ADAPTATION OF COMPUTER USE AGENTS TO NEW SOFTWARE WITH USER MANUALS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in multimodal foundation models have significantly improved the ability to operate real-world software through graphical user interfaces (GUIs) using visual observations and natural language instructions. However, maintaining strong performance on unseen applications remains challenging, as practical tasks often require long-horizon procedural knowledge and application-specific operation patterns. In addition, constructing executable tasks and collecting corresponding trajectories for training still heavily rely on human-annotated data, incurring substantial cost. In this paper, we propose MAGE (Manual-augmented computer use **AG**ent **E**xecution and learning), a framework that leverages user manuals as a scalable source of procedural knowledge to improve both inference and training. During execution, MAGE retrieves task-relevant manual content and uses a planner to generate explicit procedural guidance for CUAs. To further account for the discrepancies between manuals and actual GUI environments, such as version differences or user settings, MAGE also adapts the agent policy through environment interaction. During training, MAGE converts manuals into executable task-plan pairs and uses the plans to guide environment rollouts toward task-relevant states; the resulting trajectories are then used to optimize the policy without human-authored demonstrations. This unified design enables effective utilization of manuals for both task execution and training, reducing reliance on human-authored training data. Experimental results across six applications demonstrate improvements in task success rates over strong baselines, highlighting the effectiveness of manual-guided planning and policy adaptation in new software environments. We provide the code and reproduction scripts in an anonymized repository: https://anonymous.4open.science/r/MAGE-8F36

1 INTRODUCTION

Recent advances in multimodal vision-language models (VLMs) (Yin et al., 2024) have enabled Computer Use Agents (CUAs) (Zhang et al., 2024; Lin et al., 2024; Qin et al., 2025; Wang et al., 2025) to operate software from visual observations and natural language instructions. Unlike conventional automation methods that rely on APIs or HTML structures, these agents interact directly with graphical user interfaces (GUIs) in a human-like manner. This enables CUAs to perform various tasks across diverse software environments.

However, existing CUAs require additional training to maintain high performance on unfamiliar applications since each application has its own interface and functionalities. Existing adaptation (Qin et al., 2025; Zhang et al., 2025a; Yuan et al., 2025) typically relies on costly human annotations to define diverse executable tasks and collect corresponding action trajectories. Although reinforcement learning (Sutton et al., 1998; Silver & Sutton, 2025; Shao et al., 2024) can reduce the need for human demonstrations, GUI tasks are inherently long-horizon: agents should sequentially uncover available options through clicks, menus, and dialogs rather than observing all relevant information at once. This makes it difficult to obtain sufficient successful trajectories, limiting the training signals required to improve the agent policy.

To address these limitations, we leverage user manuals as a source of procedural supervision. Manuals describe application functionalities and the procedures required to execute them, including functions hidden behind menus, dialogs, and toolbars. By exposing access routes to such functions, manuals can

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

reduce unguided exploration in long-horizon GUI tasks. Prior approaches have also leveraged external knowledge, such as web tutorials and past interaction experiences, to guide GUI agents (Xu et al., 2025a; Agashe et al., 2025). However, providing external knowledge only during execution does not directly address the challenge of collecting successful trajectories for policy adaptation. Moreover, manuals may be abstract or differ from actual GUI environments due to software versions, user settings, or access permissions. Thus, agents must ground procedural guidance through environment interaction and learn which actions succeed in the current GUI state.

Based on this observation, we propose MAGE (Manual-augmented computer use **AG**ent **E**xecution and learning), a unified framework that leverages user manuals to improve both execution-time planning and training-time policy optimization. During execution, MAGE retrieves task-relevant manual content and uses a planner to generate task-specific procedural plans that guide CUAs. During training, it automatically generates diverse executable tasks and corresponding plans from user manuals. Rather than treating generated plans as expert demonstrations, MAGE evaluates the collected trajectories and optimizes the agent policy using Group Relative Policy Optimization (GRPO). This joint use of manuals for planning and policy learning enables application-specific adaptation without human-authored demonstrations or step-level action annotations.

Our main contributions are summarized as follows.

- We propose MAGE, a unified framework that uses user manuals as procedural supervision for both execution-time planning and training-time policy learning, combining a planner with CUAs.
- We introduce retrieval-augmented planning, which retrieves task-relevant manual content and generates procedural plans to guide long-horizon GUI execution.
- We develop a manual-derived training method that constructs task-plan pairs and optimizes the policy from the resulting trajectories without human-authored demonstrations or step-level annotations.
- We evaluate MAGE across six applications and demonstrate improvements in task success rates over the corresponding baselines, examining the contributions of manual-guided planning and policy optimization.

2 Related Work

Computer use agents (CUAs). To mitigate reliance on specific interfaces or predefined structural representations in computer environments, VLM-based agents that understand GUIs purely from visual signals have been proposed, with representative examples including OpenCUA (Wang et al., 2025) and UI-TARS (Qin et al., 2025). Training these foundation models requires large and diverse datasets. For instance, OpenCUA (Wang et al., 2025) integrates datasets from prior works such as ShowUI (Lin et al., 2024) and UGround (Qian et al., 2025), along with general supervised fine-tuning (SFT) data from the Kimi Team (Team et al., 2025), and further incorporates demonstration data collected from Ubuntu, Windows, and macOS environments. Similarly, UI-TARS (Qin et al., 2025) constructs additional training data by allowing human annotators to refine and correct trajectories generated by the model. While such data-centric training strategies significantly improve model performance, adapting these models to new software environments requires additional data collection and annotation tailored to each target domain. This process incurs substantial cost, thereby limiting the scalability and practical applicability of CUAs.

Reducing human supervision for CUA adaptation. Prior work has explored reducing human supervision for CUA adaptation by augmenting or reusing existing datasets for training, or by leveraging external knowledge to guide task execution. BTL-UI (Zhang et al., 2025a) augments existing datasets with ROI annotations, while ZeroGUI (Yang et al., 2025) and DART (Li et al., 2025) leverage benchmark-derived tasks for reinforcement learning. Meanwhile, RAG-GUI (Xu et al., 2025a) trains a VLM to generate guidelines from web tutorials, guiding a frozen CUA at inference time, and Agent S (Agashe et al., 2025) uses external knowledge for hierarchical planning. However, data-driven approaches still rely on task-specific data, while knowledge-based approaches primarily assist execution at inference time without adapting the CUA's policy. In contrast, MAGE leverages user manuals for both planning and policy optimization, improving task execution without repeatedly generating plans throughout an episode.

3 METHOD

MAGE is a manual-augmented adaptation framework for CUAs. Given a target program and its user manual, MAGE uses the manual as procedural supervision in two complementary ways: it retrieves task-relevant instructions to guide long-horizon execution, and it generates executable task-plan pairs to support policy training. These two components address the main bottlenecks in GUI adaptation: reliable task execution and scalable collection of successful trajectories.

3.1 PROBLEM DEFINITION

Given a natural language task instruction $\mathcal{I}$, our goal is to complete the task $\mathcal{I}$ by performing a sequence of actions. At each time step t, the agent observes a screenshot o_t, generates an action a_t, and receives the next observation o_{t+1}. This process forms a trajectory $\tau = \{(o_1, a_1), (o_2, a_2), \ldots, (o_T, a_T)\}$ where T is the length of the trajectory. The action space includes common mouse and keyboard actions such as click, right click, double click, scroll, drag, and typing. Please refer to Appendix E for the detailed descriptions of each action and its arguments.

