EFFICIENT AGENT TRAINING FOR COMPUTER USE

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling up high-quality trajectory data has long been a critical bottleneck for developing human-like computer use agents. We introduce PC Agent-E, an efficient agent training framework that significantly reduces reliance on large-scale human demonstrations. Starting with just **312** human-annotated computer use trajectories, we further augment them by synthesizing diverse alternative action decisions with Claude 3.7 Sonnet. Trained on these enriched trajectories, our PC Agent-E model achieved a remarkable **141%** relative improvement, and even surpassed the Claude 3.7 Sonnet by **10%** on WindowsAgentArena-V2, an improved benchmark we also released. By integrating robust human computer use skills with automated AI data synthesis capabilities, our method not only brought substantial improvements over training on human trajectories alone, but also significantly surpassed direct distillation from Claude 3.7 Sonnet.

1 INTRODUCTION

Developing autonomous agents that can operate computers as humans do Anthropic (2024); He et al. (2024); OpenAI (2025) has long been a landmark pursuit in Artificial Intelligence (AI). Such computer use agents, powered by Vision-Language Models (VLMs), perceive screenshots to interact directly with Graphical User Interfaces (GUIs) — clicking buttons, navigating menus, and entering text. This allows them to automate a wide range of digital tasks, ranging from routine paperwork and online shopping to complex content creation, promising a significant reduction in manual human workload.

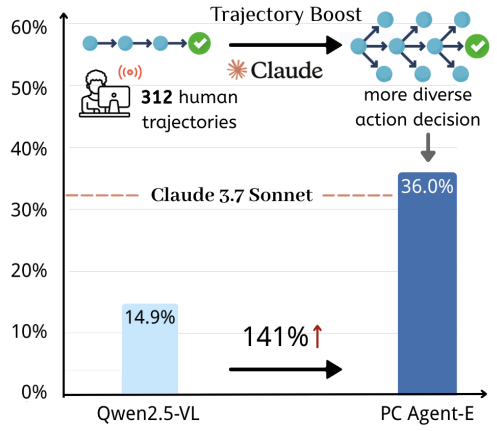


Figure 1: PC Agent-E achieves state-of-the-art open-source performance in Windows computer use with just 312 augmented trajectories.

However, current models still fall significantly short of human performance Xie et al. (2024); Bonatti et al. (2024). This capability gap is even more pronounced within the open-source community, which lacks any solution competitive with leading proprietary systems like Claude 3.7 Sonnet Anthropic (2025a). Instilling these advanced computer use capabilities in open-source models remains an unsolved problem. A key factor contributing to these deficiencies is the extreme scarcity of high-quality computer use trajectory data Ou et al. (2024); Xu et al. (2025a).

In this work, we explore **efficient agent training** for computer use, enabling open-source models to even exceed the performance of proprietary counterparts with minimal human annotation. Inspired by recent findings Huang et al. (2024); Muennighoff et al. (2025); Ye et al. (2025) that synthesizing high-quality data using advanced reasoning models like Deepseek-R1 Guo et al. (2025) can efficiently enhance LLM reasoning, we extend the similar idea to the field of computer use agents.

We propose **PC Agent-E**, an efficient agent training framework that integrates human expertise with AI automation. Starting from a small set of real-world human computer use trajectories, we leverage

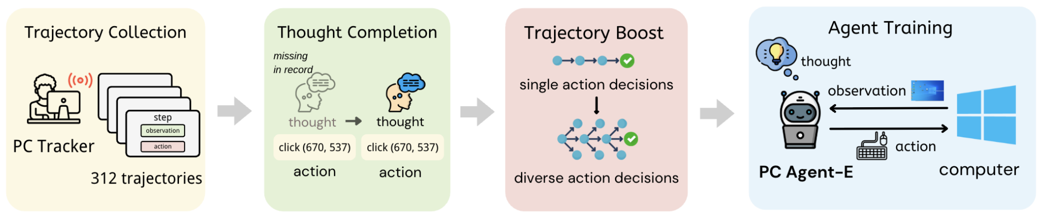


Figure 2: Overview of our framework, consisting of four key components: (1) Trajectory Collection, gathering a small set of human trajectories by recording user actions and state observations at each step; (2) Thought Completion, reconstructing the implicit thought process missing in raw human trajectories; and (3) Trajectory Boost, diversifying action decisions to further augment trajectories (4) Agent Training, developing a strong computer use agent with remarkable data efficiency.

a frontier agent model to diversify action decisions (action with thought in ReAct Yao et al. (2023) paradigm), further enhancing the data quality. Training on these augmented trajectories, our agent demonstrates strong computer use capabilities with remarkable data efficiency.

We begin by collecting **312** human computer use trajectories with PC Tracker He et al. (2024), a tool for gathering human-computer interaction data, with only two humans annotating one day. These trajectories include task descriptions, screenshots, and human keyboard/mouse actions. We subsequently reconstruct the implicit thought process behind human actions, obtaining comprehensive human trajectories with thoughts. The successful completion of these tasks is inherently assured by human proficiency, obviating the need for additional verification.

While these human trajectories already serve as valuable agent training data, we further augment them through **Trajectory Boost**, a data synthesis method we developed to enrich each trajectory step with diverse alternative action decisions. The key insight is that computer use tasks can be completed through multiple valid pathways, meaning each step has various reasonable action alternatives supported by rational thought. To capture this diversity, we use a strong agent model to synthesize other possible action decisions. Specifically, recognizing that each human trajectory step captures an ***environment snapshot*** essential for computer use agents to make decisions, we provide these snapshots to Claude 3.7 Sonnet Anthropic (2025a) and sample multiple possible action decisions, thereby significantly enriching and diversifying the trajectory data.

Experimental results demonstrate that with only a small set of human-annotated trajectories, our method can boost the performance of an open-source model to that of frontier models. Trained with only 312 trajectories augmented by Claude 3.7 Sonnet, our PC Agent-E model achieves an impressive **141%** relative improvement over the base model Qwen2.5-VL-72B and even outperforms the teacher model Claude 3.7 Sonnet by 10% on **WindowsAgentArena-V2**, a benchmark we improved from WindowsAgentArena Bonatti et al. (2024). Furthermore, PC Agent-E generalizes well to different operating systems on OSWorld Xie et al. (2024). Our ablation study further demonstrates that our method for utilizing human demonstrations is not only superior to relying solely on the human trajectories but is also significantly more effective and efficient than directly distilling from the teacher model.

