

OSWORLD: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments

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[◇] Tao Yu[♣]
[♣]The University of Hong Kong [♣]CMU [♡]Salesforce Research [◇]University of Waterloo

Abstract

Autonomous agents that accomplish complex computer tasks with minimal human interventions have the potential to transform human-computer interaction, significantly enhancing accessibility and productivity. However, existing benchmarks either lack an interactive environment or are limited to environments specific to certain applications or domains, failing to reflect the diverse and complex nature of real-world computer use, thereby limiting the scope of tasks and agent scalability. To address this issue, we introduce OSWORLD, the *first-of-its-kind scalable, real computer environment* for multimodal agents, supporting task setup, execution-based evaluation, and interactive learning across various operating systems such as Ubuntu, Windows, and macOS. OSWORLD can serve as a unified, integrated computer environment for assessing *open-ended* computer tasks that involve arbitrary applications. Building upon OSWORLD, we create a benchmark of 369 computer tasks involving real web and desktop apps in open domains, OS file I/O, and workflows spanning multiple applications. Each task example is derived from real-world computer use cases and includes a detailed initial state setup configuration and a custom execution-based evaluation script for reliable, reproducible evaluation. Extensive evaluation of state-of-the-art LLM/VLM-based agents on OSWORLD reveals significant deficiencies in their ability to serve as computer assistants. While humans can accomplish over 72.36% of the tasks, the best model achieves only 12.24% success, primarily struggling with GUI grounding and operational knowledge. Comprehensive analysis using OSWORLD provides valuable insights for developing multimodal generalist agents that were not possible with previous benchmarks. Our code, environment, baseline models, and data are publicly available at <https://os-world.github.io>.

1 Introduction

Humans interact with computers to perform essential tasks in the digital realm, including web browsing, video editing, file management, data analysis, and software development. These task workflows often involve multiple applications through graphical user interfaces (GUI) and command line interfaces (CLI). Autonomous digital agents, powered by advancements in large vision-language models (VLMs), have the potential to revolutionize how we interact with computer environments [27, 41, 1]. By following high-level natural language instructions, these agents can make digital interactions more accessible and vastly increase human productivity. However, a major challenge in developing such multimodal agents is the absence of a benchmark based on a real interactive environment that covers the diversity and complexity of real-world computer use across various operating systems, interfaces, and applications, consequently restricting task scope and agent scalability.

Previous benchmarks provide datasets of demonstrations without executable environments [9, 38, 20]. Their non-execution-based evaluation assumes a single solution for each task and wrongfully penalizes alternative correct solutions. These benchmarks also miss opportunities for essential

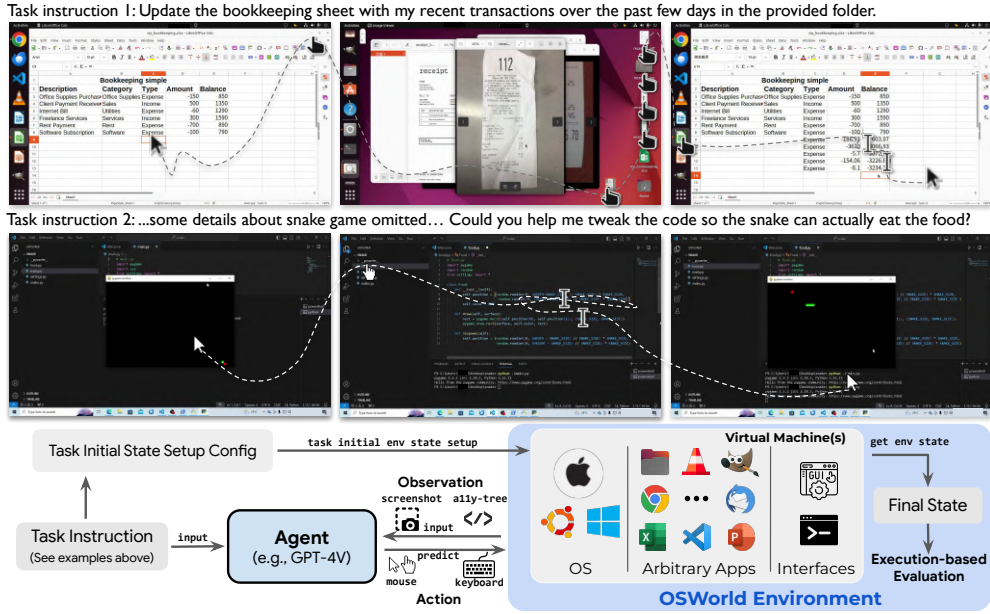


Figure 1: OSWORLD is a *first-of-its-kind scalable, real computer environment* for multimodal agents, supporting task setup, execution-based evaluation, and interactive learning across operating systems. It can serve as a unified environment for evaluating *open-ended* computer tasks that involve arbitrary apps (e.g., task examples in the above Fig). We also create a benchmark of 369 real-world computer tasks in OSWORLD with reliable, reproducible setup and evaluation scripts.

agent development methods like interactive learning and real-world exploration. Building realistic interactive environments is a major challenge in developing multimodal agents. Prior work that introduce executable environments simplify the observation and action spaces of human-computer interaction and limit task scope within specific applications or domains, such as web navigation in a few domains [41, 29, 54, 62], coding [53] and the combination [31, 50, 33]. Agents developed in these restricted environments cannot comprehensively cover computer tasks, lacking the support of evaluating tasks in complex, real-world scenarios that require navigating between applications and interfaces in open domains (task examples in *e.g.*, Fig. 1).

To address this gap, we introduce OSWORLD, the *first-of-its-kind scalable, real computer environment* designed for the development of multimodal agents capable of executing a wide range of real computer tasks beyond isolated interfaces and applications. This executable environment allows free-form raw keyboard and mouse control of real computer applications and supports initial task state configuration, execution-based evaluation, and interactive learning across mainstream operating systems (*e.g.*, Ubuntu, Windows, macOS). As shown in Fig. 1, OSWORLD enables evaluation of *open-ended* computer tasks that involve arbitrary applications, ranging from image viewing to software functionality integration and programming. Thus, OSWORLD can serve as a unified, real computer environment that allows users to define their agent tasks without the need to build application/domain-specific simulated environments.