3.2 TASK EXECUTION WITH RETRIEVAL-AUGMENTED PLANNING

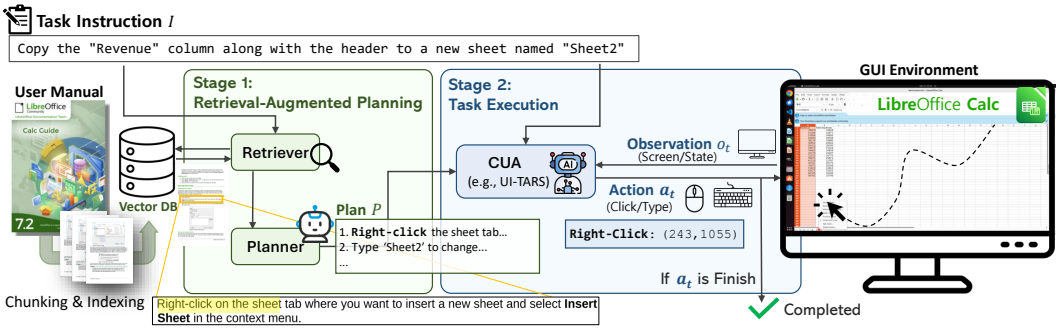


Figure 1: Task execution with user manuals. MAGE retrieves relevant manual sections, then generates a retrieval-augmented plan, and finally performs GUI task execution conditioned on the plan.

CUAs execute a task by repeatedly selecting the most appropriate action a_t based on the current observation o_t and the past trajectory $\tau_{<t} = \{(o_j, a_j) | 1 \leq j < t\}$. However, in many GUI environments, the information required for the task completion is not fully visible in the current screenshot. Important functions are often hidden behind menus, dialogs, toolbars, or keyboard shortcuts, degrading the performance of agents for long-horizon tasks. To address the problem, we propose retrieval-augmented planning which uses procedural guidance from user manuals during task execution.

As shown in Figure 1, our task execution consists of two stages. In the first stage, MAGE performs retrieval-augmented planning, following the retrieval-augmented generation framework (Lewis et al., 2020). It first retrieves relevant manual sections to provide detailed procedural information to the planner and generates an execution plan consisting of a sequence of actions with specific information such as button names and keyboard shortcuts. In the next stage, MAGE executes the task similarly to conventional CUAs but with the execution plan.

Retrieval-augmented planning. MAGE retrieves manual sections relevant to $\mathcal{I}$ and generates a task-specific plan. Let $\mathcal{M} = \{m_1, \ldots, m_{|M|}\}$ denote the set of manual sections. We encode each manual section m_i using an embedding model as $\mathbf{h}_i = E(m_i)$. The embedding vectors are indexed in a vector database before executing the CUA. To retrieve the relevant section, MAGE encodes the task instruction as $\mathbf{q} = E(\mathcal{I})$ and finds the top-K sections $\mathcal{M}_\mathcal{I}$ with the highest cosine similarity $\cos(\mathbf{q}, \mathbf{h}_i)$. The planner then takes the task instruction $\mathcal{I}$, the relevant sections $\mathcal{M}_\mathcal{I}$, and the initial observation o_1 as inputs, and generates a plan $\mathcal{P}$ that includes step-by-step procedures with specific actions. We use a multimodal LLM as the planner and provide its prompt in Appendix F.2.

Task execution with the generated plan. Given the generated plan, MAGE executes the task by conditioning the CUA on the task instruction $\mathcal{I}$, the plan $\mathcal{P}$, the current observation o_t, and the past

trajectory $\tau_{<t}$. At each time t, the agent selects the next action as

$$a_t \sim \pi_\theta(\cdot|o_t, \tau_{<t}, \mathcal{P}, \mathcal{I}). \tag{1}$$

The plan remains fixed throughout the episode, while the CUA selects actions based on the current observation and interaction history.

3.3 Training CUAs with Manual-Derived Tasks

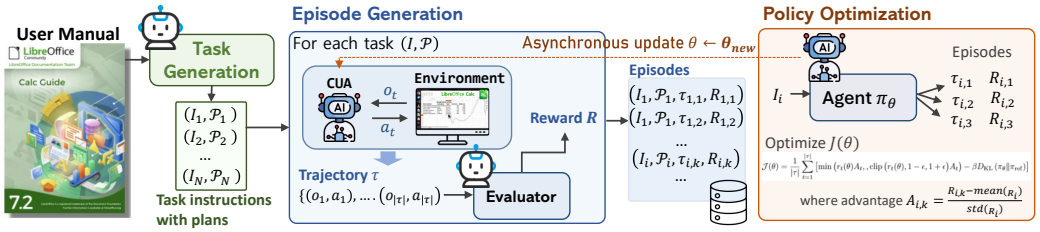


Figure 2: Training CUAs with a user manual. MAGE uses manual-derived task-plan pairs to collect episodes for policy optimization.

Retrieval-augmented planning provides procedural guidance, but manuals do not fully specify the actual GUI state of a target environment. In addition, interface elements can be hidden, customized, or restricted depending on application settings, workspace configurations, and user permissions. Thus, even when a manual describes a valid procedure, the agent should learn how to execute it robustly in the observed environment. To address this problem, we utilize user manuals not only for execution-time planning but also for policy training.

As shown in Figure 2, the training pipeline consists of three stages: (1) task generation, (2) episode generation, and (3) policy optimization. In the first stage, MAGE uses the manual to construct a set of executable task-plan pairs. This stage is performed once before training begins. Second, the current policy executes these task-plan pairs in the target GUI environment to collect episodes. Third, the collected episodes are evaluated and used to optimize the policy via GRPO (Shao et al., 2024). The second and third stages are repeated iteratively, so that each updated policy produces new episodes for subsequent optimization.

3.3.1 Task Generation

In this stage, MAGE generates a wide range of executable task instructions from the manual. Since policy optimization requires successful trajectories, training becomes ineffective when the initial policy rarely completes long-horizon GUI tasks. To address this, we generate plans together with tasks to guide execution and steer rollouts toward task-completing trajectories.

For each manual section, MAGE uses both the textual content and images as inputs. A multimodal LLM interprets the functional descriptions and examples in the section, and generates a setup task $\mathcal{I}^{\text{setup}}$, a setup plan $\mathcal{P}^{\text{setup}}$, a main task $\mathcal{I}^{\text{main}}$, and a main plan $\mathcal{P}^{\text{main}}$. The setup task is an optional preliminary task that configures the initial GUI state required to execute the main task, while the main task is the final task that requires using the functionality described in the corresponding manual section. Each plan provides step-by-step procedural guidance for executing its corresponding task.

By explicitly separating setup tasks, the framework can generate and execute diverse tasks without requiring manually prepared initial environments. The resulting setup and main tasks naturally involve common operations in the target program, such as entering values into spreadsheet cells or navigating application menus. Training on these tasks exposes the agent to frequently used program operations while covering diverse functionalities and usage procedures described in the manual. The prompt template for task generation is provided in Appendix F.1.

3.3.2 Episode Generation

In this stage, trajectories are constructed by executing the generated task pairs in the GUI environment. The trajectory collection follows the interaction process described in Section 3.2. When a setup task is included in a task pair, the agent first executes the setup task to configure the required GUI

state, and proceeds to the main task only upon successful completion. If there is no setup task, the agent directly executes the main task from the initial state. Each task execution generates a trajectory $\tau = \{(o_1, a_1), (o_2, a_2), \ldots, (o_T, a_T)\}$ consisting of a sequence of observations and actions.

