *e.g.*, `computer.click`/`computer.scroll`, and structured CLI operations `shell.exec`. Each atomic action is represented as a by its action type and a dictionary containing its arguments. As listed in Table 7, the action space is partitioned into GUI and CLI actions, where the former consists of atomic mouse and keyboard operations, while the latter executes structured shell commands. This pattern allows a unified execution interface and can be interleaved within a trajectory, allowing the agent to select the action most appropriate for the current environment state. The environment returns observations o after execution to capture the action outcome, thus, GUI interactions are grounded through screenshots, while CLI interactions are grounded through standardized shell outputs containing the command output, error stream, and execution status. All pointer coordinates are represented in a resolution-independent normalized coordinate system: $x, y \in [0, 1000]$. In addition, our hybrid action space is also compatible with additional `tool.call`, affording flexible tool invocation capabilities.

2.1.3 Environment Infrastructure

Our rollout infrastructure is a backend-agnostic, profile-driven orchestration layer for reproducible CUA tasks across Linux, Windows, and MacOS. We construct the VM pool through a semi-automated assembly pipeline. First, we manually prepare a set of base images spanning the application combinations defined in Table 9. We then programmatically derive additional variants by perturbing these images, *e.g.*, by removing randomly selected applications and reorganizing desktop layouts, thereby expanding environment coverage at low marginal cost. Each generated image is registered in a catalog and accompanied by a descriptor recording its operating-system type and version, and application inventory, including version and package metadata.

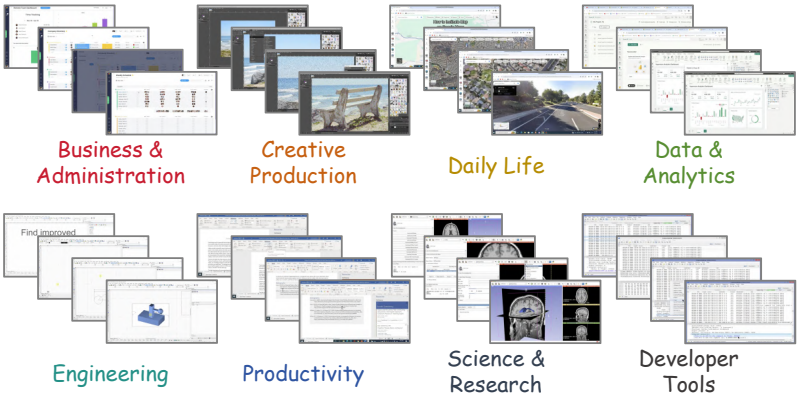
To better manage and allocate CPUs and other resources, we employ a global resource-aware scheduler that bounds concurrency by occupied VM slots, while standardized lifecycle hooks manage resource acquisition,



(a) Pipeline of double-loop recursive flywheel (dashed black box), integrating open-world seeding (upper right), and autocatalytic propagation (lower right).



(b) Hierarchical application taxonomy.



(c) Representative applications across the eight selected domains.

Figure 2 Overview of our proposed **(a)** CUA-Master recursive flywheel; **(b)** hierarchical domain/subdomain/application taxonomy, and **(c)** representative applications covered by the taxonomy.

controller readiness, network tunneling, environment reset, and cleanup. During rollout, each episode is instantiated from a versioned VM image catalog, typically using an immutable base image with a disposable writable overlay. The writable overlay accommodates harness bootstrapping (Codex, Claude Code, OpenClaw, *etc.*), task-specific initialization, and transient application states. This infrastructure supports over 100 different CUA environments and allows over 2,000 concurrently running VMs, substantially increasing the richness, diversity, and efficiency of large-scale rollout.

2.2 Recursive Flywheel

2.2.1 Data Flywheel

As illustrated by the black dashed box in Figure 2a, we formulate our methodology as a recursive flywheel driven by two complementary forces. **Open-world seeding** (Section 2.3) continuously injects real-world diversity and operational knowledge by discovering underrepresented applications, operations, and workflows, mitigating the risks of closed-loop degeneration associated with purely self-generated data [11, 43, 71]. **Autocatalytic propagation** (Section 2.4) transforms these real-world seeds into reusable primitives, enabling composable task synthesis and reliable trajectory collection in executable environments. Together, these two forces establish a self-reinforcing cycle in which **open-world seeding provides external knowledge and diversity and forms the necessary foundation, while autocatalytic propagation systematically**

amplifies its coverage, complexity, and scale.

2.2.2 World-Data-Model Flywheel

Beyond the inner data flywheel, we establish a broader world–data–model flywheel. Starting from the open world, diverse applications, operations, and workflows are distilled and expanded into executable tasks and trajectories, providing grounded supervision for model training. As the model evolves, its emerging capabilities and remaining limitations provide feedback to guide subsequent open-world exploration, directing data acquisition and scaling toward underrepresented workflows and increasingly challenging tasks. Through this iterative cycle of **world exploration, data construction, model improvement, and feedback-driven re-exploration, the agent progressively expands its capability frontier.**

2.3 Open-world Seeding

As discussed above, one of the most prominent limitations lies in the insufficient richness of tasks. Open-world video resources, *e.g.*, YouTube and TikTok, contain a wealth of software demonstration videos, which can be leveraged to enrich task diversity. However, their supervision is mostly implicit, temporally diffuse, and often contaminated by non-operational content. We therefore introduce an open-world seeding pipeline that distills instructional video into both real-world seed task $\mathcal{T}^{seed}$ and corresponding trajectory $\mathcal{X}^{seed}$.

Compared to previous works [27, 57, 70], our key insight is to **regard this process not merely as direct trajectory generation, but also as seed task collection and a foundation for further propagation**. Besides, we introduce a **visual-semantic dual-path reconstruction mechanism** to substantially improve the accuracy of trajectory reconstruction.

2.3.1 Corpus Harvesting

We organize the open-world search space using a hierarchical application taxonomy as listed in Figure 2b, 2c, and Table 9, including 8 main categories, 36 subdomains, and over 650 applications; and a set of operation-oriented keywords, *e.g.*, "tutorial", "how to", "guide", *etc.* For every application, we instantiate compositional queries to retrieve videos from the open-world internet video databases, namely YouTube, TikTok, and Bilibili. To remove low-quality and unpromising candidates, all retrieved videos are first deduplicated by source identity and normalized title, followed by metadata and noise filtering. Specifically, metadata gates enforce admissible video duration, upload recency, source availability, and taxonomy-consistent application attribution; a lightweight noise filter further suppresses promotional, entertainment, language-learning, and other incidental mentions of software that do not demonstrate CUA interactions.

To further filter unsuitable candidates, the audio of uniformly sampled frames $\{f_1, \ldots, f_n\}$ are labeled by the Whisper-Large-v3 model through `Faster-Whisper` [38] with timestamped transcript and taxonomy metadata as $\{\hat{f}_1, \ldots, \hat{f}_n\}$. Then, we use an annotator $\mathcal{A}$, normally a more powerful model, as the score evaluator (details in Figure 11) to assign an overall CUA suitability score $\sigma \in \{0, \ldots, 5\}$ by assessing whether the video presents a legible computer-use interaction and a reconstructible step-by-step interaction trajectory. It additionally predicts CUA presence, trajectory availability, and visual clarity, providing structured signals for subsequent corpus curation.

2.3.2 Seed Task Mining

A tutorial video may contain multiple workflows interleaved with non-CUA fragments that are not suitable for task collection. We therefore formulate seed task mining as multimodal temporal episode discovery. Given metadata, the timestamped transcript, and uniformly sampled overview frames $\{f_1, \ldots, f_n\}$, we use an annotator $\mathcal{A}$ as task generator (details in Figure 12) to decompose and segment independent tasks from a complete video:

$$\{\hat{f}_1, \ldots, \hat{f}_n\} \xrightarrow{\mathcal{A}} \{(t_i^s, t_i^e, \mathcal{T}_i)\}_{i=1}^{M}, \tag{4}$$

where M denotes the number of segmented tasks, and t^s and t^e are the start and end time stamps of the segmented task, retaining only observable computer-operation state-continuous intervals and excluding

introductions, advertisements, talking-head commentary, slide-only explanations, and other non-operational content. Consequently, a single tutorial video can yield multiple independently executable seed tasks. Each episode is assigned a concise goal inferred jointly from its visual context, narration, application identity, and demonstrated state transition. This transforms a coarse video-level corpus into task-level supervision.

2.3.3 Dual-path Trajectory Reconstruction

Given each mined seed task and its temporal scope, we reconstruct its trajectory through a visual-semantic dual-path trigger, which is low-cost and efficient, and recheck mechanism to refine annotations. Specifically, the semantic path first decomposes each timestamped transcript segment into atomic clauses according to discourse connectives and temporal markers, *e.g.*, "then", "next", and "after that". It further identifies operation-bearing clauses through action expressions, such as "click", "press", "scroll", and "drag", as semantic triggers: $\mathcal{C}^{\text{sem}} = \{t_1^{sem}, \ldots, t_m^{sem}\}$. In parallel, the visual path is activated by `FFmpeg` package when frame-level appearance-change scores exceed a predefined threshold as visual triggers: $\mathcal{C}^{\text{vis}} = \{t_1^{vis}, \ldots, t_m^{vis}\}$, indicating a potentially meaningful interface transition, such as a menu opening, dialog appearance, page navigation, or content update. We temporally detect the overlap of the two triggers into a boundary lattice, and obtain the final trigger candidates:

$$\mathcal{C} = \left\{ \left(t_i^{\text{sem}}, t_j^{\text{vis}} \right) \middle| |t_i^{sem} - t_j^{vis}| \le T \wedge t_i^{sem} \in \mathcal{C}^{sem} \wedge t_j^{vis} \in \mathcal{C}^{vis} \right\}, \tag{5}$$

where T denotes the predefined temporal window, judging whether two intervals are overlapped to trigger.

The annotator $\mathcal{A}$ as action rechecker jointly examines the local transcript context and paired pre- and post-action frames to refine each candidate (details in Figure 13), which rejects non-operational changes, and optimizes the action boundaries by adding, removing, splitting, or merging candidates. Accepted candidates are then processed by $\mathcal{A}$ as action annotator into CUA actions in batches (details in Figure 14), outputting interaction coordinates, typed content, and other necessary keys. Finally, we use $\mathcal{A}$ as trajectory verifier (details in Figure 15), to remove duplicated actions, split sequences at substantial temporal or semantic discontinuities, and retain qualified trajectories.

2.4 Autocatalytic Propagation

Open-world seeding introduces diverse real-world workflows, but the resulting seed tasks provide limited coverage of the combinatorial space of computer-use operations. Moreover, although annotation-verified, trajectories reconstructed from instructional videos may contain annotation noise and do not necessarily correspond to real executions in the target environments. Moreover, existing CUA task and trajectory synthesis methods often rely on rigid generation or rewriting strategies, potentially leading to homogeneous outputs [7, 28, 49, 60]. Other lines might suffer from unverifiable synthesis [21, 44, 55], and poorly reliable rollout pipeline [23, 25, 60], limiting the potential to systematically expand diverse and increasingly flexible workflows across applications.