Building upon OSWORLD, we create a benchmark with 369 real-world computer tasks that involve widely-used web and desktop apps in open domains, OS file I/O, and multi-app workflows through both GUI and CLI. Each task example is based on real-world computer use cases experienced by real users and often requires interactions with multiple applications and interfaces. To ensure reliable, reproducible assessment within the OSWORLD environment, 9 authors with computer science backgrounds carefully annotate each example with an initial state setup configuration to simulate human work in progress and a custom execution-based evaluation script to verify task completion. Our benchmark has a total of 134 unique evaluation functions, which are orders of magnitude larger than prior work [62], showcasing the complexity, diversity, and evaluation challenges of tasks in our benchmark. The human performance study indicates that task examples from OSWORLD are more time-consuming and challenging compared to those in prior work.

We extensively evaluate state-of-the-art LLM and VLM-based agent baselines, including the GPT-4V series [37], the Gemini-Pro models [45] and the Claude-3 Opus [3], as well as Mixtral [18] and CogAgent [16] from the open-source community. The performance of these experiments ranges from 0.99% to 12.24%, with subsets of applications even reaching 0%, for workflow tasks that involve

cooperation from multiple apps, the highest performance of the baseline agent is only 6.57%. This indicates that current LLMs and VLMs are far from capable of serving as computer assistants (§4.2). Results also show that while additional knowledge such as the accessibility tree and Set-of-Mark (§4.1) can be helpful, it can also lead to potential misguidance and varies across models. We also observe performance changes in these agents compared to consistent human performance across different types of computer tasks. Analysis reveals that VLM-based agents struggle to ground on screenshots to predict precise coordinates for actions, tend to predict repetitive actions, are unable to handle noise from unexpected application windows and exhibit limited knowledge of basic GUI interactions and domain-specific features of apps (§5.2, §5.4). Feeding higher resolution and more trajectory history can help improve the performance by even doubling while requiring longer context length and efficient modeling (§5.2). We open-source OSWORLD environment and benchmark, including environment initial state setup, reliable evaluation scripts, documentation, and our implementation of baseline models to promote research towards the goal of generalist capable computer agents¹. Future work can focus on enhancing VLM GUI grounding abilities, including interaction commonsense knowledge, higher-resolution support, and coordinates accuracy for more robust GUI interactions. Additionally, efforts can be made to improve agent architectures to better handle complex computer tasks through exploration, memory, and reflection.

2 OSWORLD Environment

In this section, we will introduce the task definition of autonomous agents, the components and implementation of the OSWORLD environment, and the supported observation and action spaces.

2.1 Task Definition

An autonomous digital agent task can be formalized as a partially observable Markov decision process (POMDP) $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ with state space \mathcal{S} , observation space \mathcal{O} (§2.3, including natural language \mathcal{D}), action space \mathcal{A} (§2.4), transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Given current observation $o_t \in \mathcal{O}$ (a natural language instruction observation and a screenshot observation (*e.g.*, computer screenshot), ally tree, or their combination according to facilities available), an agent generates executable action $a_t \in \mathcal{A}$ (*e.g.*, clicking on the certain pixel of the screen — `.click(300, 540, button='right')`, press key combination — `.hotkey('ctrl', 'alt', 't')`), which results in a new state $s_{t+1} \in \mathcal{S}$ (*e.g.*, current Desktop environment) and a new partial observation $o_{t+1} \in \mathcal{O}$ (*e.g.*, current screenshot). The interaction loop repeats until an action that marks termination (DONE or FAIL, see Sec. 2.4) is generated or the agent reaches the max number of steps (*e.g.*, 20 in our experiments). In this version of OSWORLD, we implement an execution-based reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ (§2.2.3). The reward function awards a value of 1 or a positive decimal under 1 at the final step if the state transitions meet the expectations of the task objective (*i.e.*, the goal is successfully achieved or partially achieved), or if the agent accurately predicts failure for an infeasible task. In all other scenarios, it returns 0.

2.2 Real Computer Environment Infrastructure

OSWORLD is an executable and controllable environment that supports task initialization, execution-based evaluation, and interactive agent learning in a range of *real* operating systems (*e.g.*, Ubuntu, Windows, macOS) using virtual machine techniques, shown in the middle and right of Fig. 2. Virtual machine offers a safe isolated environment and prevents the agent resulting in irreversible damaging effect on the real host machine. The snapshot feature also enables efficient reset of the virtual environment. The environment is configured through a config file (shown in the left of Fig. 2) for interface initialization during the initialization phase (including downloading files, opening software, adjusting interface layout) (§2.2.2, highlighted with red in Fig. 2), post-processing during the evaluation phase (activating certain windows, saving some files for easy retrieval of information, highlighted with orange), and acquiring files and information for evaluation (such as the final spreadsheet file for spreadsheet tasks, cookies for Chrome tasks, highlighted with yellow in Fig. 2), as well as the evaluation functions and parameters used (§2.2.3, highlighted with green in Fig. 2). See App. A.1 for more details.

¹<https://os-world.github.io>

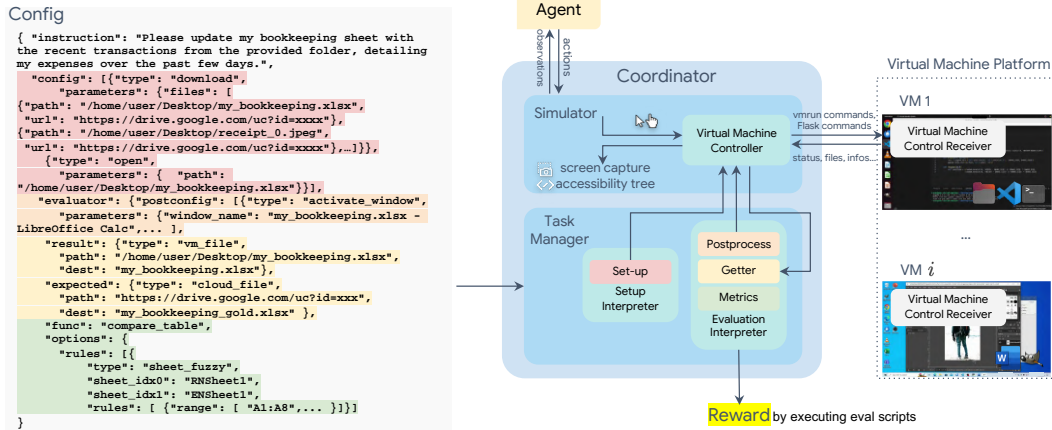


Figure 2: Overview of the OSWORLD environment infrastructure. The environment uses a configuration file for initializing tasks (highlighted in red), agent interaction, post-processing upon agent completion (highlighted in orange), retrieving files and information (highlighted in yellow), and executing the evaluation function (highlighted in green). Environments can run in parallel on a single host machine for learning or evaluation purposes. Headless operation is supported.