To determine task success, we adopt the LLM-as-a-judge paradigm and employ a vision-language model (VLM) as an evaluator. The evaluator assigns a binary success reward $R^{\text{success}} \in \{0, 1\}$ to each trajectory, where $R^{\text{success}} = 1$ if the task is judged as successful and $R^{\text{success}} = 0$ otherwise. However, prior work (Yang et al., 2025) has shown that evaluating full trajectories can be unreliable, as important details may be overlooked and hallucinations may lead to incorrect judgments. To mitigate this issue, we determine task success from the task instruction and the last five screenshots of the trajectory. This design aims to reduce susceptibility to hallucinations and evaluation cost. We provide a detailed analysis of the evaluator in Appendix D.3.

For policy optimization, we construct training episodes from the collected task executions. Setup and main tasks are handled as separate training instances rather than being merged into a single trajectory. Accordingly, each task execution produces a separate training episode, defined as a tuple $(\mathcal{I}, \mathcal{P}, \tau, R^{\text{success}})$. Each task is executed multiple times to collect a diverse set of episodes, which are subsequently used for GRPO-based policy optimization.

3.3.3 Policy Optimization with Manual-Derived Episodes

In our adaptation setting, the amount of interaction data and available computation are limited compared to general foundation model training. Thus, we utilize GRPO (Group Relative Policy Optimization) (Shao et al., 2024) to optimize the CUA from collected episodes instead of using other reinforcement learning algorithms requiring a separate value model, such as PPO (Schulman et al., 2017).

Round-based training. GRPO requires multiple rollouts per task to estimate group-level statistics, and learning becomes more effective as the proportion of episodes with positive rewards increases. However, generating new rollouts at every training step incurs substantial computational cost. We define a *round* as a cycle of episode generation and GRPO-based training. To address this, we propose a round-based training strategy that builds on an offline learning setup by reusing existing data while incrementally incorporating rollouts generated by the latest policy. In each round, newly generated episodes are added to the existing dataset and used for subsequent training. By repeating this process, rollouts with higher success rates are progressively collected, enabling the dataset to expand and the policy to improve over time.

Episode sampling. Instead of computing expectations over all episodes within a group, we adopt a sampling-based approximation. For each task, one episode is randomly sampled for training, while its advantage is computed using all available episodes in the group. In this process, setup tasks and main tasks are treated as distinct, independent task instances, and sampling is performed separately for each task. Due to the dependency between setup and main tasks, not all attempts result in valid episodes, and multiple trials for a given task may yield only a limited number of episodes. In particular, when all episodes for a given task have identical rewards or when only a single episode is available, the reward variance becomes zero, making advantage estimation ill-defined. To address this issue, we exclude episode groups with zero reward variance from policy optimization. This approach reduces computational overhead by reusing previously collected episodes rather than generating new rollouts at every training step.

Reward design. We define the reward function based on both task success and execution efficiency. Let τ_i denote the i-th trajectory with length T_i, and R_i denote the corresponding trajectory-level reward. We use the success reward R_i^{success} defined in Section 3.3.2. To encourage efficient execution, we define a step-based reward that is only applied upon success:

$$R_i^{\text{step}} = \begin{cases} \frac{T_{\max} - T_i}{T_{\max} - T_{\min}}, & \text{if } R_i^{\text{success}} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Here, $T_{\max}$ and $T_{\min}$ denote the maximum and minimum trajectory lengths among episodes of the same task. The step-efficiency reward is set to zero when $T_{\max} = T_{\min}$. The total reward is given by:

$$R_i = R_i^{\text{success}} + \lambda R_i^{\text{step}}, \tag{3}$$

where λ is a weighting coefficient.

Advantage estimation. Advantage is computed using the set of episodes corresponding to the same task. Let $\mathcal{D}_u = \{\tau_i\}_{i=1}^{N}$ denote the set of trajectories for task u. We compute the normalized advantage as:

$$A_{i,k} = \frac{R_{i,k} - \text{mean}(\mathcal{R}_i)}{\text{std}(\mathcal{R}_i)}. \tag{4}$$

Here, i indexes the task, and k indexes the k-th trajectory (episode) associated with task i. $\mathcal{R}_i = \{R_{i,k}\}_{k=1}^{K_i}$ denotes the set of rewards for all trajectories corresponding to task i, and $\text{mean}(\mathcal{R}_i)$ and $\text{std}(\mathcal{R}_i)$ represent the mean and standard deviation over this set, respectively.

Objective function. We optimize the policy using a clipped GRPO objective. Let π_θ denote the current policy and $\pi_{\theta_{\text{old}}}$ the old policy. At the beginning of each training epoch, we reset the old policy to the current policy. For a sampled trajectory (τ), we define the probability ratio at step t as

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid \mathcal{I}, \mathcal{P}, \tau_{<t}, o_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid \mathcal{I}, \mathcal{P}, \tau_{<t}, o_t\right)}. \tag{5}$$

The objective for a sampled episode is:

$$\mathcal{J}(\theta) = \frac{1}{|\tau|}\sum_{t=1}^{|\tau|}\left[\min\left(r_t(\theta)A_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)A_t\right) - \beta D_{\text{KL}}\left(\pi_\theta \| \pi_{\text{ref}}\right)\right], \tag{6}$$

where ϵ is the clipping parameter, β controls the KL penalty, and π_{ref} is the initial CUA policy before adaptation, kept frozen throughout training.

4 Experiments

4.1 Experimental Setup

Benchmarks and applications. We evaluate MAGE on the OSWorld benchmark (Xie et al., 2024) for five applications: LibreOffice Calc, Impress, and Writer, GIMP, and VSCode. These applications require diverse multi-step GUI interactions including spreadsheet editing, slide modification, document editing, image editing, and code-related operations. Since OSWorld is a widely used CUA benchmark, tasks on these applications may already be in CUA training data. To examine whether MAGE can also adapt CUAs to applications with potentially limited coverage in their training, we construct 30 new evaluation tasks for Slack, a workplace communication platform.

Models. We evaluate our framework with two CUA backbones: UI-TARS-1.5 (7B) (Qin et al., 2025) and EvoCUA (8B) (Xue et al., 2026). For retrieval-augmented planning, we use Qwen3-Embedding-0.6B (Zhang et al., 2025b) for manual retrieval and Qwen3.6 (27B) for plan generation. At the beginning of each task, the planner generates a task-specific plan conditioned on the task instruction, initial screenshot, and retrieved manual content. The generated plan remains fixed throughout execution. We use GPT-5.4 for manual-based task generation and trajectory evaluation during training.

Training and planning setup. For LibreOffice applications, we use the official LibreOffice user manuals[1] for both retrieval-augmented planning and manual-derived task construction. For GIMP[2] and VSCode[3], we use their official documentation, and for Slack, we use the Slack Help Center[4]. We use the Round-based Training strategy described in Section 3.3.3 for policy optimization. Training is performed separately for each application to evaluate application-specific adaptation. Within each application, the training configuration differs between the UI-TARS-1.5 and EvoCUA backbones. Detailed backbone-specific settings, including the number of rounds and episodes generated per round, are provided in Appendix A.2.

Evaluation setup. To account for the inherent stochasticity of CUAs, we evaluate our main models over five independent runs on both OSWorld and Slack and report the mean and standard deviation.