Therefore, we introduce autocatalytic propagation, which successively performs primitivization, composable task synthesis, and resilient trajectory collection. Our key insight is to **treat the real-world rich application operations as composable primitives, which can be flexibly combined to generate tasks under fine-grained control**, *e.g.*, single- or cross-application, and task complexity. Besides, the resilient trajectory collection framework with **feedback-guided snapshot-rollback retry mechanism** notably enhances the success rate and efficiency of rollouts.

2.4.1 Primitive-Based Composable Task Synthesis

Directly generating or rewriting seed instructions provides limited control and quality over task semantics and potentially results in homogeneous tasks when scaling up. We instead primitivize each seed task $\mathcal{T}$ and its associated trajectory $\mathcal{X}$ into reusable, application-level ability primitive sequence, using the annotator $\mathcal{A}$ as primitive extractor (details in Figure 16), which is represented as:

$$\left(\mathcal{T}^{seed}, \mathcal{X}^{seed}\right) \xrightarrow{\mathcal{A}} \{p_1, \ldots, p_n\}, \text{where } p_i = (u_i, a_i, \mathcal{P}_i, \mathcal{E}_i, \mathcal{V}_i), \tag{6}$$

where n denotes the number of generated primitives, each of which is a five-tuple comprising the associated application u, the reusable ability a, execution preconditions $\mathcal{P}$, environment requirements $\mathcal{E}$, and verifier specification $\mathcal{V}$, to formulate the operational dependencies required for execution and verification. Here, we follow and further improve the strategies from [28, 52] to automatically build corresponding $\mathcal{E}$ and $\mathcal{V}$. We aggregate the extracted primitives into an application-indexed primitive library $\mathcal{B}$. Semantically equivalent primitives are canonicalized and deduplicated, while variants with distinct preconditions, environment requirements, or verification logic are retained. This representation aims to abstract away task-specific entities while preserving the operational dependencies necessary for reliable execution, enabling a primitive to be instantiated with different files, content, interface states, and application configurations.

Task synthesis is formulated as constrained primitive composition. Given a target complexity budget K and an application set $\{u_1, \ldots, u_N\}$, task complexity can be explicitly controlled along two dimensions: the number of composed primitives K and the number of involved applications N, rather than through unconstrained linguistic rewriting. For each compatible composition, we use the annotator $\mathcal{A}$ to instantiate a synthetic task $\mathcal{T}^{syn}$ (details in Figure 17) with a coherent instruction $\mathcal{I}^{syn}$. It simultaneously composes their environment requirements $\{\mathcal{E}_1, \ldots, \mathcal{E}_K\}$ into an executable environment specification $\mathcal{E}^{syn}$ and consolidates the primitive-level verifier specifications $\{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ into a task-level verifier $\mathcal{V}^{syn}$:

$$(\mathcal{B}(\{u_1, \ldots, u_N\}), K) \xrightarrow{\mathcal{A}} \mathcal{T}^{syn} = (\mathcal{I}^{syn}, \mathcal{E}^{syn}, \mathcal{V}^{syn}), \tag{7}$$

where $\mathcal{B}(\cdot)$ provides the primitives of selected applications. When composing primitives, we first check compatibility of their preconditions $\mathcal{P}_i$ and environment requirements $\mathcal{E}_i$. Primitives with conflicting preconditions or incompatible environment states cannot be composed together. The resulting synthetic precondition is the conjunction of compatible primitive preconditions. When $N = 1$, a single-application task is generated, whereas $N > 1$ yields a cross-application task. Based on the selection strategy, we include high-quality task $\mathcal{T}^{syn-hq}$ into a task set for particular model training stage.

2.4.2 Resilient Trajectory Collection

Simply discarding a failed rollout and restarting from scratch is inefficient, since a failure is often caused by a localized misoperation while most of the preceding trajectory remains valid. Besides, even successful trajectories may contain redundant and unnecessary actions, repeat observations, and verbose reasoning, introducing unnecessary execution overhead and inefficient supervision. We therefore introduce a feedback-guided snapshot-rollback retry mechanism. For each arbitrary task $\mathcal{T}$, the environment infrastructure (Section 2.1.3) first materializes an environment by selecting a compatible image from the VM pool and initializes necessary configurations to rollout trajectory. When verification fails or inefficiency is detected, the annotator $\mathcal{A}$ as detector (details in Figure 18) jointly examines the verifier residuals $\mathcal{V}$, collected trajectory $\mathcal{X}$, and executable environment $\mathcal{E}$, thus the feedback is generated as:

$$(\mathcal{T}, \mathcal{X}, \mathcal{E}) \xrightarrow{\mathcal{A}} (\hat{s}, \mathcal{F}), \tag{8}$$

where $\hat{s}$ estimates the earliest state-divergence step, and $\mathcal{F}$ produces execution-grounded feedback. The feedback describes the observed failures or inefficiency, without exposing a privileged action solution, alleviating potential distribution mismatch [29, 66].

Subsequently, we restore the environment to the latest valid snapshot preceding $\hat{s}$. Inspired and further modified by [6], rather than replaying whole history from the initial snapshot, we create incremental snapshots at stable boundaries. Each snapshot stores the VM state together with a lightweight state signature, including active applications, window layout, relevant files, and controller metadata. The agent policy Π preserves the stable interaction prefix, discards the undesirable suffix, and resumes execution conditioned on the diagnostic feedback as hints. This process continues until $\mathcal{V}$ reports pass or a predefined retry budget is exhausted. Only the successful and the most efficient branch is retained as the final trajectory $\mathcal{X}$, while failed or inefficient suffixes are excluded to avoid introducing state-inconsistent supervision. Thus, retries form a lightweight branching search that reuses successful prefixes and significantly enhances the efficiency of large-scale rollouts.

2.5 Model Training

2.5.1 Recipe

As discussed in Section 2.2, we integrate model training into the recursive world-data-model flywheel, where each iteration comprises a complete model optimization. Specifically, preceding iterations employ SFT to progressively acquire diverse knowledge and capabilities. In each iteration, based on the in-house assessment of model, including domain- and application-level hierarchical failure cases and rollout success rates analyses, we identify the gaps and limitations of the current policy, guiding the next iteration of adaptive data scaling and the CUA ability frontier.

As capability gains from SFT gradually saturate, we transition to RL in the final iteration to further enhance the task-level capability upper limit. As discussed, the synthesized tasks are built upon controllable capability primitives, which renders native support for difficulty-level categorization. Combining this with task generation quality and rollout success rates, we curate a high-quality task collection for RL training.

2.5.2 Supervised Fine-Tuning (SFT)

At the ith SFT iteration turn, we fine-tune the policy on verified trajectories $\bigcup_{t=1}^{i}\left(\mathcal{X}_t^{seed} \cup \mathcal{X}_t^{syn}\right)$ collected through both open-world seeding and autocatalytic propagation. Training follows an autoregressive objective over executable action sequences, conditioned on task instructions and interaction histories. Following previous CUA works [26, 46, 60], to avoid potential optimization drift, each iteration of SFT is trained starting from the same base checkpoint, with the updated data from each iteration added to the original training set.

2.5.3 Reinforcement Learning (RL)

To break through the capability plateau, we further optimize the SFT-trained policy on the high-quality synthetic tasks (Section 2.4.1) through RL in real executable computer environments. Here, we mainly employ and optimize the efficient online RL framework for CUAs [28]. Leveraging the task-specific verifiers established during task synthesis, we derive outcome-based rewards directly from execution results, without additional manual reward annotation. This final stage builds upon broad operational competence and focuses on improving long-horizon decision-making, execution robustness, and task completion.

3 Experiments

3.1 Settings

3.1.1 Baselines

We compare CUA-Master against a diverse set of generalist and specialist models. The generalist baselines include open-source models: Qwen-3.5-27B/-397B-A17B [35], Qwen-3.6-27B [36], Qwen-3.8-27B/Max [37], Minimax-M3-428B-A23B [30], Kimi-K2.6-1T-A32B [31], and Kimi-K3-2.8T-A104B [47], as well as proprietary models: Seed-2.1-Pro [40], GPT-5.6-Luna/Sol [32], and Claude-Sonnet/Opus-5 [3, 4]. For specialist baselines, we consider OpenCUA-72B [51], CUA-Gym-397B-A17B [49], EvoCUA-1.5-32B [17], ScaleCUA-9B [28], Qwen-CUA-397B-A17B [26], and UI-Mate-27B [9], which are specifically developed for computer-use tasks. This comparison enables us to evaluate CUA-Master against both general foundation and dedicated CUA models.

For fair comparison, we re-evaluate all baseline models under a consistent benchmark protocol, using the same environment configurations, action interfaces, and evaluation criteria, unless otherwise specified.

3.1.2 Benchmarks

We primarily evaluate CUA-Master on four challenging CUA benchmarks, including two GUI-only benchmarks: OSWorld Verified [56], OSWorld 2.0 [68], and two under a hybrid GUI+CLI setting: WeaveBench [24], and Agents' Last Exam [45]. Besides, we additionally evaluate three cross-platform benchmarks: WindowsAgentArena [5], MacOSWorld [63], and MacAgentBench [14] , to assess cross-platform generalization. For Agents' Last Exam, we calculate the full-pass rate, while reporting average scores for the other three

Table 1 Main results across benchmarks. Step, Token, Time, and Score represent the per-task averages of execution steps, total token usage, wall-clock runtime, and task scores. ⋆ denotes closed-source models, while others are open-source. † indicates benchmarks evaluated by GUI mode, while ‡ are using GUI+CLI hybrid assessment.