In summary, our key contributions are threefold:

1. We propose **Trajectory Boost**, a simple data synthesis method that unlocks remarkable data efficiency for training computer use agents. By augmenting human trajectories with diverse action decisions from a frontier model, our method demonstrates significantly greater effectiveness and efficiency than both using human data alone and direct distillation.
2. We release **WindowsAgentArena-V2**, an improved benchmark rectifying evaluation dependence, infeasible hacking, and other limitations in the original WindowsAgentArena benchmark, ensuring more robust and fair evaluations of computer use.
3. We developed **PC Agent-E**, an open-source computer use agent that achieves performance comparable to leading proprietary models. Trained with just 312 augmented trajectories, our model successfully outperforms the strong teacher model, Claude 3.7 Sonnet, showing exceptional data efficiency.

2 Related Work

2.1 Computer Use Agent

With the advancement of VLMs Bai et al. (2025); Deitke et al. (2024), the way computer use agents interact with computers has gradually shifted from relying on textual representations such as accessibility trees Agashe et al. (2024); Wu et al. (2024a) to directly using screenshots Qin et al. (2025); Xu et al. (2025b); He et al. (2024). Existing computer use agents can be categorized based on how much human prior is built into their design: One is **modular agent workflows** Agashe et al. (2024); Wu et al. (2024a), which defines specialized modules and prompts multi-agents to collaborate. The other is **native agent models** Anthropic (2024); Qin et al. (2025); OpenAI (2025), which depends on a single model to take action step by step based on its history and current state.

While modular agent workflows can reduce task complexity, their heavy reliance on human priors hinders adaptation to new domains and limits end-to-end optimization Saltzer et al. (1984); Pan et al. (2024). With the continuous enhancement of model capabilities, native agent models have emerged as the dominant paradigm. This approach offers flexibility, generalizability, and sustainable gains via supervised fine-tuning (SFT) Xu et al. (2025b) or reinforcement learning (RL) OpenAI (2025). Our work explores the efficient agent training methods for native agent models through SFT.

2.2 Data Synthesis

As Large Language Models (LLMs) grow ever more powerful, it has become a common practice to use them to synthesize data. Distillation methods Taori et al. (2023); Gunasekar et al. (2023); Xu et al. (2023) leverage state-of-the-art (SOTA) models to generate large-scale data for training weaker models. On the other hand, self-improvement methods enable a model to bootstrap and refine its own training data Wang et al. (2023).

In the domain of computer use agents, data synthesis can be broadly categorized into three aspects: (1) large-scale datasets that build foundational GUI understanding, with tasks like screenshot captioning or question–answering Liu et al. (2024); Qin et al. (2025) (2) single-step visual grounding, where mouse click tasks are synthesized from specific locations on the GUI Gou et al. (2025); Wu et al. (2024b) (3) multi-step trajectory, in which recent research has explored leveraging web tutorials to guide trajectory generation Ou et al. (2024); Xu et al. (2025a) or reverse-synthesizes tasks from the agents' own exploration records Sun et al. (2025); Murty et al. (2024). Our work differs from prior work by synthesizing high-quality multi-step trajectories based on real-world human demonstrations and emphasizing data efficiency.

3 Method

3.1 Overview

We propose PC Agent-E, an efficient agent training framework for computer use that integrates human expertise with AI automation, as illustrated in Figure 2. Our method generates high-quality trajectory data by combining authentic human-computer interactions with diverse action decisions, offering advantages in both realism and diversity.

1. First, we gathered a small set of 312 task trajectories from human annotators, recording both the screen state observation and the human action at each step, and then filtered the data to remove erroneous steps and trajectories. (§3.2)
2. Subsequently, we reconstructed the latent human thought process before each action decision, based on the corresponding screen state observation and history step context. (§3.3)
3. Then, using human trajectories as *environment snapshots*, we employ Claude 3.7 Sonnet to synthesize diverse alternative action decisions with the Trajectory Boost method. (§3.4)
4. Finally, we develop PC Agent-E, our SOTA native agent model for Windows computer use, trained on our augmented trajectories with a simple end-to-end scaffold. (§3.5)

3.2 Trajectory Collection

We collected human computer use trajectories with PC Tracker He et al. (2024), a tool that records the screen state observation and the human keyboard/mouse action at each step for a given task. The recorded actions are structured in a unified action space $\mathcal{A}$, as shown in Table 1. For task generation, we first manually compose a small seed set across multiple software applications and then enlarge it with LLMs. The resulting tasks were distributed to human annotators, who completed the tasks on their own Windows computers with PC Tracker recording trajectories automatically. After finishing a task, annotators could either discard unsatisfactory trajectories or modify the task descriptions based on their actual execution, thereby ensuring the correctness and completeness of the collected trajectories. We then applied a set of rule-based filters to remove entire trajectories or individual steps that exhibited errors or other undesirable behaviors.

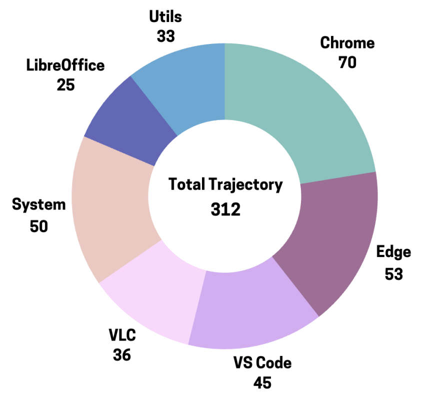


Figure 3: Distribution of the 312 task trajectories across different applications.

We employed a rigorous data decontamination procedure on these collected trajectories. Each task description was compared against the tasks in our main evaluation benchmark (§4) using n-gram overlap and semantic similarity metrics. Trajectories with task descriptions exhibiting excessive similarity to any test task were removed from the dataset.