[1] https://documentation.libreoffice.org/en/english-documentation/
[2] https://docs.gimp.org/3.2/en/
[3] https://code.visualstudio.com/docs
[4] https://slack.com/help

Table 1: Task success rates (%) on OSWorld applications under a 50-step budget. For models evaluated over multiple runs, we report mean ± standard deviation. Bold and underlined values indicate the highest and second-highest mean success rates, respectively, among systems with base CUAs of 8B parameters or fewer.

Model	Size	Runs	Calc	Impress	Writer	GIMP	VSCode	Overall
o3	-	1	8.51	4.26	21.74	38.46	21.74	15.66
Claude-Sonnet-4.5	-	1	65.96	57.49	65.22	57.69	69.57	62.66
UI-TARS-DPO (Qin et al., 2025)	72B	1	12.77	25.45	43.48	61.54	47.83	33.11
OpenCUA (Wang et al., 2025)	32B	3	13.48 ± 1.23	38.28 ± 4.26	40.58 ± 2.51	69.23 ± 3.85	55.07 ± 10.94	38.75 ± 1.52
OpenCUA (Wang et al., 2025)	72B	1	31.91	44.66	60.87	76.92	73.91	52.40
GUI-Owl-1.5 (Ye et al., 2025)	32B	1	48.94	63.81	60.87	76.92	86.96	64.45
EvoCUA (Xue et al., 2026)	32B	1	55.32	59.49	69.57	76.92	86.96	66.24
OpenCUA (Wang et al., 2025)	7B	3	13.12 ± 2.15	32.60 ± 4.33	33.33 ± 2.51	43.59 ± 5.88	46.38 ± 2.51	30.82 ± 0.27
TianXi-Action (Cheng et al., 2025)	7B	2	6.38 ± 0.00	38.24 ± 0.05	**54.35** ± 3.07	55.77 ± 2.72	<u>67.39</u> ± 3.07	38.24 ± 1.29
UI-TARS-1.5 (Qin et al., 2025)	7B	5	14.47 ± 6.45	38.15 ± 3.38	40.87 ± 2.38	53.85 ± 3.85	56.52 ± 5.32	36.83 ± 3.09
EvoCUA (Xue et al., 2026)	8B	5	32.77 ± 2.43	35.98 ± 3.38	46.09 ± 5.83	<u>80.00</u> ± 3.22	66.09 ± 1.94	<u>47.54</u> ± 0.96
MAGE (UI-TARS-1.5)	7B	5	<u>34.04</u> ± 3.69	**45.39** ± 1.90	44.35 ± 1.94	60.77 ± 5.70	66.09 ± 3.64	47.31 ± 2.02
MAGE (EvoCUA)	8B	5	**37.02** ± 1.90	<u>45.32</u> ± 3.56	<u>50.43</u> ± 4.96	**81.54** ± 1.72	**76.52** ± 2.38	**53.67** ± 1.73

Table 2: Mean SR comparison on the Slack.

Model	Mean SR ± SD (%)
UI-TARS-1.5	$26.67 \pm 5.27\%$
EvoCUA	$32.67 \pm 4.94\%$
MAGE (UI-TARS-1.5)	$\mathbf{49.33 \pm 11.88\%}$
MAGE (EvoCUA)	$\mathbf{51.33 \pm 8.03\%}$

Table 3: Ablation study on OSWorld Calc.

RAP	PO	SR (%)	Improvement
-	-	14.47	-
-	✓	14.89	+0.42
✓	-	25.53	+11.06
✓	✓	**34.04**	**+19.57**

For OSWorld, we evaluate performance across five applications. Because the number of tasks differs across applications, the overall Success Rate (SR) is micro-averaged by dividing the total number of successful tasks by the total number of tasks across all applications. For the backbone models used in MAGE, the five runs include results from the official OSWorld dashboard (two for UI-TARS-1.5 and one for EvoCUA).

To evaluate MAGE on Slack, we construct a separate test set consisting of 30 task pairs across 15 themes. For Slack evaluation, we use Gemini 3.1 Pro Preview to determine task success from the task instruction and the last five screenshots of each trajectory. When a setup task is present, the agent must complete it before attempting the main task. We report the Success Rate (SR) across all 30 task pairs based on successful completion of the main task. Further details on the Slack evaluation are provided in Appendix B.

4.2 Main Results

OSWorld. As shown in Table 1, MAGE consistently improves performance across all five applications and both backbone models. To assess the statistical significance of the overall success rate improvement, we conduct a two-sided Welch's t-test. The improvements are statistically significant for both UI-TARS-1.5 ($p = 4.13 \times 10^{-4}$) and EvoCUA ($p = 3.63 \times 10^{-4}$). Notable improvements are observed in Calc and VSCode, with gains of 19.57 %p for UI-TARS-1.5 on Calc and 10.43%p for EvoCUA on VSCode. Many tasks in Calc and VSCode can be performed using keyboard shortcuts or application-specific commands, potentially reducing the need for complex menu navigation. We hypothesize that MAGE benefits from retrieving these procedures from user manuals and incorporating them into its plans, helping agents execute tasks more effectively.

Slack. As shown in Table 2, MAGE improves the success rate on Slack by 22.66%p and 18.66%p for UI-TARS-1.5 and EvoCUA, respectively. Both improvements are statistically significant under a two-sided Welch's t-test over five independent runs ($p = 0.0094$ and $p = 0.0035$, respectively). Notably, these gains are larger than those observed on OSWorld for both backbones. The relatively low performance of both backbone models on Slack may reflect limited exposure to Slack-specific tasks during training. Although their exact training data coverage is unknown, Slack is absent from several widely used CUA benchmarks. By leveraging user manuals for planning and policy optimization, MAGE may compensate for this lack of application-specific training experience, leading to larger

performance gains on Slack. These results suggest its potential for adapting to new applications with limited prior training data.

4.3 Analysis

We provide additional analyses of MAGE using UI-TARS-1.5 as the backbone.

4.3.1 Ablation Study

We conduct an ablation study to evaluate the contributions of MAGE's two key components: Retrieval-augmented Planning (RAP) during execution and Policy Optimization (PO) during training. The baseline and MAGE results are averaged over five independent runs, while the other variants are evaluated using a single run. With PO alone, the success rate is 0.42 percentage points (pp) higher than the baseline, whereas with RAP alone, it is 11.06 pp higher than the baseline. Combining RAP and PO provides an additional gain of 8.51 pp over RAP alone, resulting in a total improvement of 19.57 pp over the baseline. These results highlight the effectiveness of manual-guided planning and suggest that PO provides additional benefits when combined with RAP.