2.2.1 Overview

OSWORLD environment runs on the host machine. Its Coordinator accepts a configuration file at the initialization of a computer task, runs commands to automatically create a virtual machine instance, and initializes the required state for the task through the Task Manager. The configuration file specifies the snapshot of the virtual machine to be used (which stores the complete state of a computer at a certain moment and can be restored to this state at any time) and also indicates the information needed for setup (such as downloading files and opening some software, making some additional settings, etc.). Once the environment is set up, agents start to interact with the environment, receiving observations such as screenshots, the accessibility (a11y) tree, and customized streams such as terminal outputs. Agents subsequently generate executable actions (e.g., `.click(300, 540)`) that manipulate the keyboard and mouse. Each action of the agent is input into the environment as a code string, and the environment’s Simulator executes them in the virtual machine. After the completion of a task, the Task Manager performs post-processing (such as file saving, or reopening certain apps) according to the task’s post-config, retrieves data to the host machine (fetching images or configuration files from the virtual machine or cloud, etc.), and then runs evaluation scripts to assess the completion of the task. Multiple virtual machines can run simultaneously on a single host machine, thereby parallelizing training and evaluation.

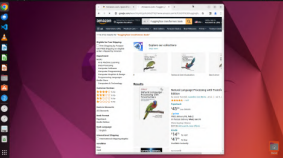
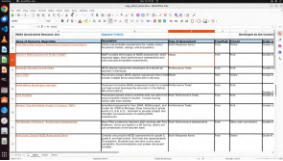
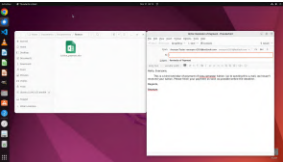
2.2.2 Initial Task Environment Setup

Many real-world scenarios requiring assistance occur not at the beginning of digital activities, such as right after launching an application or when a computer has just been started, but rather at intermediate stages, such as when certain software is already open or the computer has experienced a crash. Therefore, we aim to simulate these intermediate states as closely as possible to replicate real-world scenarios. The naturalness we bring in also leads to more challenges for agents to model and explore. We adopted a hybrid approach for configuration instead of solely relying on example-wise snapshots for restoration since it would store much unnecessary hardware state information, resulting in each example requiring gigabytes of space. The procedure is divided into three stages: start the VM emulator, prepare files (download the files or scripts from the cloud, *etc.* optional), and execute reprocessing commands (open files or tabs, change the window size, *etc.* optional). We provide convenient APIs to configure initial conditions and world settings, standardizing our tasks to make this process user-friendly and easily extendable for scaling. For more details on setup see App. B.5.

2.2.3 Execution-Based Evaluation

Evaluating the successful execution of general computer tasks presents a significant challenge, as these tasks defy reduction to a uniform pattern or measurement by a single metric. To ensure a thorough assessment, we design example-specific evaluation metrics including pre-setup, post-processing, and

Table 1: Examples of our annotated evaluation scripts, which involve retrieving data from configuration files, the environment, and the cloud, and executing functions to assess functional correctness and obtain results. The example-wise evaluation facilitates the diversity of tasks and reliable evaluation of complex, real-world, open-ended tasks.

Initial State	Task Instruction	Evaluation Script (Simplified)
	<i>Can you help me clean up my computer by getting rid of all the cookies that Amazon might have saved?</i>	<pre>cookie_data = get_cookie_data(env) rule = {"type": "domains", "domains": [".amazon.com"]} is_cookie_deleted(cookie_data, rule)</pre>
	<i>Rename "Sheet 1" to "LARS Resources". Then make a copy of it. Place the copy before "Sheet 2" and rename it by appending a suffix "(Backup)", ...</i>	<pre>result = get_file(env) expected = get_file(cloud) rules = [{"type": "sheet_name", {"type": "sheet_data", "sheet_idx0": 0, "sheet_idx1": 1}...]} compare_table(result, expected, rules)</pre>
	<i>I've drafted an e-mail reminder for those who haven't paid tuition. Please help me to check out their e-mails from the payment record and add to the receiver field.</i>	<pre>tree = get_a11y_tree(env) rules = [{"selectors": ["tool-bar[attr id=MsgHeadersToolbar] label[name=To] [attr class=\"address-pill\"]> label[attr class=\"pill-label\"] [name*=\"fox@someuniversity.edu...]] check_a11y_tree(tree, rules)</pre>

dedicated functions, tailored to the software in use and the task's specific requirements. This involves interpreting the software's internal files, utilizing specific packages, and preemptively setting up scaffolding based on the software's permissions (e.g., opening remote debugging ports for Chrome and VLC, creating extensions for VS Code). Occasionally, this process may also require assistance from reverse engineering tools, such as for decrypting account information in Thunderbird.

As a result, we construct a vast collection of functions that make final wrangling and retrieve files and data information of varying types, categories, and granularities from the cloud and software from virtual machines as well as evaluation functions covering different aspects and their combinations, inputting this information as parameters to assess the outcomes. We show some evaluation examples in Tab. 1. , demonstrate the retrieval of cookie data from virtual machines, obtaining files from both virtual machines and cloud services, fetching the current runtime interface's accessibility tree from the virtual machines, and determining success based on this information whether Amazon's cookies have been deleted, whether the generated table is accurate, and whether the correct interface has been accessed. Need to note when the type of task has real-time characteristics (such as the number of citations of someone's paper, the content of blogs, etc.), we include dynamic functions (such as crawler scripts) inside getter to obtain the real-time values at the moment of evaluation and then use them to compare with the results obtained by the agent upon task completion. See more in App. B.6.

2.3 Observation Space

The observation space in OSWORLD contains a **complete screenshot of the desktop screen**, including the mouse's position and shape, various application windows, files, and folders that are opened in different sizes and orders, maintaining the same perception as a human. Also, to be aligned with previous agent-building web and mobile research [29, 26, 9, 62] that provide and support the use of the webpage's DOM and app's view hierarchy, OSWORLD also provides **XML-format accessibility (a11y) tree** (obtained via ATSPI² on Ubuntu, via PyWinAuto on Windows, etc.), which can support additional information for modeling. These raw observations allow rich interactions between multiple applications but induce challenges in long-horizon decision-making from high-resolution images (e.g., 4k screenshots) and structured long text (e.g., accessibility trees). For more detailed information on observation space, refer to App. A.2.