This procedure finally yielded 312 real-world human computer use trajectories, with distribution across applications shown in Figure 3. The whole annotation process was completed by two annotators within a single day, with an average of roughly 3 minutes per trajectory. Given humans' proficiency in computer use, the mechanisms for annotators to discard trajectories or revise task descriptions after execution, and our data filtering process, no additional verification was required to ensure trajectory correctness.

3.3 Thought Completion

We first reconstruct the implicit thought process behind human actions using an iterative approach. Specifically, for each action in the raw trajectory, we provide Claude 3.7 Sonnet with: task description, historical actions with their previously reconstructed thought processes, the current action, and the corresponding screenshot. Based on this information, the model generates the implicit thought process behind action. As shown in Figure 4, the recorded raw human trajectory was converted to a human trajectory with thoughts, where the reconstructed thought process is added to each step. See our prompt in Appendix D.1.

Action	Description
click (x, y)	clicks at coordinates.
right click (x, y)	right-click at coordinates.
double click (x, y)	double-click at coordinates.
drag from (x1, y1) to (x2, y2)	drag the mouse.
scroll (x)	scrolls the screen with offset x.
press key: enter	presses the Enter key.
hotkey (ctrl, c)	performs the Ctrl+C hotkey.
type text: hello	type text "hello".
wait	pauses for some time.
finish	the task is finished.
fail	the task is failed.

Table 1: Unified action space $\mathcal{A}$.

3.4 Trajectory Boost

After thought completion, we obtain comprehensive human trajectories with explicit thought processes. While these trajectories already serve as valuable agent training samples, we further augment them through a simple but effective approach called **Trajectory Boost**, which synthesizes diverse alternative action decisions for each step of the trajectory.

The motivation behind Trajectory Boost is that computer use tasks inherently allow for multiple valid solution pathways. Consequently, at any given step, several reasonable actions supported by rational thought processes may exist, extending beyond the single solution adopted by human annotators. To capture this inherent diversity, we utilize a frontier computer use agent model, Claude 3.7 Sonnet, to generate single-step alternative action decisions. Its long-horizon planning capabilities, advanced

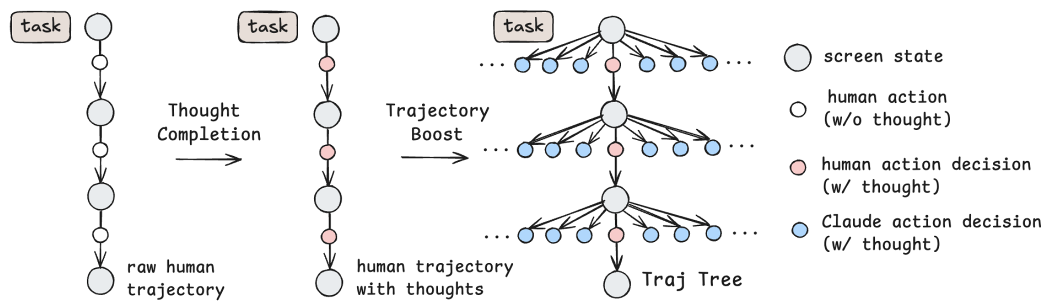
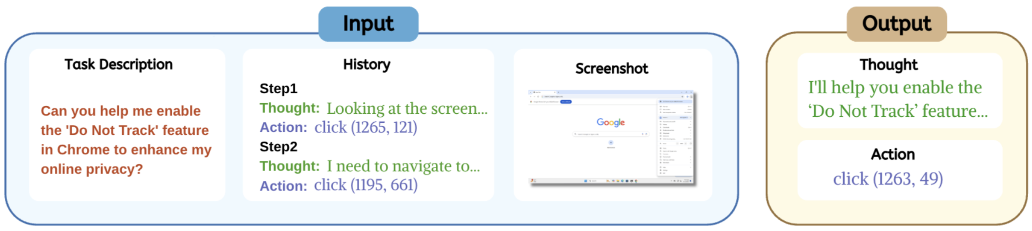

task
Thought Completion
Trajectory Boost
raw human trajectory
human trajectory with thoughts
Traj Tree
screen state
human action (w/o thought)
human action decision (w/ thought)
Claude action decision (w/ thought)

Figure 4: Visualization of our Trajectory Boost method. (Left) Raw human trajectory recorded by PC Tracker. (Center) Human trajectory with reconstructed thoughts after Thought Completion, where the red node indicates human action decisions. (Right) The final *Traj Tree*, where the blue node indicates augmented diverse action decisions synthesized by Claude 3.7 Sonnet.

reasoning patterns, and broad knowledge of computer use enable it to generate thought processes and actions that are highly informative and valuable, thereby substantially enhancing the richness and diversity of our trajectory data.

Specifically, we recognize that each step in a human trajectory captures an *environment snapshot* of the computer, providing the necessary information for both humans and agents to make decisions. For step k on a human trajectory with observation o_k, thought process t_k, action a_k and task description T, the *environment snapshot* is $< T, o_k, h_k >$, where the history context $h_k = (t_1, a_1, t_2, a_2, \ldots, t_{k-1}, a_{t-1})$ is constructed with previous human steps. We input this *environment snapshot* to Claude 3.7 Sonnet instantiate in the PC Agent-E scaffold (§3.5), and sample multiple single-step action decisions (t'_k, a'_k) from it. Prompts used are shown in Appendix D.2. In practice, we sample 9 action decisions in parallel. This produces a ***Traj Tree***, as shown in Figure 4, with human trajectory forming the main trunk and the augmented action decisions branching off as leaf nodes. These sampled action decisions from Claude 3.7 Sonnet are not executed in real computer environments, but serve as important augmented data for later agent training.

3.5 Agent Training



Can you help me enable the 'Do Not Track' feature in Chrome to enhance my online privacy?

Step1
Thought: Looking at the screen...
Action: click (1265, 121)
Step2
Thought: I need to navigate to...
Action: click (1195, 661)

I'll help you enable the 'Do Not Track' feature...

click (1263, 49)

Figure 5: A training example that also demonstrates the inference process of the PC Agent-E scaffold.