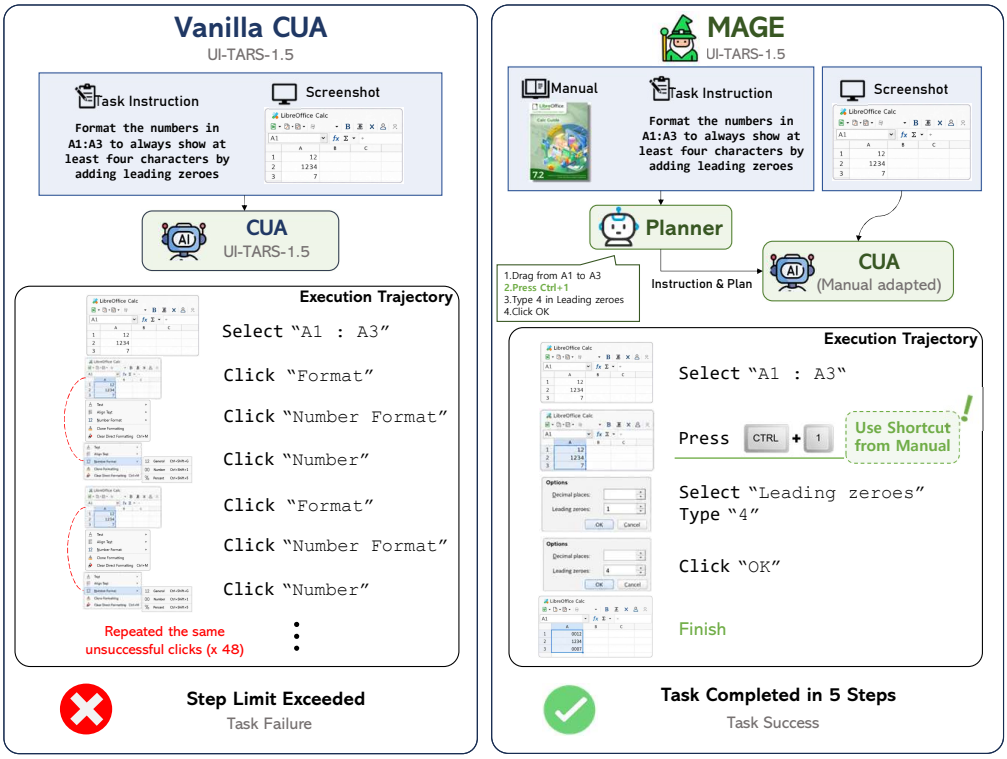


Figure 3: Qualitative comparison of GUI execution on the same task before and after applying MAGE.

4.3.2 Qualitative Evaluation

As shown in figure 3, the vanilla CUA repeatedly executes a sequence of three distinct clicks 16 times, resulting in 48 unnecessary actions before reaching the maximum step budget of 50 and failing the task. In contrast, MAGE retrieves five relevant manual sections (Numbers as Text, Date and Time, Leading Zeroes, Manual Line Break, and Introduction) and generates a task execution plan based on the retrieved information. The resulting plan instructs the agent to use the `Ctrl+1` shortcut to open the Format Cells dialog, enabling it to successfully complete the task in only five steps.

Additionally, when the planner generates a plan without access to the retrieved manual sections, the agent still completes the task in seven steps by opening the Format Cells dialog through mouse clicks. These results show that providing a plan even without user manuals helps the agent avoid

Table 4: Adaptation cost of MAGE across applications using UI-TARS-1.5 as the base agent.

	Calc	Impress	Writer	GIMP	Slack	Average
API Cost ($)	79.80	30.66	33.74	28.95	15.99	37.83
Episode Generation (h)	70.90	26.63	26.13	26.63	33.35	36.73
Policy Optimization (h)	24.24	8.44	8.73	4.24	3.80	9.89

repetitive actions. Incorporating user manuals further enables the agent to leverage documented application-specific functionalities. This allows the agent to execute tasks more efficiently.

4.3.3 Cost Analysis

To quantify the resources required to adapt MAGE to a new application, we analyze the adaptation cost when using UI-TARS-1.5 as the base agent, as summarized in Table 4. Specifically, we measure the monetary API cost, the wall-clock times required for episode generation, and policy optimization. On average across the five applications reported in the table, adapting MAGE to a new application requires 46.62 hours on three NVIDIA H100 GPUs and approximately $37.83 in API costs.

To contextualize the cost of training data generation, OpenCUA (Wang et al., 2025) employed 634 annotators to collect approximately 22,000 tasks over six months. AgentTrek (Xu et al., 2025b) generated approximately 10,400 verified trajectories at an estimated API cost of $5,726.7. Although these figures represent different data collection scales, MAGE demonstrates the feasibility of adapting CUAs to individual applications at a relatively low API cost without human-authored demonstrations.

5 Conclusion

We presented MAGE, a framework that leverages user manuals for both task execution with manual-guided planning and training with manual-derived tasks. In the execution pipeline, MAGE retrieves task-relevant manual sections and uses a planner to generate a plan conditioned on the task instruction and the initial screen state. The generated plan is provided to CUAs, together with the task instruction and trajectory history, allowing them to incorporate external procedural knowledge during action generation. In the training pipeline, MAGE constructs manual-derived tasks from function-level manual sections, collects self-generated trajectories in the GUI environment, assigns rewards through an automatic evaluator, and performs GRPO-based policy optimization. This design integrates manual-based task construction, agent–environment interaction, reward assignment, and policy optimization without requiring human-authored expert demonstrations.

Our experiments demonstrate that incorporating user manuals into planning and policy optimization improves task success across multiple applications. These results suggest that user manuals can be used not only as human-facing documentation, but also as scalable supervision for building more capable CUAs.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets; assist in the writing of proofs; design and provide feedback on research methodology and experiments; implement methods; assist with translation; clean and reformat datasets; and support qualitative and thematic data analysis. We did not use generative AI tools to propose or refine hypotheses, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, or interpret experimental results.

All AI-assisted outputs were reviewed and validated by the authors. In particular, AI-generated tasks and plans were manually examined for usefulness, feasibility, requirement coverage, procedural correctness, executability, and scope safety. AI-assisted implementations were tested through controlled experiments and benchmark evaluations, and automatically generated trajectory assessments were cross-checked against the official rule-based evaluators. AI-assisted proof writing was reviewed by the authors for logical correctness, fidelity to the intended meaning, and consistency with the manuscript. The authors independently analyzed and interpreted the experimental results. We take full responsibility for the final content of this work, including all text, claims, code, data, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work aims to reduce the human demonstration effort required to adapt computer-using agents to target applications. Such agents may also cause unintended operations, data loss, or unauthorized actions. Our experiments evaluate task execution in designated test environments; the results do not establish safe deployment in unrestricted settings. Practical deployment should incorporate restricted permissions, user confirmation for sensitive actions, and appropriate logging. The safeguards discussed here are deployment recommendations rather than guarantees provided by our method.

REPRODUCIBILITY STATEMENT

The main text and appendix describe the task-generation procedure, retrieval-augmented planning, policy optimization, and evaluation protocols. The appendix provides prompt templates, training configurations, action spaces, and additional analyses. To support reproducibility, We release the training and execution code, reproduction scripts, tasks, trajectories, and trained model checkpoints through the anonymized repository linked in the abstract. However, Slack requires account authentication, and releasing its execution trajectories may inadvertently reveal identifying information and compromise author anonymity. We therefore restrict the release of Slack trajectories during the anonymous review process and plan to make them available once the anonymity requirements are lifted.

REFERENCES

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

Yuhao Cheng, Liang Tang, Shuxian Li, Yukang Huo, Tiaonan Duan, Kaer Huang, Yanzhe Jing, and Yiqiang Yan. Evolving in tasks: Empowering the multi-modality large language model as the computer use agent. *arXiv preprint arXiv:2508.04037*, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Pengxiang Li, Zechen Hu, Zirui Shang, Jingrong Wu, Yang Liu, Hui Liu, Zhi Gao, Chenrui Shi, Bofei Zhang, Zihao Zhang, et al. Efficient multi-turn rl for gui agents via decoupled training and adaptive data curation. *arXiv preprint arXiv:2509.23866*, 2025.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for generalist gui agent. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Rui Qian, Xin Yin, Chuanhang Deng, Zhiyuan Peng, Jian Xiong, Wei Zhai, and Dejing Dou. Uground: Towards unified visual grounding with unrolled transformers. *arXiv preprint arXiv:2510.03853*, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

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

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

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, et al. Kimi-vl technical report. *arXiv preprint arXiv:2504.07491*, 2025.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, et al. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

Ran Xu, Kaixin Ma, Wenhao Yu, Hongming Zhang, Joyce C Ho, Carl Yang, and Dong Yu. Retrieval-augmented gui agents with generative guidelines. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17877–17886, 2025a.