2.4 Action Space

²<https://docs.gtk.org/atspi2/>

Action space \mathcal{A} in OSWORLD encompasses all mouse and keyboard actions, including movement, clicks (left-key, right-key, multiple clicks), dragging, keystrokes, hotkeys, and others, covering all human-computer action space. Some action examples are shown in Tab. 2 and the complete action list can be found in Appendix A.3. We use the widely used mouse and keyboard control library `pyautogui`³ for our action space. This library leverages the high-level programming language Python to replicate and replay various human inputs into computers through code, allowing us to construct a universal and complete representation of actions. The agent must generate syntax-correct `pyautogui` Python code to predict valid actions. Basic actions, such as `press` and `moveTo`, can be integrated within program structures, such as for-loops, significantly improving the expressiveness of an action. Timing is also crucial, as highlighted in previous studies on mobile devices [46], as well as the ability to determine whether a task is infeasible or completed. Therefore, we add three special actions named `WAIT`, `FAIL`, and `DONE` to enhance the aforementioned action spaces. Previous efforts towards creating domain-specific agents, such as MiniWoB++ [41, 29], CC-Net [17], and WebArena [62, 21], have defined action spaces that include clicks and typing, as well as some actions specially designed for web browsing. However, they do not model all possible actions on a computer, leading to limitations when attempting actions like right-clicking and clicking with the `ctrl` key held to select items. This imposes an upper bound on agent learning capabilities.

Table 2: Some examples of the mouse and keyboard actions \mathcal{A} in OSWORLD. See App. A.3 for the complete list.

Function	Description
<code>moveTo(x, y)</code>	Moves the mouse to the specified coordinates.
<code>click(x, y)</code>	Clicks at the specified coordinates.
<code>write('text')</code>	Types the specified text at the current cursor location.
<code>press('enter')</code>	Presses the Enter key.
<code>hotkey('ctrl', 'c')</code>	Performs the Ctrl+C hotkey combination (copy).
<code>scroll(200)</code>	Scrolls up by 200 units.
<code>scroll(-200)</code>	Scrolls down by 200 units.
<code>dragTo(x, y)</code>	Drags the mouse to the specified coordinates.
<code>keyDown('shift')</code>	Holds down the Shift key.
<code>keyUp('shift')</code>	Releases the Shift key.
<code>WAIT</code>	Agent decides it should wait.
<code>FAIL</code>	Agent decides the task is infeasible.
<code>DONE</code>	Agent decides the task is finished.

3 OSWORLD Benchmark

We introduce the OSWORLD benchmark, which encompasses 369 real computing tasks defined and executed on Ubuntu. Additionally, we provide a set of 43 tasks for Windows built on OSWORLD environment⁴. The environment preparation, annotation process, data statistics, and human performance are described in this section.

3.1 Operating System and Software Environments

OSWORLD supports real operating systems, including Windows, macOS, and Ubuntu, for the development of automated computer agents. For development purposes, we offer an extensive set of examples on Ubuntu and its open-source applications, leveraging their open-source nature and more accessible APIs for task setting and evaluation. We also provide annotated testing examples for Windows, focusing on applications with similar functionalities. For the first time, our real OS environments enable us to define all kinds of computer tasks, including those that involve interacting with multiple applications (e.g., Chrome and file manager) and interfaces (GUIs and CLIs). Considering availability, the strength of the user community, and diversity, we mainly focus on eight representative applications as well as the basic ones the system provides: Chrome for web browsing, VLC for media playback, Thunderbird for email management, VS Code as a coding IDE, and LibreOffice (Calc, Writer, and Impress) for handling spreadsheets, documents, and presentations respectively, GIMP for image editing, and other basic OS apps like terminal, file manager, image viewer, and PDF viewer. Each example drawn from these applications separately or in combination showcases distinct operational logic and necessitates skills including commonsense knowledge, high-resolution perception, mastery of software shortcuts, and the precise controlling of mouse and keyboard movements. For more details, check App. B.1 and B.2.

3.2 Tasks

We create a benchmark suite of 369 real-world computer tasks on Ubuntu environment collected from authors and diverse sources such as forums, tutorials, guidelines, *etc.*, to show the capability

³<https://pyautogui.readthedocs.io/en/latest/>

⁴Due to copyright issues, these Windows tasks require further activation by the user.

for open-ended task creation within OSWORLD. Each example is carefully annotated with a natural language instruction, a setup configuration with corresponding files and setup actions for initialization of initial states upon our provided VM image, and a manually crafted evaluation script to check if the task is successfully executed. We also adapt 43 tasks from the Ubuntu set for analytic usage on Windows. Overall, it takes 9 computer science students (all student authors) over 3 months, consuming approximately 1800 man-hours (650 hours on single-app tasks, 750 hours on workflow tasks and 400 hours for double-checking).

Task instructions and scenarios To draw the most diverse and close-to-reality usage cases, we explore several types of resources, including official guidelines & tutorials, video pieces giving tips and tutorials on the Internet (*e.g.*, TikTok and YouTube), how-to websites (*e.g.*, WikiHow), Q&A forums (*e.g.*, Reddit, Quora, Superuser, & StackOverflow), formal video courses (*e.g.*, Coursera and Udemy), and public-available personal blogs & guidelines. The detailed resources used in our benchmark are listed in App. B.3. The examples are selected by judging their popularity, helpfulness, and diversity, revealed by the views and votes. Meanwhile, we notice that it is challenging to find enough examples on the internet for tasks that involve the collaboration of multiple software applications. Therefore, the authors conducted extensive brainstorming, combining some existing examples or drawing inspiration from daily-life scenarios, to compile the tasks. The instructions and task-related files are then crafted from these real-world guidelines and questions by the authors. After the selection, each example will be cross-checked by the other two authors on the feasibility, ambiguity, and alignment with the source. We not only collect tasks that can be finished, but also collect the infeasible ones that are inherently impossible to be completed due to deprecated features or hallucinated features raised by real users, which results in 30 infeasible examples in our benchmark. Additionally, to demonstrate the unification ability of OSWORLD environment for the creation of open-ended computer tasks, we also integrate 84 examples from other benchmarks focusing on single-application or domain-specific environments such as NL2Bash [28], Mind2Web [9], SheetCopilot [24], PPTC [13], and GAIA [34]. Refer to App. B.4 for more details and B.8 for sampled examples for the showcase. A total of about 400 man-hours were spent to collect these examples.

Initial state setup configs To construct the initial state, we prepare the files required for the task and set up the initial state. For the files, we try to obtain them from the sources of the tasks we found, or, in cases where the files are not publicly available, we recreate them as realistically as possible based on scenarios. For the initial state setup, we also developed some functions based on the APIs of software and OS to control the opening and resizing of software windows and reimplement some functions that are difficult to achieve with APIs using *pyautogui*. For different tasks, we write configs to set the files and initial steps in the virtual machine and verify them in the environment. For example, the setup stage (highlighted in red color, keyed as “config”) in Figure 2 involves downloading files into the virtual machine to prepare a close-to-reality initial environment, and then open the file of interest with the corresponding application. The setup steps for each example take about 1 man-hour to construct.