PC Agent-E adopts a deliberately simple end-to-end scaffold, as our primary focus is on validating the effectiveness of our agent training methodology rather than optimizing performance through complex workflow design or elaborate prompt engineering. At inference, PC Agent-E takes ⟨screenshot, task description, history⟩ as input and outputs a ⟨thought, action⟩ decision in the ReAct Yao et al. (2023) paradigm, as shown in Figure 5. The action space is the same as $\mathcal{A}$ in Table 1, and every action is executed via the `PyAutoGUI` library. The history is a textual log of previous thoughts and actions. To maintain simplicity in both training and inference, past screenshots are excluded, although we believe that adding this image history would be beneficial for improving model performance. The prompt used for the scaffold is shown in Appendix D.3.

For training, we transform each action node from our *Traj Tree* into an individual training sample. The training sample structure and the inference-time scaffold of the agent share a direct correspondence,

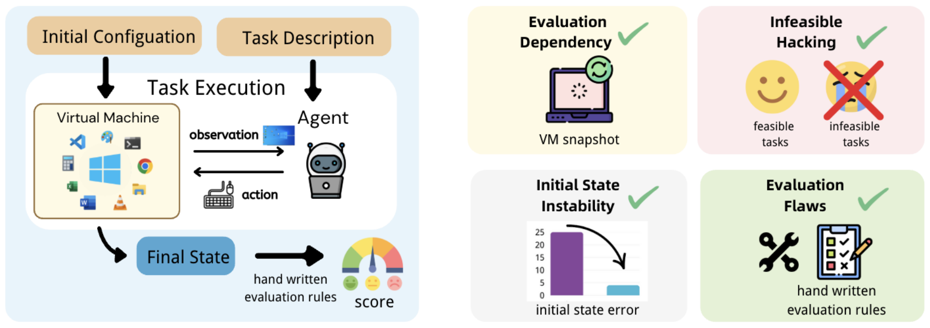


Figure 6: (Left) Overview of the WindowsAgentArena benchmark. (Right) Our main modifications to the updated WindowsAgentArena-V2 benchmark.

as illustrated in Figure 5. For both human-demonstrated and model-synthesized action nodes, the history in the training sample includes only prior human actions on the main trunk of the *Traj Tree*. This is consistent with the historical context available to both humans and the model when making the corresponding action decision. We finally obtained 27K training samples from 312 augmented trajectories, each following a consistent input-output structure at inference time.

4 WindowsAgentArena-V2

We initially performed our evaluation on WindowsAgentArena Bonatti et al. (2024), a benchmark designed to assess computer use ability in realistic Windows OS environments through diverse tasks across multiple applications. It provides automatic initial configuration of virtual machine (VM) state and hand-written evaluation rules (see Figure 6). However, we identified several limitations during our assessment. To ensure evaluation reliability, we developed **WindowsAgentArena-V2**, an updated benchmark comprising 141 tasks across 11 widely-used Windows applications, all derived from the original WindowsAgentArena but with improvements detailed below.

Addressing the evaluation dependency issue. The original benchmark lacked VM state reset between task evaluations, allowing changes from previous tasks to potentially affect subsequent ones. We implemented VM snapshot restoration before each evaluation, ensuring consistent starting states, preventing inter-task interference, and aligning with the i.i.d. (independent and identically distributed) assumption. We also installed some essential software missing from the original VM snapshot but required for proper evaluation.

Preventing infeasible hacking. Current computer use benchmarks such as WindowsAgentArena and OSWorld often include infeasible tasks, which are inherently impossible to complete due to issues such as deprecated system features or user-generated hallucinated commands Xie et al. (2024). The evaluation metric for these tasks is simply considering a task successful if the action `FAIL` is output at any point during execution. However, we found such evaluation methods particularly easy to hack: an agent can trivially achieve a perfect score on infeasible tasks by always outputting `FAIL`, without demonstrating any meaningful computer use capabilities. In contrast, completing a feasible task typically requires the agent to execute actions step-by-step to actually fulfill the task objective, posing a significantly different level of difficulty.

We refer to this phenomenon as ***infeasible hacking***, a vulnerability confirmed by our subsequent experiments (§5.6), in which a weaker model achieved markedly higher scores on infeasible tasks. Since agents receive identical scores for feasible and infeasible tasks, their coexistence undermines benchmark fairness. Additionally, given that current computer use agents' capabilities are far from optimal, we argue it is presently more valuable to focus on enhancing agents' performance on feasible tasks. Therefore, as a temporary solution in WindowsAgentArena-V2, we removed all infeasible tasks to prevent *infeasible hacking*.

Guaranteeing VM initial state stability. We found that the state of VM after task initial configuration often exhibited errors like unstable network connections, software launch failures, or system lags. To address this, we designed a validation framework combining rule-based and LLM-based evaluations to verify the initial state, with a re-test mechanism allowing up to three restart attempts for faulty initializations. This approach reduced the initialization failure rate from 10%–30% (depending on hardware) to below 5%.

Fixing evaluation flaws. We discovered that some evaluation functions contained bugs or lacked robustness. For instance, in the task "`clearing YouTube history to facilitate finding other histories`", the evaluation erroneously awarded full scores to agents that deleted the entire browsing history, clearly contradicting user intent. We identified and corrected several evaluation errors and relied on human evaluators for a few complex tasks to improve assessment reliability.

5 Experiment

In this section, we conduct extensive experiments to evaluate PC Agent-E and validate our Trajectory Boost method. Our experiments are designed to answer the following key questions:

1. How does PC Agent-E perform against SOTA methods on computer use tasks? (§5.2)
2. How does Trajectory Boost's data scaling surpass using human demonstrations alone? (§5.3)
3. How does Trajectory Boost differ from and outperform Direct Distillation? (§5.4)
4. How does test-time scaling affect the performance of PC Agent-E? (§5.5)
5. How well does PC Agent-E generalize to unseen environments? (§5.6)

5.1 Setup

Benchmarks We use WindowsAgentArena-V2 (§4) for the main evaluation, as our training data were collected on the Windows system. We also include results on the original WindowsAgentArena Bonatti et al. (2024) in Appendix B for completeness. To test generalization across operating systems, we report results on OSWorld Xie et al. (2024), another computer use benchmark for Linux systems.