Method	OSWorld Verified†				OSWorld 2.0†				WeaveBench‡				Agents' Last Exam‡			
	Step	Token	Time	Score	Step	Token	Time	Score	Step	Token	Time	Score	Step	Token	Time	Score
Generalist																
Qwen-3.5-27B [35]	32.2	396K	7m	54.6	224.0	1.9M	1h48m	0.8	75.6	1.0M	41m	13.5	165.3	4.1M	1h25m	1.3
Qwen-3.5-397B-A17B [35]	32.8	412K	11m	61.0	208.4	4.3M	2h2m	10.1	52.4	2.4M	1h4m	19.6	133.7	9.3M	1h53m	5.3
Qwen-3.6-27B [36]	34.4	408K	7m	61.4	190.6	1.8M	1h31m	8.9	86.9	1.3M	47m	16.5	128.8	3.7M	1h11m	5.3
Qwen-3.8-27B [37]	28.1	330K	9m	81.2	179.7	3.7M	1h28m	36.5	52.6	3.4M	57m	14.1	187.1	9.4M	1h48m	19.1
Qwen-3.8-Max [37]	22.8	284K	13m	82.8	209.4	4.7M	1h54m	44.7	44.3	4.1M	1h3m	29.2	114.9	6.7M	1h21m	25.0
Minimax-M3-428B-A23B [30]	37.4	385K	18m	68.3	309.0	12.3M	2h21m	23.0	-	-	-	-	-	-	-	-
Kimi-K2.6-1T-A32B [31]	19.0	188K	12m	72.7	-	-	-	-	-	-	-	-	114.8	1.8M	1h5m	10.5
Kimi-K3-2.8T-A104B [47]	19.5	309K	16m	84.2	90.4	4.0M	1h29m	55.9	39.8	4.1M	46m	43.2	127.4	9.8M	1h19m	25.0
Seed-2.1-Pro⋆ [40]	40.5	367K	17m	69.8	-	-	-	-	-	-	-	-	158.4	10.2M	1h37m	17.1
GPT-5.6-Luna⋆ [32]	33.1	275K	9m	77.3	210.7	6.0M	2h9m	46.3	63.4	3.7M	47m	44.0	192.0	9.1M	1h28m	22.4
GPT-5.6-Sol⋆ [32]	24.6	201K	10m	83.6	80.5	4.2M	1h13m	59.0	42.5	2.5M	38m	50.6	98.7	6.9M	57m	26.3
Claude-Sonnet-5⋆ [4]	31.5	420K	11m	79.0	-	-	-	-	78.3	4.4M	1h3m	40.8	-	-	-	-
Claude-Opus-5⋆ [3]	21.1	277K	9m	86.3	102.4	6.0M	1h46m	64.2	60.6	4.1M	56m	55.7	122.7	8.9M	1h22m	28.3
Specialist																
OpenCUA-72B [51]	24.6	185K	11m	59.3	-	-	-	-	-	-	-	-	-	-	-	-
CUA-Gym-397B-A17B [49]	26.9	325K	10m	72.0	191.2	2.5M	1h56m	14.5	64.7	1.6M	1h5m	14.0	-	-	-	-
EvoCUA-1.5-32B [17]	17.1	110K	9m	62.9	123.6	0.8M	56m	6.9	-	-	-	-	-	-	-	-
ScaleCUA-9B [28]	15.2	214K	8m	69.8	-	-	-	-	-	-	-	-	-	-	-	-
Qwen-CUA-397B-A17B [26]	28.9	377K	11m	84.0	183.0	4.5M	2h18m	43.6	66.2	4.0M	1h11m	35.1	148.7	7.3M	1h22m	21.1
UI-Mate-27B [9]	23.7	263K	8m	77.6	121.8	4.4M	1h28m	24.0	31.0	1.5M	30m	18.3	86.6	5.9M	50m	13.2
CUA-Master-27B (Ours)	**19.0**	**247K**	**8m**	**80.2**	**86.9**	**3.4M**	**1h7m**	**36.9**	**38.0**	**1.7M**	**34m**	**43.9**	**103.2**	**6.4M**	**52m**	**25.0**

benchmarks. We report four metrics averaged across tasks: the number of execution steps (Step), total token consumption (Token), wall-clock execution time (Time), and task performance score (Score). Here, we follow the setting of task runtime computation in [2], reflecting both the model time and the real execution time under the specified deployment configurations, but still influenced by deployment scheme and API speed. Higher scores indicate better task performance, while lower steps, token consumption, and execution time reflect greater execution efficiency.

3.1.3 Implementations

For training, all experiments are conducted on 16 nodes, each with 8 NVIDIA-H200-141GB GPUs. Initialized from Qwen-3.5-27B [35], we first perform SFT (Section 2.5.2), adopting AdamW optimization with a peak learning rate of 5×10^{-6}, a cosine schedule, a warmup ratio of 0.03, and a batch size of 128. For RL training, we follow the efficient online RL framework of ScaleCUA [28], a modified GRPO-style algorithm for CUAs, with frontier sampling and visual context segmentation. For each task, we sample 8 trajectories in real VMs (Section 2.1.3) to estimate advantages from verifier rewards. We use AdamW with a learning rate of 1×10^{-6} and a batch size of 64, corresponding to up to 512 sampled trajectories per batch.

For each SFT iteration, which trained on a cumulatively expanded dataset from the same initial checkpoint, we include 63K, 38K, 14K tasks and corresponding trajectories respectively, thus the total training data at the last SFT iteration is about 115K; while we use about 12K high-quality tasks for RL stage. More details of training data is reported in Section B.

For inference, to allow sufficient exploration in CUA tasks, we keep the temperature of most models to 1, except the models with other recommended settings. We deploy all open-weight models with 1-node (for models under 100B parameters) or 2-node (for models between 100B and 1T parameters) schemes; each node is equipped with 8 NVIDIA H200-141GB GPUs, and the inference is optimized based on vLLM [22]. And we use 6-node optimized deployment scheme for K3. For closed-source models, we evaluate them by directly calling their official APIs.

3.2 Main Results

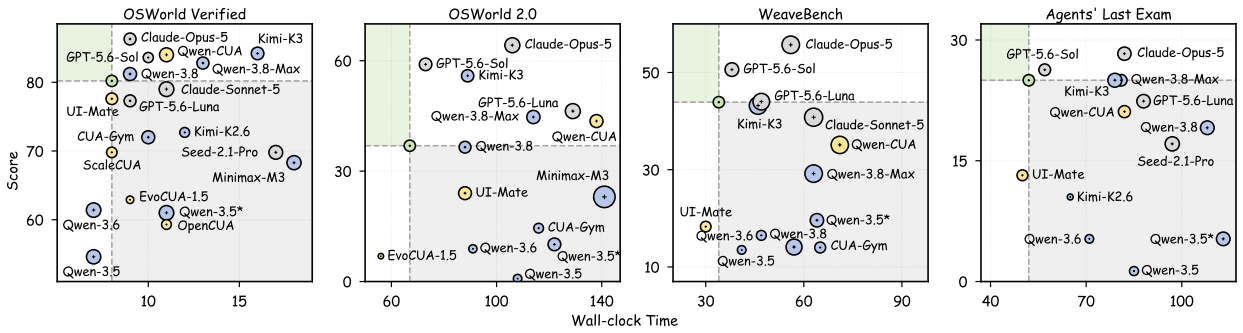


Figure 3 Efficiency comparison. The x-axis, y-axis, and the size denote wall-clock runtime, score, and token consumption respectively. For distinction, Qwen-3.5* represents the 397B-A17B model, the unmarked variant is 27B.

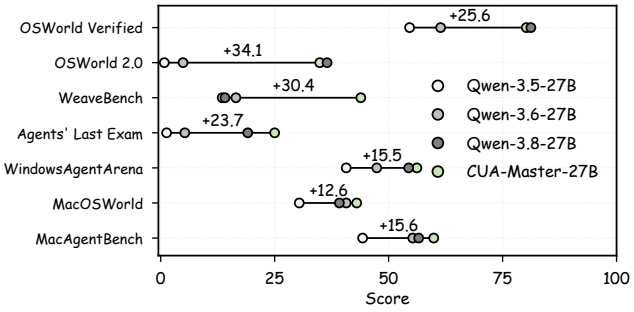


Figure 4 Comparisons with Qwen-3.5-27B derived models.

Table 2 Cross-platform generalization evaluation.

Method	Windows AgentArena†	Mac OSWorld†	Mac AgentBench‡
Qwen-3.5-27B [35]	40.7	30.4	44.3
Qwen-3.6-27B [36]	47.4	40.7	55.3
Qwen-3.8-27B [37]	54.4	39.2	56.6
EvoCUA-1.5-32B [17]	56.0	37.8	36.6
Qwen-CUA-397B-A17B [26]	50.4	42.0	65.3
UI-Mate-27B [9]	63.2	37.1	45.5
CUA-Master-27B (Ours)	**56.2**	**43.0**	**59.9**

3.2.1 CUA Ability Evaluation

As shown in Table 1, CUA-Master-27B achieves scores of 80.2 on OSWorld Verified, 36.9 on OSWorld 2.0, 43.9 on WeaveBench, and 25.0 on Agents' Last Exam, respectively. Compared with UI-Mate-27B, a powerful similarly sized specialist, it consistently improves performance by 2.6, 12.9, 25.6, and 11.8 across the four benchmarks. As reported in Figure 4, against all the models derived from Qwen-3.5-27B, our model achieves comparable performance of Qwen-3.8-27B on GUI-only tasks, while delivering substantial gains on hybrid CUA scenarios. Notably, on the most challenging benchmark, it only lags behind GPT-5.6-Sol and Claude-Opus-5, and surpasses all other counterparts (when full-pass rate is equal, compare the average score). Despite its substantially smaller parameters, CUA-Master-27B achieves competitive performance against Qwen-CUA-397B-A17B on GUI-only benchmarks, while even outperforming it on hybrid benchmarks, respectively.

3.2.2 Performance-efficiency Comparison

As illustrated in Figure 3, CUA-Master-27B demonstrates a favorable performance–efficiency trade-off, illustrating the faster model decision-making time and also superior interactive execution efficiency. On OSWorld Verified, it achieves a score of 80.2 in only 8 minutes, and on OSWorld 2.0, it slightly surpasses Qwen-3.8-27B (36.9 vs. 36.5) while reducing runtime by 24%. The efficiency advantage becomes particularly pronounced on challenging hybrid benchmarks. On WeaveBench, CUA-Master achieves a competitive score in 34 minutes, matching GPT-5.6-Luna and with 28% less runtime. Similarly, on Agents' Last Exam, it matches the performance of Kimi-K3 and Qwen-3.8-Max while requiring 34–36% less time.

3.2.3 Generalization

As shown in Table 2, CUA-Master-27B demonstrates strong cross-platform generalization across Windows and macOS environments, achieving scores of 56.2, 43.0, and 59.9 on WindowsAgentArena, MacOSWorld, and MacAgentBench, respectively. These results demonstrate that CUA-Master generalizes effectively across heterogeneous operating systems and interaction modalities.



Figure 5 Demonstrations of three selected trajectories of GUI+CLI hybrid CUA executions.

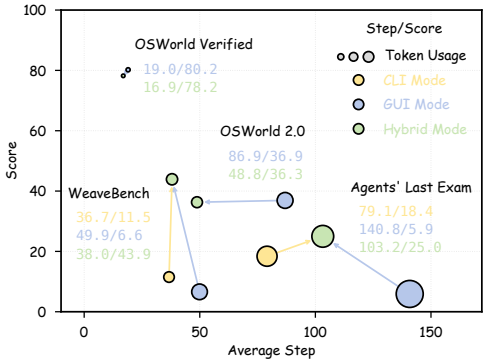


Figure 6 Step, token, and score comparisons of GUI-only, CLI-only, and Hybrid modes.