Execution-based evaluation For each task, we select the appropriate getter functions, evaluator functions, and parameters to compose the configuration file. The getter function is used to extract key components (*e.g.*, the modified file, the text contents displayed in a window element) from the final state of the environment, and the evaluator function assesses success based on the extracted key components. If a function does not exist, we will construct it and add it to the function library of the environment. After completing each evaluation, the annotator conducts initial tests with self-designed test cases. Then, in the human evaluation and experiment running phases, each example is further scrutinized and iterated upon by different individuals three times from the perspective of alignment with the instruction and correctness under different solutions. As a result, we implement nearly sample-specific executable evaluation scripts, resulting in a total of 134 unique evaluation functions for assessing functional correctness—significantly more than the previous benchmarks. The average time spent on developing the evaluation for an example and its examination amounts to approximately 2 man-hours from graduate students.

Quality control Once annotation is finished, each example is attempted by two authors who did not participate in annotating that specific example, acting as agents to complete the task. This process evaluates the current example’s quality and provides feedback to the annotators (such as unclear instructions or inability to complete the task, crashes in corner cases, serious instances of false

positives and negatives, *etc.*), and involves joint revisions and supplements. During experiments for human performance and baselines, we further fixed examples found to have issues, dedicating over 400 man-hours for four rounds of checks. Further investment of time and a more red teaming could further reduce false positives and negatives, which we will leave to future work.

3.3 Data Statistics

Table 3: Key statistics in OSWORLD. The “Supp. tasks” refers to the Windows-based tasks, that could only be used after activation due to copyright restrictions.

Statistic	Number
Total tasks (Ubuntu)	369 (100%)
- Multi-App Workflow	101 (27.4%)
- Single-App	268 (72.6%)
- Integrated	84 (22.8%)
- Infeasible	30 (8.1%)
Supp. tasks (Windows)	43
Initial States	302
Eval. Scripts	134

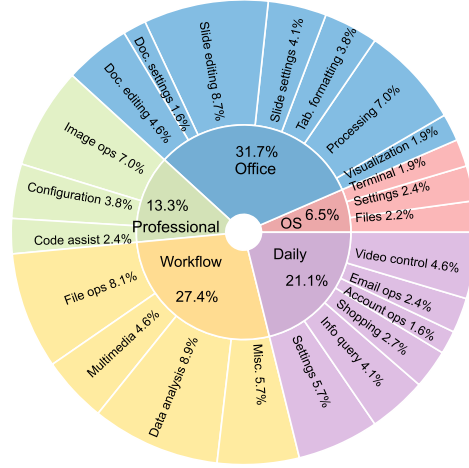


Figure 3: Distribution of task instructions in OSWORLD based on the app domains and operation types to showcase the content intuitively.

Statistics To facilitate the analysis and comprehension of the agent’s capabilities, we cluster the examples into the software categories. Specifically, these categories include OS, Office (LibreOffice Calc, Impress, Writer), Daily (Chrome, VLC Player, Thunderbird), Professional (VS Code and GIMP), and Workflow (tasks involving multiple apps). The main statistics of OSWORLD are presented in Tab. 3 and Fig. 3, showcasing the outline and a broad spectrum of tasks. Specifically, OSWORLD contains a total of 369 tasks (and an additional 43 tasks on Windows for analysis), with the majority (268 tasks or 72.6%) aiming at single application functionalities and a remarkable section of workflow-related tasks (101 tasks or 27.4%). The dataset’s diversity is further affirmed by the inclusion of tasks considered infeasible, totaling 30 tasks or 8.1% of the dataset. Additionally, a total of 84 tasks (22.8%) are integrated from related datasets, highlighting the dataset’s applicability in universal modeling. Remarkably, the dataset incorporates 302 distinct initial states and 134 different evaluation scripts, underscoring the comprehensive approach towards evaluating the tasks’ complexity and requirements. More statistic details are available in App. B.4.

Comparison with existing benchmarks OSWORLD is compared with a number of existing benchmarks in Table 4. OSWORLD take utilizes raw mouse and keyboard actions that is universal to the computer environment, rather than focusing on specific computer applications (*e.g.*, a browser [62, 9]), with multimodal observation including screenshot (Multimodal Support column). This universal action space enables the constructed agents to handle general tasks in the digital world. Our executable environment allows agents to freely explore during both the learning and evaluation phases, rather than providing only static demonstrations to evaluate an agent’s prediction of the next step (Executable Env. column). Moreover, it does not solely focus on interactions within a single app but also considers interactions across multiple apps and the overall task (Cross-App column). Unlike many evaluations that offer the same evaluation script or a few scripts for a certain type of task, the OSWORLD benchmark provides example-wise, execution-based evaluation for tasks. Specifically, the total of 134 unique execution-based evaluation functions in our benchmark is significantly more than previous work, demonstrating the complexity, diversity, and evaluation challenges of tasks in our benchmark (# Exec.-based Eval. Func. column). It also allow us to freely choose open-ended tasks and scale to new environments, rather than struggling in crafting new ones. Constructing intermediate initial states as task setup increases realism and poses challenges to the agents’ exploration capabilities (Intermediate Init. State column).

3.4 Human Performance

Table 4: Comparison of different environments for benchmarking digital agents. The columns indicate: whether they provide a *controllable* executable environment (Control. Exec. Env.), the ease of adding new tasks involving arbitrary applications in open domains (Environment Scalability), support for multimodal agent evaluation (Multimodal Support), support for and inclusion of cross-app tasks (Cross-App), capability to start tasks from an intermediate initial state (Intermediate Init. State), and the number of execution-based evaluation functions (# Exec.-based Eval. Func.).