Model Baseline We compare PC Agent-E with several SOTA models. These include the leading proprietary models Claude 3.7 Sonnet Anthropic (2025a) and Claude 3.7 Sonnet with extended thinking Anthropic (2025b), as well as open-source models UI-TARS Qin et al. (2025), UI-TARS-1.5 Team (2025), and Qwen2.5-VL-72B Bai et al. (2025). We also compared with the popular GPT-4o OpenAI (2024) model.

Method Baseline We compare our Trajectory Boost method with two alternative training approaches. The first is standard behavior cloning on the 312 human trajectories after thought completion. The second is direct distillation from Claude. We sample 10 end-to-end trajectories from Claude 3.7 Sonnet for each of the 312 tasks. The resulting 3,120 trajectories are then used for training with the identical procedure as PC Agent-E, matching the trajectory number of our method for a fair comparison.

Settings All experiments and models utilized a screenshot-only observation setting with a uniform screen resolution of 1280×720. For the UI-TARS model series, we adopted their native framework, which supports image history and code block actions. All other models, including Claude and Qwen, were evaluated using our simple PC Agent-E scaffold. The default maximum number of steps was set to 30, and we also investigated the impact of varying this step limit on model performance.

Training We train our PC Agent-E model based on the Qwen2.5-VL-72B Bai et al. (2025) backbone with 27k data mentioned in §3.5. Experiments on a smaller model Qwen2.5-VL-7B are also included in Appendix C. We set the image resolution to 1280×720 and context length to 8,192 tokens. Further training details can be found in Appendix A.

5.2 Main Results

As shown in Table 2, PC Agent-E achieves a remarkable **141%** relative improvement over the base model Qwen2.5-VL-72B on WindowsAgentArena-V2, even surpassing the strong teacher model Claude 3.7 Sonnet by **10%**, establishing itself as the SOTA open-source model for Windows computer use. Notably, Claude 3.7 Sonnet used to synthesize our training data did not have the thinking mode enabled, but PC Agent-E achieves performance comparable to the stronger Claude 3.7 Sonnet with extended thinking.

Models	**Libreoffice**	**Chrome**	**Edge**	**System**	**VS Code**	**VLC**	**Utils**	**Total**
Number of Tasks	42	17	13	24	19	14	12	141
GPT-4o	0.0	5.9	0.0	8.3	0.0	0.0	0.0	2.1
Qwen2.5-VL-72B	0.0	34.7	15.4	20.8	26.3	7.6	16.7	14.9
UI-TARS-1.5-7B	**7.1**	34.7	23.1	45.8	21.1	7.6	16.7	21.3
UI-TARS-72B-DPO	0.0	40.6	38.5	58.3	36.8	7.6	25.0	26.2
Claude 3.7 Sonnet	2.4	46.5	**61.5**	54.2	52.6	29.0	16.7	32.6
Claude 3.7 Sonnet (thinking)	2.4	64.1	46.2	**66.7**	52.6	21.9	25.0	35.4
PC Agent-E (Ours)	4.8	**64.1**	46.2	50.0	**57.9**	**35.7**	**33.3**	**36.0**

Table 2: Results of success rate (%) for different models on WindowsAgentArena-V2.

Analysis To gain deeper insight into the specific capabilities enhanced through our training, we conducted a qualitative analysis by examining 50 trajectories that Qwen2.5-VL-72B failed but PC Agent-E succeeded, as well as trajectories where both models failed. We categorized the failure patterns into three types: (1) ***Knowledge***: models may lack specific computer use knowledge. For instance, a model might not know how to enable a particular feature in VLC (a media player software). (2) ***Planning***: models may make incorrect planning, such as failing to recognize and recover from previous erroneous actions. (3) ***Grounding***: models may execute actions that are inconsistent with their plan, primarily manifested as mouse-clicking errors. We found that our improvements primarily stem from enhanced *planning* capabilities. After training, PC Agent-E produces noticeably longer thought processes and demonstrates improved reasoning capabilities in verification and self-correction. We did not observe significant improvements in *knowledge* or *grounding* capabilities.

5.3 Data Scaling over Human Demonstrations

To validate the effectiveness of our Trajectory Boost method, we investigate the relationship between the scale of synthesized data and model performance. We define data **scaling factor**, s, as the ratio of the total action number used for training to the action number in the original human trajectory. For the model trained exclusively on the human demonstrations, the scaling factor is $s = 1$. Our final model, PC Agent-E, was trained using 9 synthesized actions and 1 original human action per step, corresponding to the scaling factor $s = 9 + 1 = 10$.

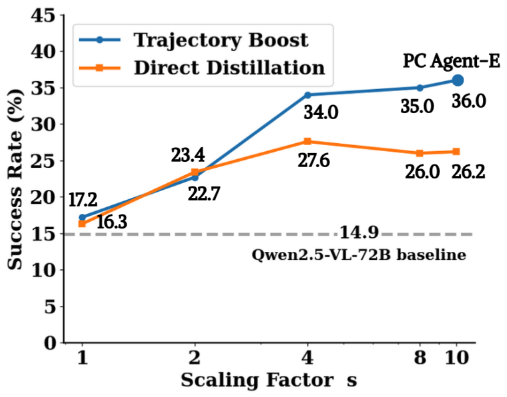


Figure 7: Performance of Trajectory Boost and Direct Distillation method with different data scaling factor s on WindowsAgentArena-V2.

As the blue line shown in Figure 7, our results reveal that model performance with the Trajectory Boost method scales significantly with the scaling factor. Compared to training on human trajectories alone, which yields a limited gain (improved from 14.9 to 17.2), PC Agent-E achieves substantially greater performance gains (improved from 14.9 to 36.0). This improvement is primarily driven by the diverse action decisions synthesized from the frontier model with thought processes. This supplements the single human-annotated solution and instills the frontier model's advanced planning capabilities into our agent, thereby yielding performance that far exceeds training on human trajectories alone.

5.4 Trajectory Boost vs. Direct Distillation

To demonstrate that our method is more than a simple form of distillation, we compare Trajectory Boost against a direct distillation baseline. For this baseline, we directly sample trajectories from our teacher model, Claude 3.7 Sonnet, end-to-end.

Superior Performance As shown in Figure 7, our method significantly outperforms the direct distillation baseline at most of the training data scales (blue line versus orange line). We attribute this to the high quality of our synthesized data. Our method uses human trajectories as a reliable foundation and leverages the frontier model to perform single-step augmentation. This avoids the error accumulation that can occur in end-to-end trajectory distillation.