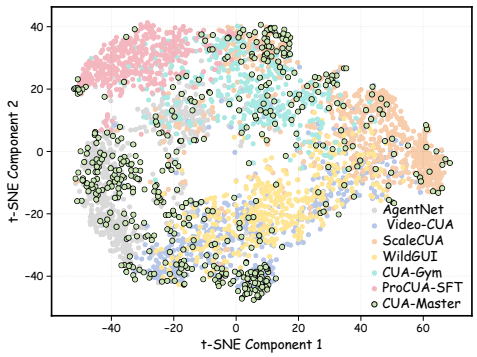


Figure 7 t-SNE visualization of the embeddings of randomly sampled tasks.

Table 3 Comparisons of rollout success rates on various level of difficulty.

K / N	Baseline	Ours
5 / 1	74.0%	95.4%
10 / 1	53.0%	89.1%
20 / 1	31.5%	83.7%
10 / 3	29.2%	62.3%
20 / 3	19.3%	56.9%
40 / 3	14.6%	49.8%
20 / 5	1.9%	32.8%
40 / 5	0.4%	24.4%
50 / 8	0.0%	18.7%

3.3 Additional Analyses

3.3.1 Effectiveness of Hybrid Action Space

To investigate the effectiveness of the hybrid action space, we additionally compare CUA-Master under GUI-only, CLI-only, and hybrid execution modes. As shown in Figure 6, hybrid execution substantially improves both task performance and execution efficiency, particularly on complex benchmarks. On WeaveBench, the hybrid mode achieves a score of 43.9, outperforming GUI-only (6.6) and CLI-only (11.5) by 37.3 and 32.4 points, respectively, while reducing execution steps by 23.8% compared to GUI-only. Especially on the more long-horizon, complex Agents' Last Exam benchmark, hybrid execution improves the score from 5.9 to 25.0 over GUI-only, with 26.7% fewer steps; on OSWorld 2.0, it improves performance by 1.4 points while reducing execution steps by 43.8%.

As demonstrated in Figure 5, the visualization of trajectories further illustrates the complementary strengths of GUI and CLI actions: CLI enables file processing, batch computation, structured data manipulation, and other efficient operations, while GUI supports visual inspection and interactive application operations. By adaptively switching between these modalities according to task requirements, CUA-Master effectively exploits their complementary capabilities, enabling more efficient, and effective computer-use workflows.

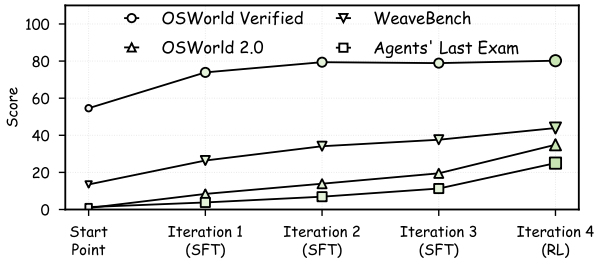


Figure 8 Performance trends across training iterations.

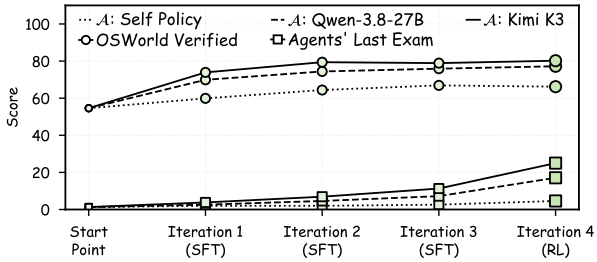


Figure 9 Influences of different annotator models.

3.3.2 Analysis of Data Flywheel

We analyze the effectiveness of our data flywheel from two perspectives: task diversity and data construction efficiency. As illustrated in Figure 7, the t-SNE visualization suggests that tasks generated by our data flywheel exhibit broad coverage in the embedding space, spanning multiple distributions of existing CUA datasets: AgentNet [51], VideoCUA [20], ScaleCUA [28], WildGUI [57], CUA-Gym [49], and ProCUA-SFT [21], while extending into sparsely covered regions. Furthermore, as reported in Table 3, the rollout success rates consistently surpass those of the baseline, which utilizes naive environments to rollout with the same retry budget but re-rollout from the start, across all difficulty levels. Notably, when synthesizing tasks with $K = 40$ primitives across $N = 3$ applications, which is comparable to the average difficulty of Agents' Last Exam, our method achieves a nearly 50% success rate compared to 14.6% for the baseline. The advantage remains substantial under the most challenging configurations, achieving 18.7% versus 0.0% at $K/N = 50/8$, which involves more primitives and applications than the extreme cases in Agents' Last Exam. These results demonstrate that our data flywheel achieves both broad task coverage and a substantially higher yield of successful rollouts, enabling reliable and efficient construction of diverse, challenging data.

3.3.3 Analysis of World-data-model Flywheel

We analyze the effectiveness of our world–data–model flywheel from two perspectives. As shown in Figure 8, successive training iterations progressively improve performance across all four benchmarks, with OSWorld Verified and WeaveBench increasing by over 25 points. Notably, when SFT improvements on relatively simple tasks approach saturation, the final RL stage further boosts performance on OSWorld 2.0 and Agents' Last Exam, which are more challenging and demand stronger exploration, by approximately 15 points, respectively. Moreover, as illustrated in Figure 9, stronger annotation models consistently yield greater performance gains throughout the iterative process. The self-policy strategy, *i.e.*, using the model from each training iteration as the annotator to generate data for the next iteration, can bring about some improvements, but its gains are relatively limited. These results highlight the effectiveness of our flywheel in establishing a positive feedback loop that progressively enhances computer-use capabilities, particularly on challenging long-horizon tasks.

4 Conclusion

In this work, we introduce CUA-Master, a nested double-loop recursive framework that advances CUA systems beyond unidirectional data pipelines and closed-world self-improvement. At the system level, the world-data-model flywheel continuously incorporates open-world knowledge and leverages evolving model weaknesses to guide further iterations. At the data level, we establish a complementary flywheel that couples open-world seeding with autocatalytic propagation, which integrates visual-semantic trajectory reconstruction, primitive-based task composition, and feedback-guided snapshot-rollback retries to enable diverse, controllable, and reliable data synthesis. Extensive experiments demonstrate that CUA-Master-27B achieves competitive performance across GUI-only and GUI-CLI hybrid benchmarks, with strong execution efficiency and consistent improvements across recursive iterations. These results highlight the potential of our method for developing increasingly general and capable CUA.

References

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

[2] Pranjal Aggarwal, Lawrence Keunho Jang, Sean Welleck, Daniel Fried, Ruslan Salakhutdinov, and Jing Yu Koh. cua-speedrun: Standardized benchmarking of the speed of computer-use agents. arXiv preprint arXiv:2609.40284, 2026.

[3] Anthropic. Introducing claude opus 5, July 2026. URL https://www.anthropic.com/news/claude-opus-5.

[4] Anthropic. Introducing claude sonnet 5, June 2026. URL https://www.anthropic.com/news/claude-sonnet-5.

[5] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating multi-modal os agents at scale. arXiv preprint arXiv:2409.08264, 2024.

[6] Tianpeng Bu, Xin Liu, Qihua Chen, Hao Jiang, Shurui Li, hongtao duan, Lu Jiang, lulu hu, Bin Yang, and Minying Zhang. Recovering policy-induced errors: Benchmarking and trajectory synthesis for robust GUI agents. In International Conference on Machine Learning (ICML), 2026.

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

[8] Tongbo Chen, Junbo Niu, Zhengxi Lu, Niu Lian, Fei Tang, Yuchen Yan, Yike Hong, Yong Du, Yizhou Liu, Bofan Chen, et al. Hybridcua: Learning to orchestrate gui and cli for computer-use agents. arXiv preprint arXiv:2609.38008, 2026.

[9] Zihan Ding, Longxu Dou, Qi Gao, Xiangwu Guo, Shengchao Hu, Zilong Huang, Zihang Jiang, Lei Ke, Mengcheng Lan, Weixian Lei, et al. Ui-mate: Advancing open-weight foundation gui agents with in-context demonstrations. arXiv preprint arXiv:2608.15930, 2026.

[10] Shuai Dong, Yongfu Zhu, Yuqi Xu, Weichu Xie, Ziyue Wang, Kaiwen Tuo, Congcong Wang, Siyuan Wang, Wenqi Shao, Shuai Yang, et al. Rl starts before rl: On policy distillation for better reinforcement learning. arXiv preprint arXiv:2609.28145, 2026.

[11] Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Re-rest: Reflection-reinforced self-training for language agents. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 15394–15411, 2024.

[12] Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. Webevolver: Enhancing web agent self-improvement with co-evolving world model. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 8970–8986, 2025.

[13] Aarash Feizi, Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Kaixin Li, Rabiul Awal, Xing Han Lù, Johan S Obando Ceron, Juan A Rodriguez, Nicolas Chapados, et al. Grounding computer use agents on human demonstrations. In International Conference on Learning Representations (ICLR), volume 2026, pages 5181–5215, 2026.

[14] Yikun Fu, Bowen Fu, Zhenyu Wu, Shuang Cheng, Xiaowei Sun, Bowen Yang, Zehao Li, Yibo Zhao, Zichen Ding, Zhoumianze Liu, et al. Macagentbench: Benchmarking ai agents on real-world macos desktop. arXiv preprint arXiv:2606.22557, 2026.

[15] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 14281–14290. IEEE, 2024.

[16] Xuhao Hu, Xi Zhang, Haiyang Xu, Kyle Qiao, Jingyi Yang, Xuanjing Huang, Jing Shao, Ming Yan, and Jieping Ye. Toolcua: Towards optimal gui-tool path orchestration for computer use agents. arXiv preprint arXiv:2605.12481, 2026.

[17] Mianqiu Huang, Taofeng Xue, Chong Peng, Jinrui Ding, Sicheng Fan, Jiale Hong, Yufei Gao, Xiaocheng Zhang, Linsen Guo, Xin Yang, et al. Evocua-1.5: Online reinforcement learning for multi-turn computer-use agents. arXiv preprint arXiv:2607.09773, 2026.

[18] Yunseok Jang, Yeda Song, Sungryull Sohn, Lajanugen Logeswaran, Tiange Luo, Dong-Ki Kim, Kyunghoon Bae, and Honglak Lee. Scalable video-to-dataset generation for cross-platform mobile agents. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Conference (CVPR), pages 8604–8614, 2025.

[19] Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents. In International Conference on Learning Representations (ICLR), volume 2026, pages 63734–63768, 2026.