	Size	Control. Exec. Env.?	Environment Scalability?	Multimodal Support?	Cross- App?	Intermediate Init. State?	# Exec.-based Eval. Func.
GAIA [34]	466	✗	-	✗	✗	✗	0
MIND2WEB [9]	2350	✗	-	✓	✗	✓	0
WEBLIX [32]	2337	✗	-	✓	✗	✓	0
PIXELHELP [26]	187	✗	-	✓	✗	✗	0
METAGUI [44]	1125	✗	-	✓	✗	✗	0
AITW [38]	30k	✗	-	✓	✗	✓	0
OMNIACT [20]	9802	✗	-	✓	✗	✓	0
AGENTBENCH [31]	1091	Multi-isolated	✗	✗	✗	✗	7
INTERCODE [53]	1350	Code	✗	✗	✗	✗	3
MINIWoB++ [29]	104	Web	✗	✓	✗	✗	104
WEBSHOP [54]	12k	Web	✗	✓	✗	✗	1
WEBARENA [62]	812	Web	✗	✓	✗	✗	5
VWEBARENA [21]	910	Web	✗	✓	✗	✗	6
WIKIHOW [57]	150	Mobile	✗	✓	✗	✗	16
ASSISTGUI [12]	100	✗	✗	✓	✗	✓	2
OSWORLD	369	Computer	✓	✓	✓	✓	134

We conduct human evaluations on each example in our dataset, with annotators being computer science major college students who possess basic software usage skills but have not been exposed to the samples or software before. We recorded the time required to complete each example and whether their completion of the example was correct. For comparison, we also sampled 100 examples from WebArena [62] under the same evaluation setup.

As illustrated, tasks from our dataset generally required more time to complete, with a median completion time of 111.94 seconds (compared to 35.38 seconds in WebArena), and a significant number of examples distributed at 900 seconds or even more. In terms of accuracy, the human performance on our tasks was approximately 72.36%, significantly lower than the 88% observed on the pure web task dataset. These findings highlight the complexity and challenge of tasks in our dataset, which demand more time and effort. The lower accuracy rate further indicates that our tasks require a higher level of understanding and proficiency, underscoring the need for advanced models and techniques to tackle them effectively.

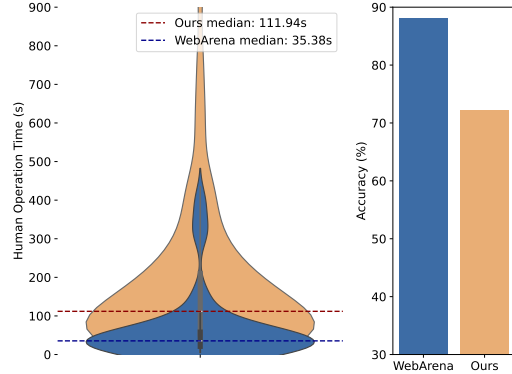


Figure 4: Human operation time and accuracy on OSWORLD and WebArena.

4 Benchmarking LLM and VLM Agent Baselines

In this section, we present the implementation details and experimental settings for several state-of-the-art LLM and VLM agent baselines on OSWORLD benchmark, as well as their performance.

4.1 LLM and VLM Agent Baselines

We adopt state-of-the-art LLM and VLM from open-source representatives such as Mixtral [18] and CogAgent [16], and closed-source ones from GPT, Gemini, and Claude families on OSWORLD, to serve as the foundation of agent. We also explore methods such as the Set-of-Marks aided approach [52, 10], which has been demonstrated to improve spatial capabilities for visual reasoning. Our prior experiments following VisualWebArena [21] adopt few-shot prompting, which involves using (observation, action) pairs as few-shot examples and inputting the current observation to

Table 5: Success rates of baseline LLM and VLM agents on OSWORLD, grouped by task categories: OS, Office (LibreOffice Calc, Impress, Writer), Daily (Chrome, VLC Player, Thunderbird), Professional (VS Code and GIMP) and Workflow (tasks involving multiple apps), for gaining insights from interfaces and operation logic. See App. C.1 and C.5 for more details.

Inputs	Model	Success Rate (\uparrow)					Overall
		OS	Office	Daily	Profess.	Workflow	
A11y tree	Mixtral-8x7B	12.5%	1.01%	4.79%	6.12%	0.09%	2.98%
	GPT-3.5	4.17%	4.43%	2.71%	0.00%	1.62%	2.69%
	Gemini-Pro	4.17%	1.71%	3.99%	4.08%	0.63%	2.37%
	GPT-4	20.83%	3.58%	25.64%	26.53%	2.97%	12.24%
Screenshot	CogAgent	4.17%	0.85%	2.71%	0.00%	0.00%	1.11%
	Gemini-ProV	8.33%	3.58%	6.55%	16.33%	2.08%	5.80%
	GPT-4V	12.5%	1.86%	7.58%	4.08%	6.04%	5.26%
	Claude-3-Opus	4.17%	1.87%	2.71%	2.04%	2.61%	2.42%
Screenshot + A11y tree	CogAgent	4.17%	0.85%	2.71%	0.62%	0.09%	1.32%
	Gemini-ProV	4.17%	4.43%	6.55%	0.00%	1.52%	3.48%
	GPT-4V	16.66%	6.99%	24.50%	18.37%	4.64%	12.17%
	Claude-3-Opus	12.5%	3.57%	5.27%	8.16%	1.00%	4.41%
Set-of-Mark	CogAgent	4.17%	0.00%	2.71%	0.00%	0.53%	0.99%
	Gemini-ProV	4.17%	1.01%	1.42%	0.00%	0.63%	1.06%
	GPT-4V	8.33%	8.55%	22.84%	14.28%	6.57%	11.77%
	Claude-3-Opus	12.5%	2.72%	14.24%	6.12%	4.49%	6.72%
Human Performance		75.00%	71.79%	70.51%	73.47%	73.27%	72.36%

generate the action, but this resulted in poor performance (success rate of 2.79% under pure-screenshot setting). We attribute the result to a lack of history encoding and change in the prompting scheme. Therefore, in the experiments, we opt to utilize the context window by providing the most recent 3 observations and actions in chat mode, *i.e.*, alternating between “user” prompts and “assistant” prompts, instead of the (observation, action) pairs. We use a temperature of 1.0 and top-p of 0.9 and truncate from the beginning of the input if still exceeding the max tokens limit required by the models. The prompts used in the experiments are provided in App.C.1. We heuristically request the agents to complete the tasks within a max step limit of 15, which is enough for most tasks. We present a summary of the results in Tab. 5 and analysis in Sec. 4.2. We implement the following four types of input settings on LLM and VLM.

Accessibility tree We aim to evaluate whether the current advanced text-based language models can reason and ground themselves in the context to generate the correct action. Since the original XML format of accessibility tree contains millions of tokens, caused by countless elements, redundant attributes, and a mass of markups, we opt to filter out non-essential elements and attributes and represent the elements in a more compact tab-separated table format. To be specific, we filter the elements by their tag, visibility, availability, existence of text or image contents, *etc.* The detailed filtering method is elaborated on in App. C.3. Only the *tag*, *name*, *text*, *position*, and *size* of the remaining elements are kept and concatenated by tab character in the input. As the raw coordinates are provided within the accessibility tree, the LLM is required to ground its action predictions to accurate coordinates.