Exceptional Efficiency Another significant advantage of our method is time efficiency. The distillation baseline requires deploying Claude in virtual machines and collecting trajectories through online interaction, which makes it resource-intensive and time-consuming. In contrast, our Trajectory Boost method performs offline data synthesis without interacting with the real environment, enabling natural parallelization. Specifically, to collect an equivalent amount (3120 trajectories) of data, the distillation baseline took about **900 hours**, while Trajectory Boost required only **3 hours** under the same hardware conditions — a drastic **300-fold speedup**.

5.5 Test Time Scaling

We also investigate how the performance of PC Agent-E varies with test time scaling, a topic that has received increasing attention in the research community Wu et al. (2025); Snell et al. (2024). We evaluated the model's performance with different numbers of max steps allowed during task completion. As shown in Figure 8, the difference in performance between the two models increases over time as the agent continues to interact with the computer. With improved *planning* capabilities, PC Agent-E benefits from "action scaling"—the ability to iteratively take more actions to respond to environmental changes, explore for solutions, correct errors, and ultimately solve the task. After training, PC Agent-E can utilize more time and computational resources to achieve better performance.

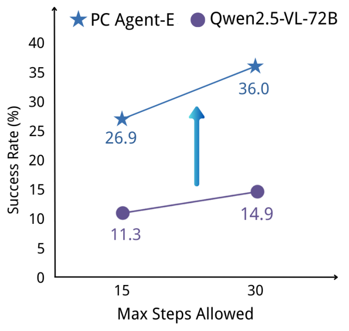


Figure 8: Test time scaling on WindowsAgentArena-V2.

5.6 Cross-Platform Evaluation

We further evaluated our model on OSWorld to assess cross-platform generalization capabilities. As shown in Table 3, despite being trained exclusively on Windows data, PC Agent-E achieves a 34% relative improvement in Linux systems as well. These results validate the generalizability of our method.

Models	Feasible	Infeasible	Total
Number of Tasks	339	30	369
Qwen2.5-VL-72B	4.4	**86.7**	11.1
PC Agent-E (Ours)	**10.9**	63.3	**14.9**

Table 3: Success rate (%) on OSWorld (30-step).

We also identified an interesting phenomenon in this experiment, which we designate as *infeasible hacking* in §4: the weaker Qwen2.5-VL-72B model paradoxically achieved markedly better performance on infeasible tasks. This observation suggests that current infeasible task evaluations do not accurately reflect computer use agents' capabilities. Future research may design better criteria for infeasible tasks, such as checking agents' rationale when declaring tasks impossible.

6 Conclusion

In this work, we introduced PC Agent-E, an efficient agent training framework for computer use. With just 312 augmented trajectories, PC Agent-E achieved a 141% improvement over the base model and outperformed the strong teacher model Claude 3.7 Sonnet. Our findings suggest that complex computer use capabilities can be elicited by a remarkably small set of high-quality trajectories.

7 Reproducibility Statement

To ensure the reproducibility of our work, we include relevant source code in the supplementary materials. This includes code and scripts for agent training, data processing, and the evaluation on WindowsAgentArena-V2. Furthermore, we provide detailed descriptions of our method within the main paper. Specifically, the data collection process is detailed in §3.2, the implementation of our Trajectory Boost method is described in §3.4, and details regarding the training and evaluation can be found in §5.1.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

Anthropic. Introducing computer use, 2024. URL `https://www.anthropic.com/news/3-5-models-and-computer-use`.

Anthropic. Claude 3.7 sonnet, 2025a. URL `https://www.anthropic.com/news/claude-3-7-sonnet`.

Anthropic. Claude's extended thinking, 2025b. URL `https://www.anthropic.com/news/visible-extended-thinking`.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025. URL `https://arxiv.org/abs/2502.13923`.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale, 2024. URL `https://arxiv.org/abs/2409.08264`.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv preprint arXiv:2409.17146*, 2024.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents, 2025. URL `https://arxiv.org/abs/2410.05243`.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need, 2023. URL `https://arxiv.org/abs/2306.11644`.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yanheng He, Jiahe Jin, Shijie Xia, Jiadi Su, Runze Fan, Haoyang Zou, Xiangkun Hu, and Pengfei Liu. Pc agent: While you sleep, ai works – a cognitive journey into digital world, 2024. URL `https://arxiv.org/abs/2412.17589`.

Zhen Huang, Haoyang Zou, Xuefeng Li, Yixiu Liu, Yuxiang Zheng, Ethan Chern, Shijie Xia, Yiwei Qin, Weizhe Yuan, and Pengfei Liu. O1 replication journey–part 2: Surpassing o1-preview through simple distillation, big progress or bitter lesson? *arXiv preprint arXiv:2411.16489*, 2024.

Junpeng Liu, Tianyue Ou, Yifan Song, Yuxiao Qu, Wai Lam, Chenyan Xiong, Wenhu Chen, Graham Neubig, and Xiang Yue. Harnessing webpage uis for text-rich visual understanding, 2024. URL `https://arxiv.org/abs/2410.13824`.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*, 2024.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Shikhar Murty, Dzmitry Bahdanau, and Christopher D Manning. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator. *arXiv preprint arXiv:2410.02907*, 2024.

OpenAI. Hello gpt-4o. openai.com, May 2024. URL `https://openai.com/index/hello-gpt-4o/`.

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

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale, 2024. URL `https://arxiv.org/abs/2409.15637`.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

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.

Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984. doi: 10.1145/357401.357402.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL `https://arxiv.org/abs/2408.03314`.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis, 2025. URL `https://arxiv.org/abs/2412.19723`.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

ByteDance Seed Team. Introducing ui-tars-1.5. seed-tars.com, 2025. URL `https://seed-tars.com/1.5/`.

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

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2025. URL `https://arxiv.org/abs/2408.00724`.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024a.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. Os-atlas: A foundation action model for generalist gui agents, 2024b. URL `https://arxiv.org/abs/2410.23218`.

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, 2024. URL `https://arxiv.org/abs/2404.07972`.