[20] Xiangru Jian, Shravan Nayak, Kevin Qinghong Lin, Aarash Feizi, Kaixin Li, Patrice Bechard, Spandana Gella, and Sai Rajeswar. Cua-suite: Massive human-annotated video demonstrations for computer-use agents. arXiv preprint arXiv:2603.24440, 2026.

[21] Jaehun Jung, Ximing Lu, Brandon Cui, Muhammad Khalifa, Shaokun Zhang, Hao Zhang, Jin Xu, Amala Sanjay Deshmukh, Karan Sapra, Andrew Tao, et al. Procua-sft technical report. arXiv preprint arXiv:2606.17321, 2026.

[22] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of Symposium on Operating Systems Principles (SOSP), pages 611–626, 2023.

[23] Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents. In International Conference on Learning Representations (ICLR), volume 2026, pages 23553–23591, 2026.

[24] Wanli Li, Bowen Zhou, Yunyao Yu, Zhou Xu, Yifan Yang, Dongsheng Li, and Caihua Shan. Weavebench: A long-horizon, real-world benchmark for computer-use agents with hybrid interfaces. arXiv preprint arXiv:2606.09426, 2026.

[25] Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, et al. Scalecua: Scaling open-source computer use agents with cross-platform data. In International Conference on Learning Representations (ICLR), volume 2026, pages 62707–62766, 2026.

[26] Dunjie Lu, Shuai Bai, Tianyi Bai, Sicheng Fan, Chang Gao, Jian Guan, Feng Hu, Mianqiu Huang, Xingyang Huang, Yizhen Jiang, et al. Qwen-cua: Native computer use for (almost) everything. arXiv preprint arXiv:2608.02352, 2026.

[27] Dunjie Lu, Yiheng Xu, Junli Wang, Haoyuan Wu, Xinyuan Wang, Zekun Wang, Junlin Yang, Hongjin Su, Jixuan Chen, Junda Chen, et al. Videoagenttrek: Computer-use pretraining from unlabeled videos. In International Conference on Learning Representations (ICLR), volume 2026, pages 122696–122720, 2026.

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

[29] Nico Messikommer, Jiaxu Xing, Elie Aljalbout, and Davide Scaramuzza. Student-informed teacher training. In International Conference on Learning Representations (ICLR), volume 2025, pages 67308–67323, 2025.

[30] MiniMax. Minimax m3: Frontier coding, 1m context, native multimodality — all in one model, June 2026. URL https://www.minimax.io/blog/minimax-m3.

[31] Moonshot AI. Kimi k2.6: From code to creation, from one to many, 2026. URL https://www.kimi.com/ai-models/kimi-k2-6. Official Moonshot AI model release page (Kimi K2.6).

[32] OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition, July 2026. URL https://openai.com/zh-Hans-CN/index/gpt-5-6/.

[33] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In International Conference on Learning Representations (ICLR), volume 2025, pages 79791–79821, 2025.

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

[35] Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[36] Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27b dense model, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b.

[37] Qwen Team. Qwen3.8-max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

[38] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

[39] Ram Ramrakhya, Andrew Szot, Omar Attia, Bogdan Mazoure, Anh Nguyen, Yuhao Yang, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. In International Conference on Learning Representations (ICLR), volume 2026, pages 123630–123660, 2026.

[40] ByteDance Seed. Seed2.1 officially released: Advancing ai productivity, June 2026. URL https://seed.bytedance.com/en/blog/seed2-1-officially-released-advancing-ai-productivity.

[41] Rui Shao, Ruize Gao, Bin Xie, Yixing Li, Kaiwen Zhou, Shuai Wang, Weili Guan, and Gongwei Chen. Hats: Hardness-aware trajectory synthesis for gui agents. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 27471–27481, June 2026.

[42] Haoting Shi, Wenhao Wang, Weicheng Fang, Yaozhong Liang, Tian Jin, Pengxiang Zhao, Guangyi Liu, Siheng Chen, and Yanfeng Wang. Cua-universe: A scalable and dynamic environment for hybrid gui+ cli agents. arXiv preprint arXiv:2609.05374, 2026.

[43] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. Ai models collapse when trained on recursively generated data. Nature, 631(8022):755–759, 2024.

[44] Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), pages 5555–5579, 2025.

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

[46] Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. SEAgent: Self-evolving computer use agent with autonomous learning from experience. In International Conference on Machine Learning (ICML), 2026.

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

[48] Venus Team, Zhuohan Cai, Haoxing Chen, Jiaxuan Chen, Weizhi Chen, Changlong Gao, Zhangxuan Gu, Yuan Guo, Yusong Hu, Jianrong Jiang, et al. Ui-venus-2 technical report. arXiv preprint arXiv:2609.00028, 2026.

[49] Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, et al. Cua-gym: Scaling verifiable training environments and tasks for computer-use agents. arXiv preprint arXiv:2605.25624, 2026.

[50] Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. arXiv preprint arXiv:2509.02544, 2025.

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

[52] Jinbiao Wei, Qianran Ma, Yilun Zhao, Xiao Zhou, Kangqi Ni, Guo Gan, and Arman Cohan. Opencomputer: Verifiable software worlds for computer-use agents. arXiv preprint arXiv:2605.19769, 2026.

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

[54] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: Foundation action model for generalist gui agents. In International Conference on Learning Representations (ICLR), volume 2025, pages 5090–5108, 2025.

[55] Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents. In International Conference on Learning Representations (ICLR), volume 2026, pages 155149–155177, 2026.

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

[57] Weimin Xiong, Shuhao Gu, Bowen Ye, Zihao Yue, Lei Li, Feifan Song, Sujian Li, and Hao Tian. Video2gui: Synthesizing large-scale interaction trajectories for generalized gui agent pretraining. arXiv preprint arXiv:2605.14747, 2026.

[58] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. arXiv preprint arXiv:2412.04454, 2024.

[59] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In International Conference on Learning Representations (ICLR), volume 2025, pages 79822–79843, 2025.

[60] Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. Evocua: Evolving computer use agents via learning from scalable synthetic experience. arXiv preprint arXiv:2601.15876, 2026.

[61] Cheng Yang, Jiayang Lyu, Shangyuan Liu, Guibin Zhang, Jiong Lin, Xinlei Yu, Junchi Yan, Shuicheng Yan, Linfeng Zhang, Qibing Ren, et al. icoder-27b: Recursive ai-led development of frontier industrial coding model. arXiv preprint arXiv:2609.29626, 2026.

[62] Cheng Yang, Yifan Wu, Yutao Huang, Zhaohua Zhang, Beiduo Chen, Muxi Chen, Chenchen Zhao, Hexuan Deng, Haolin Yang, Geyuan Zhu, et al. Autoguiworld: Image generators as visual world models for gui agent. arXiv preprint arXiv:2610.01215, 2026.

[63] Pei Yang, Hai Ci, and Mike Zheng Shou. macosworld: A multilingual interactive benchmark for gui agents. Advances in Neural Information Processing Systems (NeurIPS), 38:134014–134056, 2026.

[64] Mingqian Yu, Wenpeng Zhang, and Peilin Zhao. T-loopformer: Token-level elastic-depth looped transformers for latent reasoning with dynamic routing. arXiv preprint arXiv:2609.15160, 2026.

[65] Xinlei Yu, Zhangquan Chen, Yongbo He, Tianyu Fu, Guanting Dong, Cheng Yang, Chengming Xu, Yue Ma, Xiaobin Hu, Zhe Cao, et al. The latent space: Foundation, evolution, mechanism, ability, and outlook. arXiv preprint arXiv:2604.02029, 2026.

[66] Xinlei Yu, Gen Li, Qingyi Si, Guibin Zhang, Yuqi Xu, Congcong Wang, Shuai Dong, Kaiwen Tuo, Xiangyu Zeng, Kaituo Feng, et al. Dopd: Dual on-policy distillation. arXiv preprint arXiv:2606.30626, 2026.

[67] Xinlei Yu, Chengming Xu, Guibin Zhang, Zhangquan Chen, Yudong Zhang, Yongbo He, Peng-Tao Jiang, Jiangning Zhang, Xiaobin Hu, and Shuicheng Yan. Vismem: Latent vision memory unlocks potential of vision-language models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 31544–31555, 2026.

[68] Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, et al. Osworld2. 0: Benchmarking computer use agents on long-horizon real-world tasks. arXiv preprint arXiv:2606.29537, 2026.

[69] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. arXiv preprint arXiv:2501.11425, 2025.

[70] Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. Tongui: Internet-scale trajectories from multimodal web tutorials for generalized gui agents. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 40, pages 12367–12375, 2026.

[71] Weixiang Zhao, Yingshuo Wang, Yichen Zhang, Yang Deng, Yanyan Zhao, Wanxiang Che, Bing Qin, and Ting Liu. Large language model agents are not always faithful self-evolvers. arXiv preprint arXiv:2601.22436, 2026.

[72] Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, et al. Mai-ui technical report: Real-world centric foundation gui agents. arXiv preprint arXiv:2512.22047, 2025.

[73] Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. Advances in Neural Information Processing Systems (NeurIPS), 38:113959–113991, 2026.

Appendix

A Related Works

A.1 CUA Specialists

Research on CUAs has progressed from visual grounding to autonomous interaction and long-horizon task execution. Early works, including OS-Atlas [54], Tong-UI [70], and CogAgent [15], establish foundational GUI perception and grounding capabilities. Building upon these advances, Aguvis [58], UI-TARS-2 [34], Agent S [1], and MAI-UI [72] integrate visual understanding, reasoning, and action planning for autonomous GUI interaction. To further improve performance in complex environments, OpenCUA [51], CUA-Gym [49], EvoCUA-1.5 [17], ScaleCUA [28], UI-Mate [9], and UI-Venus-2 [48] advance reliable data synthesis and specialized training paradigms. More recently, research has expanded beyond GUI-only interaction toward hybrid action spaces. UI-TARS-2 [50] incorporates terminal and file-system operations, while ToolCUA [16] and HybridCUA [8] coordinate GUI actions with structured tools and command-line interfaces. Collectively, these advances mark a shift from fine-grained visual grounding toward scalable, practical, and reliable CUAs.