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*, volume 2025, pp. 79822–79843, 2025b.

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.

Chenyu Yang, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, et al. Zerogui: Automating online gui learning at zero human cost. *arXiv preprint arXiv:2505.23762*, 2025.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403, 2024.

Xinbin Yuan, Jian Zhang, Kaixin Li, Zhuoxuan Cai, Lujian Yao, Jie Chen, Enguang Wang, Qibin Hou, Jinwei Chen, Peng-Tao Jiang, and Bo Li. SE-GUI: Enhancing visual grounding for GUI agents via self-evolutionary reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=IbzDaIDyt6.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, et al. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*, 2024.

Shaojie Zhang, Ruoceng Zhang, Pei Fu, Shaokang Wang, Jiahui Yang, Xin Du, ShiqiCui, Bin Qin, Ying Huang, Zhenbo Luo, and Jian Luan. BTL-UI: Blink-think-link reasoning model for GUI agent. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=B0Gfxhr8V5.

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

Table 5: Data generation and execution statistics of MAGE across six applications.

	Manual	Tasks		Plan	Trajectory		
Application	**# of Sections**	**w/o Setup**	**All**	**Avg. Length**	**Trajectories**	**Avg. Steps**	**SR (%)**
Calc	536	34	228	7.47	1,665	29.74	52.13
Impress	107	66	102	5.53	609	30.71	47.62
Writer	537	33	105	6.25	710	28.45	46.76
GIMP	1,870	21	100	8.31	536	35.72	35.26
VSCode	1,636	121	260	6.78	1,417	40.52	21.88
Slack	2,045	34	60	6.32	339	35.26	31.27
Average	**1,121.83**	**51.50**	**142.50**	**6.78**	**879.33**	**33.40**	**39.15**

A Experimental Details

A.1 Implementation Details

Table 5 summarizes the statistics of manual sections, generated tasks, plans, and trajectories for each application.

User manuals. We collect publicly available English user manuals for six applications: LibreOffice Calc, Impress, Writer, GIMP, VSCode, and Slack. For LibreOffice, we use Calc Guide 7.1 (May 2021), Impress Guide 7.6 (October 2023), and Writer Guide 7.3 (March 2022), respectively. For GIMP, we use the official GIMP 3.2 documentation. For VSCode and Slack, we collect documentation from their respective official websites, including the VSCode documentation and Slack Help Center.

Manual section segmentation. We segment user manuals into section-level chunks based on their document structure. For LibreOffice, we divide the PDF manuals according to their tables of contents. For web-based documentation, including GIMP, VSCode, and Slack, we use HTML headings to define section boundaries.

Task generation. Before task generation, we exclude manual sections covering introductory material, application overviews, version updates, file sharing, and operations requiring external resources or integration with other applications. For each retained section, we generate up to two tasks. We then sample a subset of the generated tasks to keep the number of training tasks manageable. For Slack, we further exclude features unavailable in a free workspace with administrator privileges.

Trajectory generation. For each generated task, we collect trajectories over five independent runs. We report the number of generated trajectories, their average number of steps, and the training success rate (SR).

Agent inference setting. Although the formulation conditions action generation on the full trajectory, our implementation uses only the five most recent screenshots as trajectory context. For fair comparison, we retain the OSWorld-provided configurations for UI-TARS-1.5 and EvoCUA during both training and inference. For both trajectory generation and evaluation, we use a maximum interaction budget of 50 steps.

A.2 Training Details

Unless otherwise specified, training experiments are conducted on a server equipped with three NVIDIA H100 GPUs.

Common hyperparameters. For both backbones, we set the step-efficiency reward weight to $\lambda = 0.1$, the clipping parameter to $\epsilon = 0.2$, and the KL penalty coefficient to $\beta = 0.1$. These values remain fixed throughout training, without scheduling or adaptive adjustment. For retrieval-augmented planning, we retrieve the top-K manual sections with $K = 5$, unless otherwise specified.

UI-TARS-1.5. We fine-tune UI-TARS-1.5-7B using LoRA (Hu et al., 2021) with rank 16 and scaling parameter $\alpha = 32$, applied to the query, value, and output projection layers. We use the Adam optimizer with a learning rate of 1×10^{-5}. Training is conducted over two rounds. In the first round,

Table 6: Slack evaluation with and without setup tasks.

Method	Gemini 3.1 Pro Preview	
	w/o Setup (18 tasks)	All (30 tasks)
UI-TARS	27.78 ± 6.80	26.67 ± 5.27
MAGE	$\mathbf{56.67 \pm 6.09}$	$\mathbf{49.33 \pm 11.88}$

we generate three rollouts per task and train for two epochs. In the second round, we generate two rollouts per task and again train for two epochs.

EvoCUA. We fine-tune EvoCUA-8B using LoRA with rank 8 and scaling parameter $\alpha = 16$, applied to the query, value, and output projection layers. We use the Adam optimizer with a learning rate of 1×10^{-6}. We observed that EvoCUA tends to overfit more quickly than UI-TARS-1.5 under our training setup, and therefore conduct only a single training round for EvoCUA. In this round, we generate five rollouts per task and train for two epochs.

A.3 Computational Cost for VSCode

Unlike the other applications, the VSCode experiments were conducted on six NVIDIA RTX 4090 GPUs due to hardware availability. Adapting MAGE to VSCode incurred an API cost of $91.25, with 73.43 hours required for episode generation and 34.65 hours for policy optimization, resulting in a total wall-clock time of 108.08 hours. Because the VSCode experiments were conducted under a different GPU configuration from the H100-based experiments, we report these computational costs separately rather than directly comparing their wall-clock time with those of the other applications.

B Slack Test Set Construction and Robustness

This section describes the construction of the Slack test set and evaluates the robustness of our results to task structure.

Slack test set construction. To minimize potential bias toward the training data, we construct the Slack test set independently of the user manuals used for training. We define 30 evaluation tasks across 15 functional themes, including messaging, channel management, and workspace administration. For tasks requiring an initial setup, we use GPT-5.4 to construct corresponding setup tasks. The resulting test set consists of 18 main-only tasks and 12 setup–main task pairs.

Setup-free evaluation. We further examine whether the performance gains depend on the setup–main task structure. To this end, we evaluate the 18 tasks that require no preceding setup. As shown in Table 6, MAGE continues to outperform UI-TARS-1.5 on this subset. These results suggest that the improvements also extend to tasks without a separate setup stage.

C Analysis of Retrieval-Augmented Planning

Unless otherwise specified, all analyses in this section are conducted on Calc (OSWorld) using MAGE with UI-TARS-1.5 as the backbone. Results for the backbone model and MAGE are reported as the mean over five independent runs, while all other variants are evaluated using a single run. The "Without RAP" setting uses the adapted CUA without a plan during evaluation and corresponds to the PO-only setting in Table 3. We analyze the sensitivity and robustness of Retrieval-Augmented Planning (RAP) by systematically varying its key components and configurations and examining their effects on task performance.

C.1 Manual Quality Sensitivity

We evaluate MAGE's robustness to documentation quality by introducing two forms of degradation: outdated and incomplete manuals. We replace the LibreOffice 7.1 manual (2021) with the 3.3 version

Table 7: Sensitivity to manual quality on Calc. Old uses the 2010 manual instead of the 2021 manual, while Partial uses a subset of the 2021 manual.