Screenshot This is the input format that is closest to what humans perceive. Without special processing, the raw screenshot of the virtual machine is directly sent to the VLM. The VLM is to understand the screenshot and predict correct actions with precise coordinates. The raw resolution of the screen is set to 1920×1080 . In order to investigate the impact of input resolution, ablation studies are also conducted with different resolutions by manually downsampling the screenshot.

Screenshot + accessibility tree To check if a combination with the accessibility tree can improve the capacity of VLM for spatial grounding, we take this setting by inputting both raw screenshots and a simplified accessibility tree.

Set-of-Marks Set-of-Marks (SoM) [52] is an effective method for enhancing the grounding capabilities of VLMs such as GPT-4V, by segmenting the input image into different sections and marking them with annotations like alphanumerics, masks, or boxes. We leverage the information from the filtered accessibility tree and mark the elements on the screenshot with a numbered bounding box. Following VisualWebArena [21] and UFO [55], we further combine the annotated screenshot with the text metadata from accessibility tree, including the *index*, *tag*, *name*, and *text* of the elements⁵. Instead of predicting precise coordinates, the VLM is supposed to specify the action object by its number index, which will be mapped into our action space by post-processing. Ablation studies are also conducted with different resolutions for SoM setting.

4.2 Results

LLMs and VLMs are still far from being digital agents on real computers. The results from Table 5 show that when only using screenshots as input and adopting pyautogui as the code space, the success rate of the model is only 5.26% to 5.80% even with the strongest VLMs GPT-4V and Gemini-Pro-vision. Meanwhile, the most advanced batch of language models, when using the ally tree as input, has a success rate ranging from 2.37% to 12.24%. Overall, these figures of performance are significantly lower than the human-level performance which is 72.36% overall for individuals not familiar with the software. These gaps indicate that current LLMs and VLMs may still have a significant gap from humans in performance, necessitating further research in this area. Another surprising finding is that although Claude-3 Opus is reported to be competitive with GPT-4V on common benchmarks [2], it falls far behind when used as a digital agent in OSWORLD. We will present a qualitative analysis and infer reasons in Sec. 5.4.

Agent performance has much higher variance than human across different types of computer tasks. OSWORLD is capable of simulating and evaluating the various software types and combination scenarios involved in people’s daily lives in an open-ended manner. We observe performance based on software type grouping and find that agents based on LLMs show significant differences across different subsets. As shown in Table 5, performance tends to be better in tasks oriented towards CLI interfaces (such as OS-type tasks) compared to those based on GUI (such as Office tasks involving clicks on spreadsheet interfaces and document processing). Moreover, the biases between different models and settings are inconsistent, with gaps even exceeding 20%; another point is that performance on workflow-type tasks involving multiple software is far below the figures on a single software, generally below 5%. However, human performance is consistent across these tasks, fluctuating around 70% without exceeding a 5% variance, forming a significant contrast with the models. This suggests that the way humans understand and complete tasks may differ significantly from the current logic and methods based on LLMs and VLMs.

Ally tree and SoM’s effectiveness varies by models. The ally tree contains some attribute information of visible elements, including window position and size, as well as some semantic labels of the window. The performance gap illustrated in Table 5 between GPT-4V and Claude-3 with additional ally tree information and under a pure screenshot setup suggests that it still has significant room for improvement in accurately perceiving and reasoning GUI elements. Conclusions are reversed for Gemini-Pro.

While applying SoM setting, there is a decline for GPT-4V in performance compared to directly providing the model with screenshots and ally tree inputs, which contradicts the widely shown effectiveness of SoM in classic image understanding tasks [52], as well as in application areas like web agents [61, 15]. We speculate that this is due to the tasks performed within operating systems having higher resolution and much more elements, (e.g., the cells in a spread table), leading to a significant amount of noise that counteracts the auxiliary role of bounding boxes. Some tasks also require a detailed operation on the coordinate level, which cannot be modeled by the bounding box that SoM marks.

VLM agents with screenshot-only setting show lower performance, but it should be the ultimate configuration in the long run. The setting that relies solely on screenshots exhibits the lowest performance, at only 5.26%, among all. Surprisingly, it still achieves a decent outcome when

⁵This metadata is similar to but kind of different from that provided in the single ally tree setting. To be specific, the coordinates and size are replaced with element index.

managing workflow tasks (involving multiple applications) that involve multiple applications. Despite the performance, it is worth mentioning that this is the only configuration that does not require additional information, such as an accessibility (a11y) tree, making it concise, intuitive, and in alignment with human perception since the a11y tree may not be well-supported across all software or cannot be obtained under noisy conditions (*e.g.*, when the agent is restricted to viewing the computer through peripheral screens), and the massive amount of tokens contained in the a11y tree (even just the leaf nodes can have tens of thousands of tokens) can also impose an additional inference burden on the model. Future work on purely vision-based agents could lead to stronger generalization capabilities and, ultimately, the potential for integration with the physical world on a larger scale.

5 Analysis

In this section, we aim to delve into the factors influencing the performance of VLMs in digital agent tasks and their underlying behavioral logic. We will investigate the impact of task attributes (such as difficulty, feasibility, visual requirement, and GUI complexity), input measurements (such as screenshot resolution, the influence of trajectory history, and the effect of UI layout), explore whether there are patterns in the agent’s performance across different operating systems, and make a qualitative analysis in the aspect of models, methods, and humans. All experiments, unless specifically mentioned otherwise, are conducted using GPT-4V under the Set-of-Mark setting. Some takeaways from the analysis are: 1) higher screenshot resolution typically leads to improved performance; 2) encoding more a11y (text) trajectory history can boost performance, while not working for screenshots (image); 3) current VLMs are not adept at image-based trajectory history context; 4) current VLM agents are not robust to UI layout and noise; 5) the performance of VLM agents across OS is in strong correlation; 6) VLM agents have common error types like mouse-clicking inaccuracies, limited domain knowledge, and more types discussed in Sec. 5.4.

5.1 Performance by Task Difficulty, Feasibility and App Involved

We analyze the success rate across several additional subsets of tasks, as summarized in Tab. 6 and will be discussed in the following sections.

Task difficulty We categorize the tasks based on the time required for human completion into three groups: 0~60s (Easy), 60s~180s (Medium), and greater than 180 seconds (Hard), as an indicator of difficulty. Across these groups, the model’s success rate drops as the required time increases, with tasks taking longer than 180 seconds becoming almost impossible to complete (considering we have infeasible examples for agent’s luckiness), whereas human performance across these three groups is 84.91%, 81.08% and 49.57%, showing a slight decline of the same trend but not to the extent of being unachievable.