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

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, 2025a. URL `https://arxiv.org/abs/2412.09605`.

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, 2025b. URL `https://arxiv.org/abs/2412.04454`.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

A Training Details

The PC Agent-E model is fine-tuned on a dataset of 27k samples for 2 epochs using 32 NVIDIA GPUs over approximately 5 hours, with Qwen2.5-VL-72B as the base model. We set the context length to 8,192 tokens, using a batch size of 128 and a learning rate of 2e-6. The training process employs cosine annealing for learning rate scheduling, with a warm-up ratio of 0.05. The visual tower is kept frozen throughout the training process.

B Evaluation On Original WindowsAgentArena Benchmark

We also evaluate PC Agent-E's performance on the original WindowsAgentArena benchmark. As shown in Table 4, our model greatly surpasses the previous SOTA method, NAVI, across all task categories. NAVI is a complex agent framework that integrates GPT-4V Achiam et al. (2023) with a specialized tool called Omniparser Lu et al. (2024).

Method	Office	Web	System	Coding	Media & Video	Utils	Overall
NAVI Bonatti et al. (2024)	0.0	27.3	33.3	27.3	30.3	8.3	19.5
PC Agent-E (Ours)	**2.3**	**33.1**	**70.6**	**37.5**	**33.3**	**25.0**	**27.9**

Table 4: Success Rate (%) on the original WindowsAgentArena benchmark.

C Experiment on Smaller Model

To test our method on smaller models, we also trained Qwen2.5-VL-7B Bai et al. (2025) on the same 27K dataset for PC Agent-E, resulting in the **PC Agent-E 7B**. As shown in Table 5, our method yields improvements on the 7B model as well, although the gains are not as significant as those observed with the 72B model. This is because our method mainly enhances the model's *planning* ability, as discussed in §5.2, but the small model's deficiencies in *knowledge* and *grounding* limit the overall performance gain.

Method	Overall
Qwen2.5-VL-7B	5.0
PC Agent-E 7B	**6.4**

Table 5: Success Rate (%) on WindowsAgentArena-V2.

D Prompts

D.1 Thought Completion

Table 6: Thought Completion Prompt

Prompt for Thought Completion

```
You are a helpful computer use agent designed to complete tasks on a
computer. Your goal is to recreate your thought process behind a
specific action.

You will be provided with:

1. The task you are attempting to complete.
2. A history of the steps you have already performed (up to 50, if any;
none if it was the first action).
3. The specific action you chose to take.
4. The name of the element you clicked (if you clicked on an element). It
 might be too general or vague, you have to decied what to click based on
the screenshot.
5. A screenshot of the computer screen at the moment you decided to take
the action.
6. The red marks on the screenshot indicate the position of the click or
drag action.

To formulate your thought process, consider:

1. What do you observe on the screen? Consider your task and previous
action when you analyzing current screenshot.
2. Evaluate your previous action (if applicable):
   - Did it achieve the intended effect? If not, identify possible
   reasons (e.g., misclick, inactive element).
      Some typical examples for ineffective action:
       - misclick in an empty space
       - ineffective opening some elements without double click
       - ineffective type text/ press key because of inactivated input
       box
   - Did the result align with your previous plan, or did something
   unexpected happen?
      Some typical examples for ineffective action:
         - misclick in a wrong element
         - forget to clear existing text in input bar
3. Based on your action history, assess your progress toward completing
 the overall task.
4. Consider if you're exploring how to finish the task because of failed
 attempts in history steps.


Present your thought process as a clear, natural first-person
narrative that explains your reasoning at that moment.


Important requirements:
1. **DO NOT** mention the red marks in your response. These marks were
**added after the fact**
to indicate the position of your click or drag actions, and they were
not on the screen when you made the decision. **DO NOT** mention "red
box", "red square", "red circle", or "red arrow" in your response.
```

```
2. Write as if you are thinking in real-time before taking the action.
Do not include post-action evaluation or hindsight.

The task you are attempting to complete: {task_description}
Your performing history: {history_str}
The specific action you chose to perform: {action}
```

D.2 Trajectory Boost

Table 7: Trajectory Boost Prompt

Prompt for Trajectory Boost

```
You are a helpful assistant who can help users complete computer tasks,
with **full permission** to make any operations on the user's computer.
 The operating system is windows.
Based on the provided current state, you need to suggest the next action
 to complete the task. Do not try to complete the entire task in one step.
 Break it down into smaller steps, and at each step you will get a new
state to interact with.

IMPORTANT: You must strictly adhere to the following rules:

1. Choose ONLY ONE action from the list below for each response, DO NOT
perform more than one action per step.
2. Follow the exact syntax format for the selected action, DO NOT create
 or use any actions other than those listed.
3. Once the task is completed, output action finish.

Valid actions:
1. click (x, y)
   click the element at the position (x, y) on the screen
2. right click (x, y)
   right click the element at the position (x, y) on the screen
3. double click (x, y)
   double click the element at the position (x, y) on the screen
4. drag from (x1, y1) to (x2, y2)
   drag the element from position (x1, y1) to (x2, y2).
5. scroll (x)
   scroll the screen vertically with pixel offset x. Positive values of
   x: scroll up, negative values of x: scroll down.
6. press key: key_content
   press the key key_content on the keyboard.
7. hotkey (key1, key2)
   press the hotkey composed of key1 and key2.
8. hotkey (key1, key2, key3)
   press the hotkey composed of key1, key2, and key3.
```

```
9. type text: text_content
   type content text_content on the keyboard.
   Note that before typing text, you need to ensure the text box or input
    field is active/focused first. If the text box is not yet activated,
   you should first click on it to activate it, and then use type text in
   a separate step.
10. wait
    wait for some time, usually for the system to respond, screen to
    refresh, advertisement to finish.
11. finish
    indicating that the task has been completed.
12. fail
    indicating that the task has failed, of this task is infeasible
    because not enough information is provided.

Before deciding your next action, you should think carefully about the
current state of the screen and your history steps. Contain the
following points in your thought process:

1. What do you observe on the screen? Consider your task and previous
action when you analyzing current screenshot.
2. What's your previous plan and action (if applicable)? Evaluate your
previous plan and action in three conditions:
   1. It didn't make any effect. You should dentify possible reasons (e.
   g., misclick, inactive element) and adjust your plan in this step.
      Some typical examples for ineffective action:
       - misclick in an empty space
       - ineffective opening some elements without double click
       - ineffective type text/ press key because of inactivated input
       box
   2. It made some effect, but the result does not align with previous
   plan. You should dentify possible reasons (e.g., misclick, inactive
   element) and correct it in this step.
      Some typical examples for ineffective action:
         - misclick in a wrong element
         - forget to clear existing text in input bar
   3. It made some effect, and it successfully align with previous plan.
   You should progress to the next step based on the current state.
3. Based on your action history, assess your progress toward completing
 the overall task.
4. Exploring new ways to finish the task if there are already failed
attempts in history steps. **DO NOT repeat** the history actions.

Response Format: Your thought process\n\nAction: The specific action
you choose to take.

The task you are attempting to complete: {task_description}
Your performing history: {history_str}
Given the screenshot as below. What's the next step that you will do to
help with the task?
```