Manual	SR (%)
Full (2021)	**34.04**
Old (2010)	25.53
Partial (2021, 75%)	25.53
Partial (2021, 50%)	21.28
Partial (2021, 25%)	17.02
Without RAP	14.89

Table 8: Sensitivity of MAGE to retrieval quality on Calc under Top-5, lower-ranked, random, and without RAP.

Retrieval Method	SR (%)
Top-5 retrieval (Ours)	**34.04**
Random 5 sections	27.66
Low-ranked sections (Ranks 6–10)	14.89
Without RAP	14.89

Table 9: Comparison of Top-K retrieval

Top-K	SR (%)
Top-1	25.53
Top-3	27.66
Top-5 (Ours)	**34.04**
Top-10	25.53
Without RAP	14.89

Table 10: Human validation of generated tasks in terms of usefulness, feasibility, and visual verifiability.

Program	Usefulness	Feasibility	Visual Verifiability
Calc	98.0% (49/50)	80.0% (40/50)	100.0% (50/50)
Impress	86.0% (43/50)	84.0% (42/50)	100.0% (50/50)
Writer	96.0% (48/50)	88.0% (44/50)	100.0% (50/50)
Overall	**93.3% (140/150)**	**84.0% (126/150)**	**100.0% (150/150)**

(2010) to test age,[5] and randomly retain 75%, 50%, and 25% of the 2021 manual's sections to test completeness. All evaluations are conducted on Calc.

As Table 7 indicates, performance scales with manual quality. Using the older 2010 manual drops the success rate from 34.04% to 25.53%, and performance decreases as fewer sections of the 2021 manual are retained, reaching 17.02% at 25% retention. Importantly, all documentation-backed setups outperform the baseline lacking retrieval-augmented planning (RAP) (14.89%). This confirms that while MAGE optimally leverages comprehensive, up-to-date text, it still extracts actionable procedural guidance from suboptimal sources.

C.2 Retrieval Sensitivity

We analyze the sensitivity of MAGE to two retrieval factors: retrieval quality and the number of retrieved manual sections (Top-K). For retrieval quality, we compare the default Top-5 retrieval with lower-ranked sections (Ranks 6–10), five randomly selected sections, and a setting without retrieval-augmented planning (RAP). For Top-K, we vary the number of retrieved sections. All experiments are conducted on Calc under the same evaluation setting.

Retrieval quality. As shown in Table 8, performance drops to 27.66% with randomly selected sections and further decreases to 14.89% with lower-ranked sections (Ranks 6–10). This indicates that MAGE is sensitive to retrieval quality and that retrieving relevant manual sections has a substantial impact on its overall performance.

Effect of top-K. As shown in Table 9, Top-5 retrieval achieves the highest observed success rate (34.04%) among the evaluated settings. Increasing K further to 10 reduces the success rate to 25.53%. This suggests that retrieving few sections may provide insufficient procedural information, whereas retrieving many sections can introduce less relevant content that interferes with planning. Based on this analysis, we use Top-5 retrieval as the default configuration throughout our experiments.

D Human Validation of MAGE Components

We conduct a human validation study to assess the reliability of the LLM-generated supervision used in MAGE. Specifically, we examine three components: generated tasks, manual-guided plans, and trajectory evaluation.

[5]The LibreOffice manuals used in this experiment are available online: LibreOffice 7.1 Calc Guide and LibreOffice 3.3 Calc Guide.

Table 11: Human evaluation of retrieval-augmented plans.

Criterion	Pass Rate
Requirement coverage	82.98% (39/47)
Procedural correctness	72.34% (34/47)
Executability	76.60% (36/47)
Scope safety	93.62% (44/47)
Helpful for task execution	**70.21% (33/47)**

Table 12: Agreement between GPT-5.4 and the official OSWorld evaluator.

Program	Agreement
Calc	82.98% (39/47)
Impress	73.33% (33/45)
Writer	86.36% (19/22)
Overall	**79.82% (91/114)**

D.1 Human Validation of Generated Tasks

We manually inspect 150 generated tasks, including 50 tasks each from Calc, Impress, and Writer. Each task is evaluated according to three binary criteria:

1. **Usefulness:** whether the task corresponds to a meaningful operation that a real user may perform.
2. **Feasibility:** whether the instruction is sufficiently specified and executable in the given environment.
3. **Visual verifiability:** whether successful task completion can be determined from the resulting screenshots.

As shown in Table 10, 93.3% of the generated tasks are judged useful, and all tasks are visually verifiable. The feasibility rate is lower at 84.0%, with the remaining cases primarily involving ambiguous instructions or missing prerequisites, such as required files or objects. These results show that the generation process produces largely meaningful and visually verifiable tasks, while also motivating the use of filtering before episode generation to remove infeasible instances.

D.2 Human Validation of Generated Plans

We further evaluate the quality of manual-guided plans on OSWorld Calc using four criteria: **requirement coverage** (covering key task requirements), **procedural correctness** (following a valid procedure), **executability** (being actionable in the environment), and **scope safety** (avoiding unnecessary actions). Minor correctable errors are tolerated, while missing essential steps or task-blocking errors are considered failures.

As shown in Table 11, a plan is considered helpful if it satisfies requirement coverage and at least two of the remaining three criteria. Under this definition, 70.21% of the generated plans are helpful, indicating that most plans capture the task objective without introducing substantial out-of-scope actions.

D.3 External Validation of the VLM Evaluator

To validate the VLM-based evaluator, we measured its agreement with OSWorld's rule-based evaluation functions. Table 12 shows an overall agreement of 79.82% (91/114) across Calc, Impress, and Writer. Disagreements mainly occur in multi-condition tasks where modifications are not fully verifiable from final screenshots. These results indicate substantial agreement with the official evaluator, while highlighting limitations in visually ambiguous cases.

E Action Space of CUAs

UI-TARS-1.5. The action space of MAGE follows the action space of UI-TARS-1.5. Detailed descriptions of the action space are provided in Table 14.

EvoCUA. The action space of MAGE with the EvoCUA backbone follows the action space defined by EvoCUA. Detailed descriptions of the action space are provided in Table 13.

Table 13: Native action space of the EvoCUA agent.

Category	Action Primitive	Description	Required Arguments
Keyboard	`key`	Presses and releases the specified key(s).	`keys` (array)
	`key_down`	Presses and holds the specified key(s).	`keys` (array)
	`key_up`	Releases the specified key(s).	`keys` (array)
	`type`	Types the specified text.	`text` (string)
Mouse	`mouse_move`	Moves the cursor to the specified coordinates.	`coordinate` (x, y)
	`left_click`	Performs a left click.	`coordinate` (x, y)
	`right_click`	Performs a right click.	`coordinate` (x, y)
	`middle_click`	Performs a middle click.	`coordinate` (x, y)
	`double_click`	Performs a left double-click.	`coordinate` (x, y)
	`triple_click`	Performs a left triple-click.	`coordinate` (x, y)
	`left_click_drag`	Clicks and drags to the specified coordinates.	`coordinate` (x, y)
	`scroll / hscroll`	Performs vertical or horizontal scrolling.	`pixels` (number)
Control	`wait`	Pauses execution for the specified duration.	`time` (number)
	`terminate`	Terminates the task with the specified status.	`status` (`"success"`|`"failure"`)

Table 14: Action space of the UI-TARS-1.5 (Qin et al., 2025) agent.