A.2 Scalable Data Synthesis

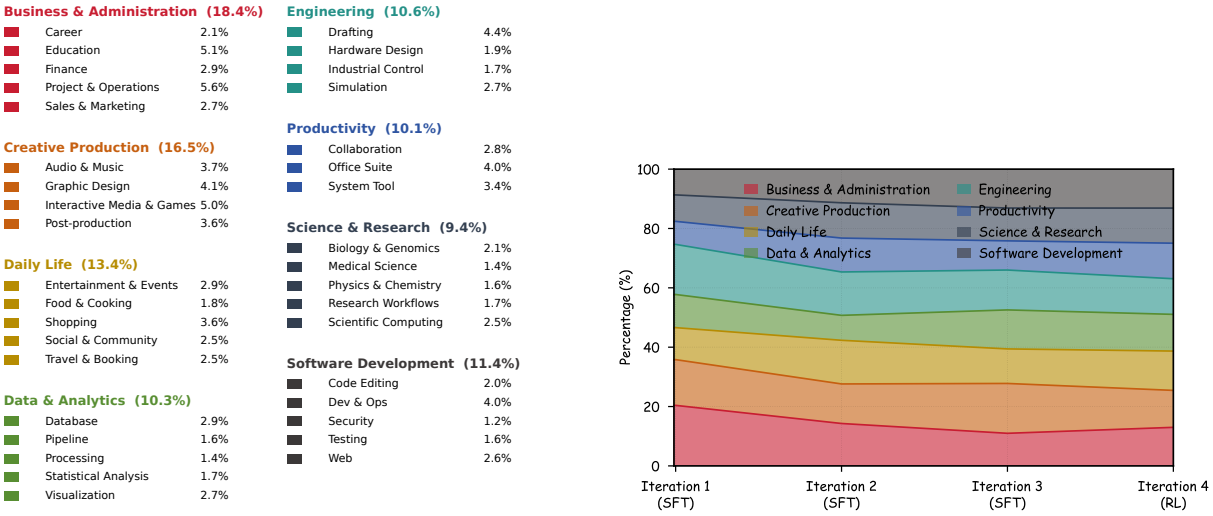
Scalable data synthesis has emerged as a key paradigm for overcoming the scarcity of high-quality computer-use trajectories. One line of research leverages naturally occurring interaction data and instructional resources. OpenCUA [51], WildGUI [57], and VideoCUA [20] exploit real-world interaction traces or videos, while AgentTrek [59] and TongUI [70] transform web tutorials into agent supervision. Another line synthesizes tasks and trajectories through environment interaction. OS-Genesis [44] adopts reverse task synthesis, whereas AutoPlay [39] discovers executable tasks through environment exploration. CUA-Gym [49], EvoCUA [60], and ScaleCUA [28] further scale supervision through verifiable task synthesis, large-scale environment interaction, and iterative policy optimization. Despite these advances, jointly achieving real-world diversity, controllable task complexity, reliable verification, and efficient execution remains challenging. Our work unifies open-world workflow acquisition with primitive-based compositional synthesis, enabling systematic expansion from authentic operational knowledge to diverse, verifiable, and increasingly complex computer-use tasks.

A.3 Recursive Improvements

Recursive improvement aims to expand agent capabilities through iterative data generation, experience refinement, and policy optimization. Early approaches such as Re-ReST [11] and Agent-R [69] exploit environmental feedback and iterative refinement to improve agent trajectories. Moving toward adaptive task generation, WebRL [33] introduces self-evolving curricula driven by failed executions, and Self-Challenging Agents [73] jointly develop task generation and policy learning through self-generated, verifiable challenges. WebEvolver [12] further couples agent improvement with an evolving world model. Nevertheless, recursive training may suffer from distributional narrowing, accumulated synthetic errors, and diminishing task novelty [11, 43, 71]. In contrast, our world-data-model flywheel anchors recursive improvement in continuously refreshed real-world knowledge. Open-world seeding introduces external diversity, compositional propagation expands task coverage and difficulty, and model feedback directs subsequent exploration and optimization, forming a grounded and adaptive cycle of capability expansion.

B Details of Training Data

Our training data are constructed through open-world seeding and autocatalytic propagation, yielding a dataset of 127K examples (115K for SFT training and 12K for RL training). To ensure diverse application coverage, we organize data construction around a hierarchical taxonomy encompassing 8 domains, 36 subdomains, and over 650 applications. As illustrated in Figure 10a, the taxonomy spans Business & Administration (18.4%), Creative Production (16.5%), Daily Life (13.4%), Developer Tools (11.4%), Engineering (10.6%), Data & Analytics (10.3%), Productivity (10.1%), and Science & Research (9.4%). This broad coverage supports the acquisition of heterogeneous operational skills across both everyday and specialized workflows.



(a) Data distributions of the full training dataset. **(b)** Data distributions of each iteration.

Figure 10 The distributions of training dataset.

Table 4 Domain results of OSWorld Verified [56].

Domain	Count	Score
Office	117	83.1
Professional	49	88.0
Workflow	93	70.7
Daily	78	78.8
OS	24	91.6
Overall	361	80.2

Table 5 Domain results of OSWorld 2.0 [68].

Domain	Count	Score
Research & Education	22	44.7
Creative Production	22	29.4
Engineering & Computing	19	29.6
Personal Services	14	42.8
Administration & Compliance	14	38.3
Business & Finance	13	41.2
Healthcare	4	30.4
Overall	108	36.9

Table 6 Illustrative domain-level average scores on WeaveBench [24].

Domain	Count	Score
Web Frontend	15	58.4
DevOps	12	49.7
Design	10	26.3
Spatial/CAD	12	25.9
Data Analysis	13	38.8
Document	17	44.1
Gaming	17	41.7
Desktop OS	18	55.3
Overall	114	43.9

As shown in Figure 10b, the training data distribution evolves across successive iterations rather than following a fixed domain mixture. Compared with the initial SFT stage, subsequent iterations increase the relative representation of Software Development, Science & Research, and Productivity, while reducing the initial concentration in Business & Administration. The final RL stage adopts a more balanced distribution across domains. This adaptive reallocation aligns data construction with evolving model capability gaps while maintaining broad application coverage.

C Details of Methodology

As listed in Table 7, we use a hybrid action space for our proposed CUA-Master, including both GUI actions and CLI actions, as well as extensible registered tools.

As summarized in Table 8, our framework employs a general-purpose annotator $\mathcal{A}$ with eight specialized roles throughout the data construction pipeline. Rather than relying on a single annotation procedure, we adopt role-specific prompts to guide $\mathcal{A}$ in performing distinct operations with corresponding inputs, objectives, and structured outputs. These prompts are shown in Fig 11, 12, 13, 14, 15, 16, 17, and 18.

Table 7 The hybrid atomic action space for CUA interactions.

Action Type	Definition
`computer.click`	Performs a single mouse click at a specified position.
`computer.double_click`	Performs a double click for activation or selection.
`computer.triple_click`	Performs a triple click for text or row selection.
`computer.move`	Moves the pointer to a specified screen position.
`computer.drag`	Drags the pointer between two screen positions.
`computer.scroll`	Performs horizontal or vertical scrolling.
`computer.type`	Inserts a sequence of textual characters.
`computer.press`	Presses a single key or a keyboard shortcut.
`computer.key_down/up`	Holds or releases an individual keyboard key.
`computer.mouse_down/up`	Holds or releases an individual mouse button.
`computer.wait`	Waits for asynchronous environment changes.
`shell.exec`	Executes a structured command in a controlled shell.
`tool.call`	Invokes a registered tool through a structured interface.
`finish`	Terminates the task with a success or failure status.

Table 8 Summary of the types, functions, allowed models, and corresponding prompts of annotator $\mathcal{A}$ used in our recursive flywheel.

Type	Function	Prompt
Open-world Seeding		
Score Evaluator	Video suitability scoring and filtering	Fig. 11
Task Generator	Seed task mining and temporal segmentation	Fig. 12
Action Rechecker	Visual-semantic action boundary refinement	Fig. 13
Action Annotator	Atomic GUI action annotation	Fig. 14
Trajectory Verifier	Trajectory-level verification and filtering	Fig. 15
Autocatalytic Propagation		
Primitive Extractor	Reusable ability primitive extraction	Fig. 16
Task Synthesizer	Composable task and verifier synthesis	Fig. 17
Detector	Failure localization and diagnostic feedback	Fig. 18

D Details of Experiments

As reported in Tables 4, 5, and 6, we further analyze the domain-level performance of CUA-Master-27B across three benchmarks. Overall, these results demonstrate broad domain coverage and general performance improvements across domains while revealing remaining capability gaps in specialized manipulation and complex application workflows.

Video Frame 1: {VIDEO_FRAME_1}
...
Video Frame N: {VIDEO_FRAME_N}

You are rating how suitable a video is as training data for a Computer-Use Agent (CUA). CUA data requires real computer-use screen recordings that show a step-by-step operation trajectory, such as clicks, typing, menus, and dialogs, that an agent could learn to reproduce.

You are given a few evenly spaced frames sampled across the video and, optionally, its transcript. Judge the whole video from this evidence.

Score the overall CUA suitability using an integer from 0 to 5:

- 5: A clear, high-resolution software CUA screencast with an explicit step-by-step operation trajectory; ideal CUA training data.
- 3–4: Mostly CUA operation, but with minor issues such as talking-head segments, slides, incomplete trajectories, or moderate visual clarity.
- 1–2: Only incidental CUA content, predominantly non-operational content, or poor visual clarity.
- 0: No software CUA trajectory, such as game footage, real-world video, music videos, movies, or pure talking-head content.

Additionally assess:
- is_cua: whether the video primarily presents a real software computer-use interaction.
- has_trajectory: whether it contains a reproducible step-by-step operation.
- clarity: an integer from 0 to 5 indicating UI legibility.

Transcript: <TRANSCRIPT>

Respond with only the following JSON object:
{"score": <0–5 integer>,"is_cua": <boolean>,"has_trajectory": <boolean>,"clarity": <0–5 integer>,"reason": "<one concise sentence>"}

Figure 11 Prompt for video suitability scoring and filtering.

Overview frame 1: <OVERVIEW_IMAGE_1>
...
Overview frame N: <OVERVIEW_IMAGE_N>

You are planning how to extract Computer-Use Agent (CUA) trajectories from a tutorial video. Identify only ranges that contain real software operation, including GUI interaction, terminal commands, and explicit structured tool calls. Exclude intro, outro, marketing, slide-only, talking-head, and explanation-only ranges. Choose one primary long-horizon episode: the longest single continuous workflow with a clear user goal, visible start state, many intermediate CUA actions, and visible completion or stable end state. Do not combine independent examples, repeated demonstrations, or unrelated tutorial sections into one episode. Prefer the longest coherent workflow over isolated tips. Also list every other coherent exportable episode in episode_segments so the pipeline can split one tutorial video into multiple trajectories. Each episode must be independently trainable and continuous.

Application: <APPLICATION>
Taxonomy: <CATEGORY} / <SUBCATEGORY>
Transcript: <TRANSCRIPT>

Return ONLY JSON with this shape:
{"task_title":"...","primary_episode": {"start":1.0,"end":60.0,"task":"complete workflow goal","summary":"why this is one coherent episode","confidence":0.0},"episode_segments":[{"start":1.0,"end":60.0,"task":"complete workflow goal","summary":"single coherent workflow","confidence":0.0}]," ignore_ranges": [{"start":0.0,"end":1.0,"reason":"intro"}], "high_value_ranges":[{"start":1.0,"end":10.0,"reason":"CUA operation"}], "confidence":0.0}

Figure 12 Prompt for seed task mining and temporal segmentation.