D.3 PC AGENT-E SCAFFOLD

Table 8: PC Agent-E scaffold Prompt

Prompt for PC Agent-E scaffold

```
You are a helpful assistant who can help users complete computer tasks,
with **full permission** to make any operations on the user's computer.
Based on the provided current state, you need to suggest the next action
 to complete the task. Do not try to complete the entire task in one step.
 Break it down into smaller steps, and at each step you will get a new
state to interact with.
IMPORTANT: You must strictly adhere to the following rules:
1. Choose ONLY ONE action from the list below for each response, DO NOT
perform more than one action per step.
2. Follow the exact syntax format for the selected action, DO NOT create
 or use any actions other than those listed.
3. Once the task is completed, output action finish.

Valid actions:
1. click (x, y)
click the element at the position (x, y) on the screen

2. right click (x, y)
right click the element at the position (x, y) on the screen

3. double click (x, y)
double click the element at the position (x, y) on the screen

4. drag from (x1, y1) to (x2, y2)
drag the element from position (x1, y1) to (x2, y2).

5. scroll (x)
scroll the screen vertically with pixel offset x. Positive values of x:
scroll up, negative values of x: scroll down.

6. press key: key_content
press the key key_content on the keyboard.

7. hotkey (key1, key2)
press the hotkey composed of key1 and key2.

8. hotkey (key1, key2, key3)
press the hotkey composed of key1, key2, and key3.

9. type text: text_content
type content text_content on the keyboard.

10. wait
wait for some time, usually for the system to respond, screen to refresh
, advertisement to finish.

11. finish
indicating that the task has been completed.

12. fail
indicating that the task has failed, of this task is infeasible because
not enough information is provided.
```

```
Response Format: {Your thought process}
Action: {The specific action you choose to take}

Your task is: {task_description}
History of the previous actions and thoughts you have done to reach the
current screen: {history_str}
--------------------------------------------
Given the screenshot, what’s the next step you will do to help with the
task?
```

E PC Tracker User Manual

1. Introduction

PC Tracker is a lightweight infrastructure for efficiently collecting large-scale human-computer interaction trajectories. The program runs seamlessly in the background, automatically capturing screenshots and keyboard & mouse activities.

2. Installation

- Ensure your operating system is Windows.
- Extract our software package to a location with sufficient disk space (recommended to have more than 3GB of available space for storing recorded data).

3. Quick Start

- (Optional) Set screen resolution to 16:9.
- Open the extracted folder and launch `main.exe`.

4. Instructions

After launching the tracker, you can choose between **Task Oriented Mode** or **Non-Task Oriented Mode** for recording.

Task Oriented Mode

This mode is divided into two sub-modes: **Given Task** and **Free Task**.

Given Task In this mode, you will be assigned an uncompleted task each time.

- **Next Task**: Click Next Task to get the next task.
- **Previous Task**: Click Previous Task to return to the previous task.
- **Bad Task Feedback**: If you think the current task is difficult to complete, click Bad Task to discard it permanently. Alternatively, you can start the task and modify its description after completion based on your actual execution.
- **Start Recording**: Click Start, and the tracker window will automatically minimize while recording begins.
- **End Task**: After completing the task, click Finish to save the record. Or if the task execution fails or you don't want to record it, click Fail.
- **Modify Task Description**: After finishing the task, you can modify the task description based on your actual execution.

Free Task In this mode, you can freely use the computer and summarize the task description and difficulty yourself.

- **Start Recording**: Click Start, and the tracker window will automatically minimize while recording begins.
- **Save and Summarize This Record**: Fill in the task description, select difficulty (easy/medium/hard), and click Save to save the record.
- **Discard This Record**: Click Discard to discard the record.

Non-Task Oriented Mode

In this mode, you can freely use the computer, with similar methods to start and stop recording as described above.

5. Usage Notes

- **Does not currently support using extended screens.**
- **Does not currently support using Chinese input methods.**
- **Does not currently support using touchpads.**
- **The tracker window is fixed in fullscreen.** To support the filtering of tracker-related actions (such as clicking the Start button) in post-processing, the tracker window is fixed in fullscreen. You can reopen the tracker window by clicking to view the task description, then minimize it again, but please do not drag it to display in a non-fullscreen state.

6. Data Privacy

- After starting recording, your screenshots and keyboard & mouse operations will be automatically recorded. PC Tracker does not record any information from unopened software. If you believe the recording may infringe on your privacy, you can choose to discard the record.
- Collected data is saved in the `./events` folder (hidden by default). Each trajectory includes a Markdown file for easy visualization.

7. FAQ

Does the tracker have networking capabilities? PC Tracker is completely local, does not support networking, and will not upload your data.

What if my computer screen resolution is not 16:9? If your screen resolution is not 16:9, it will affect the subsequent unified processing of data. We recommend adjusting your screen resolution to 16:9.

How much space does the collected data occupy? The specific data size varies. Generally, even with intensive recording operations for 1 hour, it will not generate more than 1GB of data.

F THE USE OF LLMS

We used LLMs to improve the grammar, clarity, and overall readability of this paper. All research ideas, content, and scientific contributions were developed and written by the human authors. All suggestions from LLMs were reviewed and edited by the authors, who retain full responsibility for the final content of this paper.