Action	Definition
`click(x, y)`	Clicks at coordinates (x, y).
`left_double(x, y)`	Double-clicks at coordinates (x, y).
`right_single(x, y)`	Right-clicks at coordinates (x, y).
`drag(x1, y1, x2, y2)`	Drags from (x_1, y_1) to (x_2, y_2).
`hotkey(key)`	Presses the specified keyboard shortcut.
`type(content)`	Types the specified text content.
`scroll(x, y, direction)`	Scrolls at (x, y) in the specified `direction`$\in$ {`up`, `down`}.
`wait()`	Waits for a brief moment.
`finished()`	Marks the task as complete.

F Prompts

F.1 Manual-Derived Task Generation Prompt

We use the following prompt to generate manual-derived task samples from each manual section. The same prompt template is used across applications, while the input program name, manual section text, and section images are replaced for each section. Placeholders enclosed in braces denote instance-specific inputs.

Manual-Derived Task Generation Prompt

```
You are an expert at generating GUI-based practice tasks for AI
    agents.

You will be given:
```

```
1. Program name
2. One section of manual text, which may span multiple pages
3. One or more images from the manual section

Your task is to generate executable GUI practice tasks based
    strictly on the given manual section.

Each task must be output as an instruction pair with:
- setup instruction, if necessary
- main instruction

Each instruction must contain:
- task: a short sentence describing the goal
- steps: a list of concrete GUI actions

Rules:

1. Generate up to two task samples, or none if the section is
    irrelevant.

2. The task must produce a visible result on the screen.
   Do not generate tasks such as:
   - opening menus only
   - checking whether a feature exists
   - saving a file
   - exploring the UI
   The result must be visually observable, such as formatted text,
    a created chart, a changed slide object, or hidden columns.

3. The starting screen is always a freshly opened blank/default
    screen.
   No files, tables, documents, slides, conversations, or other pre-
    existing content are assumed.

4. The GUI agent must not be asked to make subjective or free
    choices.
   Do not generate steps such as:
   - choose any color
   - select a preferred style
   - pick an option you like
   - apply a random format
   All actions must be deterministic and explicitly specified.

5. Output format:

[
  {
    "setup_task": "...",
    "setup_steps": [...],
    "main_task": "...",
    "main_steps": [...],
    "id": 0
  }
]

6. Setup instruction:
- Setup is only for creating or preparing data or objects required
    by the main task.
- If the main task needs any pre-existing value, text, table, slide
     object, or conversation content, it must be created in
    setup_steps.
```

```
- Main steps must never include data creation actions such as
    typing initial values, creating tables, or preparing sample
    objects.
- If the main task can be done from a blank screen without
    preparation, leave setup_task as an empty string and setup_steps
     as an empty list.

7. Main instruction:
- Main steps must only apply the manual feature to the prepared
    data or objects.
- Main steps must not include:
  - entering initial numbers or text
  - creating tables
  - preparing sample data
  - creating initial objects
- Main task must be a concrete exercise reconstructed from the
    manual section.

8. If sample data or objects are required in setup:
   - Do not rely only on images.
   - Describe exactly how to create the required data or objects in
     text steps.

9. Steps must:
   - be atomic GUI actions, such as click, type, select, drag, or
    press keys
   - be ordered
   - not include explanations
   - not mention the manual or images

10. Do not invent features not mentioned in the given manual
    section.

11. Section-based multimodal input handling:
   - The text is section-level, not single-page.
   - Multiple images may be provided for the same section.
   - Use all provided images as supporting context when relevant.
   - If text and images conflict, prioritize the manual text.
   - Do not assume hidden UI details not visible in the text or
    images.

12. Output must be only valid JSON.
   - No markdown
   - No explanation
   - No extra text

Input:
Program name:
{program_name}

Manual section text:
{manual_section_text}

Manual section images:
{manual_section_images}
```

F.2 MANUAL-GUIDED PLAN GENERATION PROMPT

This prompt is used to generate a task-specific plan during Manual-Guided Planning. It takes the task instruction, the initial screenshot, and retrieved manual sections as input, and outputs a concise step-by-step plan that guides the CUA during action generation.

Manual-Guided Plan Generation Prompt

```
You are a planner for a GUI agent operating on real screenshots.

You will be given:
1) A task instruction
2) The initial screenshot of the environment
3) Potentially relevant manual snippets retrieved for this task

Your goal is to generate a concise, step-by-step plan that the
    agent can follow to complete the task.

Requirements:
- Focus only on actions that are directly useful for completing the
     task.
- Each step must correspond to a realistic GUI interaction, such as
     click, type, scroll, or select.
- Ground each step in visible UI elements from the initial
    screenshot, such as button labels, menus, input fields, icons,
    or relative positions.
- Prefer explicit interaction targets, such as clicking the Insert
    menu at the top toolbar, instead of vague descriptions.
- If multiple paths are possible, choose the most straightforward
    and reliable one.
- Avoid assumptions about hidden UI or unavailable elements.
- If the UI element is not clearly visible, describe how to locate
    it, such as via menu navigation or scrolling.
- Do not include unnecessary explanations, reasoning, or background
     knowledge.
- Do not repeat the task instruction.

Robustness:
- If an action might fail, suggest a simple alternative interaction,
     such as using a button instead of typing.
- Ensure the plan is executable even with minor UI variations.

Output format:
- Output only one section titled "Relevant plan".
- Use short, ordered, numbered steps.
- Each step should describe exactly one action.

Input:
Task instruction:
{task_instruction}

Initial screenshot:
{initial_screenshot}

Retrieved manual sections:
{retrieved_manual_sections}
```

F.3 Trajectory Evaluation Prompt

The evaluator outputs both a binary success label and a continuous progress score. The progress score is used only for consistency checking and is not used as a training reward. For policy optimization, we assign rewards based on the binary success label.

Trajectory Evaluation Prompt

```
You are an evaluation agent.

You will be provided with:
1. The Task objective
2. The last 5 images of the task trajectory.

Your tasks consist of two parts:

PART 1: Task Success Evaluation
- Determine whether the Task objective has been successfully
    completed.
- Evaluate this by referring to:
  - the visual evidence in the last 5 step images
- Do not rely only on the final image; consider transitions and
    changes across all five images.
- Carefully analyze visual information and logical progression step-
    by-step.

PART 2: Task Progress Evaluation
- Evaluate how much the Task has progressed toward completion.
- Use:
  - the visual evidence across the last 5 step images
- Do NOT rely only on the final image.
- Compare observed behavior strictly against the Task requirements.
- The score should reflect how much of the Task has been completed
    so far, not just whether it succeeded or failed.

Scoring Guidelines:
- Assign a score between 0.0 and 1.0 in increments of 0.01.
- 0.00 = no meaningful progress toward the Task goal
- 1.00 = Task fully completed
- Intermediate values should represent proportional progress.
- Consider:
  - how many required steps have been completed
  - how correct the completed steps are
  - whether errors block further progress

Output Rules:
- Output MUST be valid JSON.
- Do NOT include any text outside the JSON object.
- Always output the score as a JSON number, not a string.

Output format:
{
  "result_thought": "Explain, step-by-step, how you analyzed the
    last 5 images and why you concluded that the Task succeeded or
    failed.",
  "result": "SUCCESS" or "FAIL",
  "score_thought": "Explain, step-by-step, how you analyzed the
    last 5 images and why you assigned this progress score.",
  "score": 0.00
}
```