Boundary frame 1: <BOUNDARY_IMAGE_1>
...
Boundary frame N: <BOUNDARY_IMAGE_N>

You are refining action boundaries for Computer-Use Agent (CUA) tutorial video mining. Use transcript, sampled frames, and candidates to produce accurate atomic CUA operation. Delete explanation-only candidates, add missing visible CUA actions, split multi-action candidates, and merge duplicated candidates. Add visible missing actions that are necessary for state transitions, including menu opens, dialog opens, focus clicks before typing, submit/save clicks, waits for navigation, and confirmation actions. Represent a complete submitted/executed terminal command as shell.exec with its exact command; do not emit both shell.exec and the lower-level terminal typing/Enter actions for the same transition. Keep unsubmitted terminal text as computer.type. When the same workflow has many visible steps, preserve them as atomic actions instead of collapsing the workflow into only the most important actions.

Reject candidates outside the primary episode unless screenshots/transcript show they are required intermediate actions in the same workflow. Do not bridge unexplained jumps with guessed actions.

Allowed action types: <ACTION_SPACE>
Application: <APPLICATION>
Video plan: <VIDEO_PLAN_JSON>
Candidates: <CANDIDATE_JSON>
Transcript: <TRANSCRIPT>

Return ONLY JSON with this shape:
{"actions":[{"id":"a1","start":1.0,"end":2.0,"action_type":"shell.exec","args":{"command":"pwd"},"target":"terminal","intent":"print working directory","confidence":0.9,"source_ids":["c1"],"evidence":"..."}],"false_positive_candidate_ids":["c2"],"merge_groups":[["c3","c4"]],"missing_actions":[{"start":3.0,"end":4.0,"action_type":"computer.scroll","args":{"dx":0,"dy":-5},"target":"list","intent":"scroll down","confidence":0.8,"evidence":"..."}]}

Figure 13 Prompt for visual-semantic action boundary refinement.

State frame 1: <STATE_IMAGE_1>
...
State frame N: <STATE_IMAGE_N>

You are labeling a short contiguous chain of atomic Computer-Use Agent (CUA) actions from a tutorial video for CUA training.
Use the state frames and step drafts to label each action. Step i is the transition from state frame i-1 to state frame i in the provided sequence.
Use the neighboring transitions as context, but return one independent action object for every requested step index.
Preserve the expected action type unless the screenshots make it clearly impossible. If changed, include changed_action_type_reason with concrete visual evidence.
Coordinates must be normalized to 0-1000 screen space, x from left to right and y from top to bottom.
Use shell.exec only for a complete submitted/executed terminal command, not for unsubmitted terminal text.
If a requested transition is not a real CUA action, set is_action=false and action=null for that step.

Allowed action types: <ACTION_SPACE>
Task: <TASK>
State frames provided: <IMAGE_COUNT>
Requested steps: <REQUESTED_STEPS_JSON>

Return ONLY a JSON object with this shape:
{"steps":[{"index":1,"is_action":true,"action":{"type":"computer.click","args":{"x":500,"y":500,"button":"left"}},"intent":"...","confidence":0.0,"target":"...","reason":"...","changed_action_type_reason":""}]}

Figure 14 Prompt for atomic computer-use action annotation.

Action observation 1: <ACTION_IMAGE_1>
...
Action observation N: <ACTION_IMAGE_N>

You are verifying a Computer-Use Agent (CUA) trajectory mined from a tutorial video. Judge whether the selected steps form a coherent, trainable CUA trajectory or self-contained CUA subtask, and use screenshots when provided. Set trajectory_valid=true when the steps are locally continuous, screenshots align with actions, and the final state is stable, even if the clip is only a useful subtask of a broader video task. For exportable_for_training=true, require a single continuous workflow or complete subtask with no large unexplained state jumps, enough intermediate actions to reproduce that subtask, aligned pre/post screenshots, and a visible completed or stable end state. Set exportable_for_training=false when the task title claims a goal that the selected steps do not perform or visibly complete, even if the individual actions are locally plausible. Set exportable_for_training=false when steps stitch together unrelated apps, sites, tabs, examples, competitor research, search/ChatGPT side workflows, or page contexts without visible navigation and return actions. Do not reward length by itself. Do not reject a short trajectory solely because it does not complete the entire tutorial-level task; reject it only if it is a mere tip, lacks a stable subtask goal, omits essential intermediate actions for the visible subtask, has task/step mismatch, or stitches unrelated examples. When the trajectory is a valid fragment but not exportable as a complete workflow/subtask, set trajectory_valid=true and exportable_for_training=false, and explain what is missing.

Application: <APPLICATION>
Transcript: <TRANSCRIPT>
Task: <TASK>
Images provided: <IMAGE_COUNT>
Steps: <TRAJECTORY_STEPS_JSON>

Return ONLY JSON with this shape:
{"trajectory_valid":true,"exportable_for_training":true,"quality_score":4,"step_validity":[{"index":1,"valid":true,"reason":"..."}],"missing_step_ranges":[],"redundant_steps":[],"order_errors":[],"reason":"..."}

Figure 15 Prompt for trajectory-level verification and filtering.

Trajectory observation 1: <TRAJECTORY_IMAGE_1>
...
Trajectory observation N: <TRAJECTORY_IMAGE_N>

You are extracting reusable application-level ability primitives from a Computer-Use Agent (CUA) task and its trajectory. An ability primitive is the smallest independently meaningful, user-visible state transition. It may contain multiple clicks, typing actions, or dialog operations, but should not be an isolated mechanical action or an entire multi-goal workflow. Use the complete task, trajectory, environment, and verifier jointly. Decompose the demonstrated workflow into an ordered primitive sequence.

Requirements:
- Assign every application a 1-based index and associate each primitive with the application where its main state change occurs.
- Express each ability as a reusable operation rather than a replay of low-level actions.
- Record its concrete inputs and observable effects.
- Include only necessary preconditions. Use "requires:<effect>" when a precondition is produced by an earlier primitive.
- Record only environment requirements supported by the provided context, including applications, VM capabilities, files, services, and configuration.
- Derive verification conditions only from the supplied verifier. Do not invent checks; use coverage="none" when the primitive is not independently verifiable.
- Cite the exact inclusive trajectory step range supporting each primitive.
- Use failed execution evidence only to identify failure modes.
- Do not infer operations across missing or unexplained state transitions.
- Keep primitives ordered, non-duplicated, environment-feasible, and dependency-consistent.

Task: <TASK_JSON>
Environment: <ENVIRONMENT_CONTEXT_JSON>
Trajectory outcome: <TRAJECTORY_OUTCOME_JSON>
Trajectory steps: <TRAJECTORY_STEPS_JSON>
Verifier: <VERIFIER_JSON_OR_SOURCE>

Return ONLY JSON with this shape:
{"applications": [{"index": 1, "application": "..."}],"primitives": [{"id": "p1","index": 1,"ability": {"name": "...","intent": "...","inputs": {},"effects": ["..."]},"preconditions": ["..."],"environment": {"required_vm_capabilities": [],"files_or_assets": [],"config_dependencies": []},"verifier": {"coverage": "full|partial|none","observable_postconditions": [],"fragments": []},"source_evidence": {"task_id": "...","episode_id": "...","step_start": 1,"step_end": 3,"success": true},"failure_modes": [],"confidence": 0.0}],"uncovered_steps": [{"step_start": 4, "step_end": 4, "reason": "..."}]}

Figure 16 Prompt for reusable ability primitive extraction.

You are synthesizing one executable and verifiable Computer-Use Agent (CUA) task from reusable application-level primitives. Select exactly K compatible primitives and arrange them in executable dependency order. Their preconditions and environment requirements must be satisfied by the initial environment or preceding primitives. The composition must form one coherent workflow rather than unrelated operations. For a cross-application task, include an explicit handoff artifact or shared state between applications. Instantiate all required files, values, and application states with concrete and consistent entities. Generate a concise, goal-oriented instruction that covers the selected primitives without exposing primitive identifiers or low-level action steps. Merge the primitive-level environment requirements into one executable environment specification, and consolidate their verifier specifications into one task-level verifier. Every requested outcome must be observable and verified, and every verifier check must correspond to the instruction. Do not invent unavailable resources, capabilities, or success criteria. If no valid composition of K primitives exists, return valid=false.

Application set: <APPLICATION_SET>
Complexity budget: <COMPLEXITY_BUDGET_K>
Target environment: <ENVIRONMENT_SPEC_JSON>
Primitive library: <PRIMITIVE_LIBRARY_JSON>

Return ONLY JSON with this shape:
{"valid": true,"selected_primitives": ["p1", "p2"],"dependency_order": ["p1", "p2"],"task": {"instruction": "...","environment": {"applications": [],"initial_state": [],"assets": [],"requirements": []},"verifier": {"success_logic": "all","checks": [{"target": "...","condition": "...","expected": "..."}]}},"confidence":0.0,"reason": ""}

Figure 17 Prompt for composable task and verifier synthesis.

Trajectory observation 1: <TRAJECTORY_IMAGE_1>
...
Trajectory observation N: <TRAJECTORY_IMAGE_N>

You are diagnosing a failed Computer-Use Agent (CUA) rollout for snapshot-rollback retry. Given the task, verifier residuals, trajectory, environment state, and available snapshots, identify the earliest action step whose resulting state diverges from the task objective or verifier requirements. Do not simply select the final failed step; preserve the longest valid trajectory prefix and select the latest valid snapshot preceding the divergence.

Generate concise, execution-grounded feedback describing the observed failure and unmet verification condition. Do not provide a privileged solution, such as exact future actions, click coordinates, shell commands, or a hidden reference trajectory. If the failure is caused by the environment or verifier, or cannot be reliably localized, set the corresponding failure_source and return null for unsupported fields.

Failure source: agent, environment, verifier, or inconclusive.
Task: <TASK>
Environment: <ENVIRONMENT_SPEC>
Verifier residuals: <VERIFIER_RESIDUALS>
Trajectory: <TRAJECTORY_STEPS_JSON>
Snapshots: <SNAPSHOT_SIGNATURES_JSON>

Return ONLY JSON with this shape:
{"failure_source":"agent","earliest_divergence_step":7,"rollback_snapshot_id":"snapshot_6","feedback":"The output artifact is not present at the verifier-checked location after step 7.","evidence":"The state after step 7 first conflicts with the required file state.","confidence":0.9}

Figure 18 Prompt for execution failure localization and diagnostic feedback generation.

Table 9 Taxonomy of domains, subdomains, and applications.

Domain	Sub Domain	Applications
Business &Administration	Career	ADP Workforce Now, Indeed, Glassdoor, Greenhouse, Workable, Simplify Jobs, We Work Remotely, Idealist, CharityVillage, LinkedIn, ZipRecruiter, BambooHR, Workday, Upwork, Fiverr
	Education	Moodle, OpenBoard, GeoGebra, Google Classroom, Canvas LMS, Blackboard, Kahoot, Quizlet, Anki, Coursera, Khan Academy, Duolingo, edX, LeetCode, Outschool, RateMyProfessors, Udemy, Skillshare, Codecademy, Grammarly, Wolfram Alpha, Photomath, Chegg, Schoology, Pluralsight, DataCamp, HackerRank, freeCodeCamp
	Finance	QuickBooks, Xero, GnuCash, FreshBooks, Mint, QuantLib, Pro Expense, Stocks, PayPal, Revolut, YNAB, Zoho Books, Quicken, TurboTax, Robinhood, Coinbase
	Sales & Marketing	Salesforce, SuiteCRM, G2, Capterra, Trustpilot, Mailchimp, Freshdesk, Twenty, HubSpot, Zoho CRM, Pipedrive, Zendesk, Constant Contact, Klaviyo, Hootsuite
Creative Production	Audio & Music	MuseScore, Dorico 6, Cubase, Audacity, FL Studio, Ableton Live, Logic Pro, Pro Tools, Ardour, LMMS, GarageBand, Adobe Audition, Avid Sibelius, Guitar Pro, REAPER, Retro Music, Audio Recorder, Music, Voice Memos
	Graphic Design	GIMP, Inkscape, Microsoft Paint, darktable, Krita, Scribus, Adobe Photoshop, Adobe Illustrator, Adobe InDesign, Affinity Photo, Affinity Designer, CorelDRAW, Figma, Canva, RawTherapee, Adobe Lightroom Classic, Font Book, PhotoPrism
	Media & Games	Blender, Ruffle, Godot Engine, GNOME Mines, GNU Chess, Hedgewars, SuperTux, XMoto, PySol, Stockfish, PokerTH, Sokoban, Sabaki, Unity, Unreal Engine, GameMaker Studio, Construct 3, Aseprite, Twine, Minecraft, Live2D Cubism, Houdini, ZBrush, Dungeon Crawl Stone Soup, NetHack, OpenRCT2, OpenTTD, The Battle for Wesnoth, Chess
	Post-production	Shotcut, Adobe After Effects, DaVinci Resolve, VLC, MPV, Adobe Premiere Pro, Final Cut Pro, HandBrake, OBS Studio, Kdenlive, OpenShot, Lightworks, Nuke, Natron, DaVinci Fusion, iMovie, QuickTime Player
Daily Life	Activities & Events	Eventix, Handy, TaskRabbit, StyleSeat, Bark, Eventbrite, Ticketmaster, Pretix, Netflix, Spotify, Twitch, Disney+, Prime Video, Plex, Jellyfin, Kodi, SeatGeek, StubHub, Fandango
	Food & Cooking	Broccoli, Eat This Much, MyRecipes, BeerAdvocate, Untappd, Vivino, Yummly, Allrecipes, Paprika, Mealime, Tandoor Recipes, Yelp, OpenTable, DoorDash, Uber Eats, Grubhub
	Shopping	Etsy, Target, Lowe's, The Ordinary, Amazon, eBay, Walmart, AliExpress, Best Buy, IKEA, Shein, Temu, Costco, Home Depot, Shopify
	Social & Community	IMDb, Goodreads, Trakt, BoardGameGeek, Ravelry, Petfinder, ASPCA, Red Cross, Change.org, Spirit Rock, Strava, Reddit, Nextdoor, Meetup, WhatsApp, Telegram, Mastodon, GoFundMe
	Travel & Booking	Tripadvisor, TripIt, AutoSlash, OsmAnd, Maps, Google Maps, Booking.com, Airbnb, Expedia, Skyscanner, Kayak, Google Flights, Hopper, Rome2Rio, Uber, Lyft
Data & Analytics	Database	DuckDB, ClickHouse, PostgreSQL, pgAdmin, MySQL Workbench, SQLite, MongoDB, Redis, MariaDB, Microsoft SQL Server, Oracle Database, Cassandra, Neo4j, DBeaver, MongoDB Compass, ClickHouse UI, Baserow
	Pipeline	Apache Airflow, dbt, RabbitMQ, Nextflow, Apache Kafka, Apache NiFi, Apache Flink, Prefect, Dagster, Pentaho Data Integration, n8n, dbt Docs
	Processing	Rscript, qpdf, Ghostscript, ImageMagick, Pandoc, Tesseract OCR, xmllint, FFmpeg, jq, sed, gawk, sox, exiftool, wkhtmltopdf
	Statistical Analysis	RStudio, SPSS, SAS, Stata, JASP, jamovi, PSPP
	Visualization	LabPlot, Grafana, Streamlit, TensorBoard, Metabase, Prometheus, Alertmanager, Jaeger, RabbitMQ Management, GoAccess, Performance Co-Pilot, Weights & Biases, Tableau, Power BI, Apache Superset, Kibana, Redash, Looker Studio, Datadog, GrowthBook, Qlik, Bokeh
Engineering	Drafting	FreeCAD, SolveSpace, OpenSCAD, Rhino 8, Autodesk Civil 3D, QGIS, MeshLab, CloudCompare, COLMAP, LibreCAD, QCAD, draw.io, AutoCAD, SolidWorks, Autodesk Fusion 360, SketchUp, Autodesk Inventor, CATIA, Autodesk Revit, Onshape, Gaussian Splatting, 3DEXPERIENCE, Grasshopper, Mermaid
	Hardware Design	KiCad, Logisim, LTspice, Verilator, GTKWave, Yosys, sbt, LIBERO, Altium Designer, Autodesk EAGLE, Intel Quartus, Xilinx Vivado, ModelSim, Proteus, OpenROAD, ngspice
	Industrial Control	Autodesk PowerMill, Moldex3D, LabVIEW, Siemens TIA Portal, CODESYS, QGroundControl
	Simulation	Elmer FEM, FDS, Smokeview, EnergyPlus, PLAXIS 3D, Isaac Lab, Isaac Sim, MuJoCo, ANSYS, COMSOL Multiphysics, OpenFOAM, Simulink, Abaqus, Gazebo, ParaView, SUMO, SALOME, OpenSees, SU2, pandapower, WRF, SimPEG
Productivity	Collaboration	Microsoft Outlook, Thunderbird, Outlook Calendar, Slack, Microsoft Teams, Zoom, Discord, Google Meet, Gmail, Cisco Webex, Mattermost, Zimbra, Rocket.Chat, Calendar, Doodle, When2meet, Typeform, FormSwift, Contacts, Freeform

Category	Subcategory	Applications
	Management	SAP ERP, Flowable, Odoo, Obsidian, FOLIO, XMind, Notion, ClickUp, Monday.com, Airtable, Asana, Trello, Microsoft Project, Wrike, Evernote, Jira, Confluence, Markor, Joplin, Tasks, Todoist, Habitica, Notes, Reminders, OpenProject, farmOS, SiYuan, Basecamp, Smartsheet, TickTick, Microsoft To Do, Google Keep, Redmine, Zoho Projects
	Office Suite	Microsoft Word, Microsoft Excel, Microsoft PowerPoint, Adobe Acrobat, Microsoft Edge PDF Viewer, LibreOffice Writer, LibreOffice Calc, LibreOffice Impress, WPS Presentation, WPS Spreadsheet, Okular, Google Docs, Google Sheets, Google Slides, OnlyOffice, Apple Pages, Apple Numbers, Apple Keynote, Foxit PDF Reader, Sumatra PDF, PDF-XChange Editor, WPS Office, Preview
	System Tool	File Explorer, Windows Photo Viewer, Windows Photos, Snipping Tool, Baobab, Flameshot, Total Commander, 7-Zip, WinRAR, Nautilus, FileZilla, PeaZip, Windows Image Viewer, tar, CUPS, dconf Editor, Dunst, GNOME Settings, GNOME System Monitor, Thunar, Simple Gallery Pro, Activity Monitor, Automator, Calculator, ColorSync Utility, Dictionary, Disk Utility, Finder, System Settings, Digital Color Meter, Weather, ownCloud
Science & Research	Biology & Genomics	BEDTools, BWA, SAMtools, BCFtools, GATK, VEP, UCSC Genome Browser, RGI, NCBI BLAST+, HISAT2, Bowtie2, IGV, Picard, Trimmomatic, FastQC, Bandage, MultiQC, sarek, QIIME 2, CellProfiler, Cellpose, Fiji (ImageJ), napari, QuPath, starfish
	Medical Science	3D Slicer, MicroDicom DICOM Viewer, ITK-SNAP, OsiriX, Horos, Weasis, FSLeyes, MONAI, OpenSim, PhysioNet, NextGen Healthcare, OpenEMR
	Physics & Chemistry	Ketcher, Avogadro, PyMOL, VMD, GROMACS, UCSF ChimeraX, Jmol, LEAP, Amber, PySCF
	Research Workflows	Zotero, Overleaf, LaTeX, Sigil, Mendeley, EndNote, JabRef, TeXstudio, Texmaker, Kile, Tropy, Semantic Scholar
	Scientific Computing	JupyterLab, Spyder, GNU Octave, MATLAB, Wolfram Mathematica, Maple, SageMath, Scilab, Maxima, Anaconda Navigator, GAP, National Weather Service, METplus, Lean 4, Coq
Developer Tools	Code Editing	Visual Studio Code, Emacs, Bluefish, Sublime Text, Vim, Neovim, Notepad++, IntelliJ IDEA, PyCharm, Android Studio, NetBeans, Zed, Script Editor, Xcode, code-server
	Dev & Ops	PowerShell, Windows Terminal, Bash, GNU Coreutils, curl, grep, uv, Docker, Docker Compose, Minikube, kubectl, Helm, Headlamp, Terraform, AWS Management Console, SLURM, Nginx, Git, GitLab, GitHub CLI, Ansible, Jenkins, Podman, Vagrant, tmux, htop, rsync, Azure CLI, Google Cloud CLI, Kubernetes, vLLM, GitHub
	Security	Wireshark, Ghidra, Trivy, Nmap, Metasploit, Burp Suite, Hashcat, OpenSSL, Aircrack-ng, Snort, tcpdump, Nikto, Zeek, Volatility, Sleuth Kit
	Testing	GDB, Valgrind, strace, bpftrace, iotop, FlameGraph, pytest, JUnit, Selenium, Cypress, Playwright, LLDB, perf, JMeter, k6
	Web	Google Chrome, Microsoft Edge, Chrome DevTools, Chrome Lighthouse, Electron, SVGO, Mozilla Firefox, Safari, Brave, Postman, webpack, Vite, npm, WordPress, Chromium, Substack, Webflow