Table 6: Success rate (SR) of GPT-4V (SoM) across different types of tasks.

Task Subset	% of Total	SR (↑)
Easy	28.72%	16.78%
Medium	40.11%	13.12%
Hard	30.17%	4.59%
Infeasible	8.13%	16.67%
Feasible	91.87%	13.34%
Single-App	72.63%	13.74%
Multi-App Workflow	27.37%	6.57%

Feasibility We also divide tasks into groups of tasks infeasible (*e.g.*, deprecated features or hallucinated features) and tasks feasible, which requires the agents to have the ability to judge based on their own knowledge and exploration results. As shown in Tab. 6, we observe that agents currently perform slightly better in terms of infeasibility (16.67% to 13.34%), but overall, they are at a relatively low level. It is noteworthy that we also observe in some methods and settings (such as under the pure screenshot setting with the Gemini-Pro model), agents tend to easily output FAIL and refuse to continue trying. This situation leads to some false positives in infeasible tasks. The focus needs to be on improving overall performance.

Number of apps involved We also examined the performance based on whether the task involved apps software or within a single app. As shown in Tab. 6, the average performance for tasks involving a single app is low, at 13.74%, but still more than double the 6.57% observed for subsets of tasks involving workflows across multiple apps. Within single-app scenarios, tasks involving GUI-intensive Office apps generally performed the worst, with subsets such as LibreOffice Calc often scoring

zero (we show more detailed results in App. C.5). These findings highlight the need for improved collaboration capabilities between software and enhanced proficiency in specific scenarios.

5.2 Performance by Multimodal Observation Variances

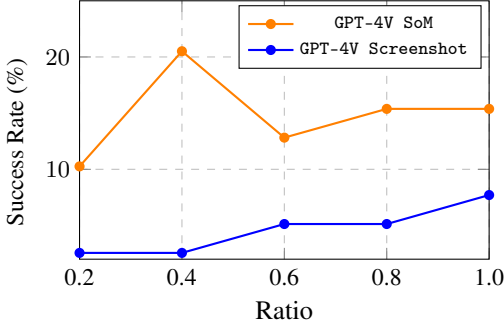


Figure 5: The effect of downsampling on the screenshot on performance with down-sampling ratios of 0.2, 0.4, 0.6 and 0.8 and run on a subset (10%) of examples.

with enhanced performance. This issue may arise from the discrepancy between the resolution of the screenshot and the coordinates of the output. However, the scenario slightly differs on SoM. Interestingly, a reduction in resolution to 768×432 (down-sampling ratio of 0.4) leads to an improvement in the agent’s performance and further diminishing the resolution even more to a down-sampling ratio of 0.2 results in a noticeable decline in performance.

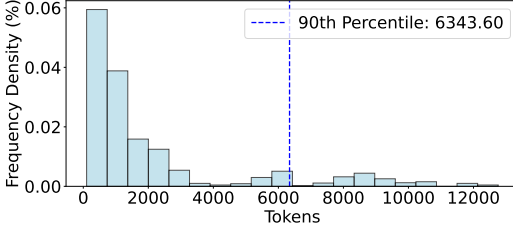


Figure 6: The length distribution of `ally` tree as observation from sampled trajectories.

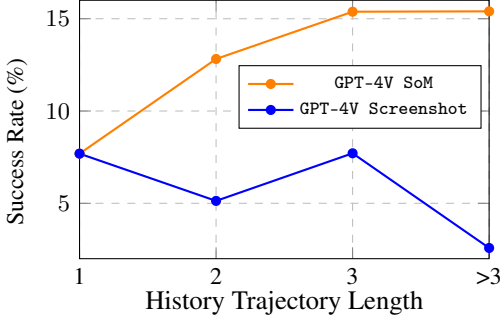


Figure 7: The effect of length of history on performance with the history encoding length of 1, 2, 3, and > 3 and run on a subset (10%) of examples.

for longer context support and understanding reasoning, improving model efficiency, and designing new agent architectures for efficient memory storage will have a significant impact on digital agents.

Higher screenshot resolution typically leads to improved performance

Despite the significant progress in display technology (1080P, 2K, and 4K), most VLMs are still trained on data far below these resolutions. We select the screenshot-only input and SoM setting to test the method’s performance under different screen input down-sampling ratios (i.e., 0.2, 0.4, 0.6 and 0.8 of the original resolution), to evaluate the impact of resolution changes on model recognition ability and accuracy. The output coordinates of the model for the screenshot setting are still expected to align with the original resolution (i.e., 1080P). The effects of varying input resolutions on performance are shown in Figure 5. For inputs based on pure screenshots, it is observed that an increase in resolution directly correlates

Longer text-based trajectory history context improves performance, unlike screenshot-only history, but poses efficiency challenges

The main experiment revealed the decisive role of the `ally` tree in performance within the current technological context. Even when we retain key attribute elements based on heuristic rules (keep nodes with tags of the document, item, button, heading, label, etc.), LLMs still require a sufficiently large context to process this information effectively. To further understand this, we sample some `ally` tree observations from OSWORLD and conducted the statistical analysis, as shown in Figure 6. The analysis indicates that a context length of 6000 is needed to accommodate about 90% of cases for a single observation. However, relying solely on current observations inherently leads to agents making repeated errors. Therefore, we include current observations as well as past N rounds of observations and actions in the constructed prompts (see appendix for more details), to explore the impact on agent performance when N is set to 1, 2, 3, and all where we put as much context as we can. The experimental results (as shown in Figure 7) show the performance increase with more history context for SoM. Future work on constructing models with enhanced capabilities

However, we also note that the inclusion of additional trajectory history does not enhance performance under the pure screenshot setting. This suggests that contemporary advanced VLMs might not be as adept at extracting robust contextual information from images as they are from textual data. Strengthening this capability to harness information from images constitutes an important avenue for future enhancements.

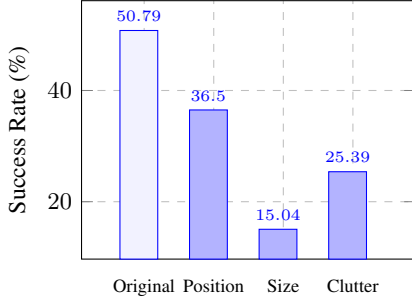


Figure 8: Decline in performance due to window perturbations.

the window as an intermediate step and are stuck on other things. This suggests that while agents possess some capability to navigate between windows, they lack a comprehensive strategy for managing window states effectively.

VLM agents struggle with perturbation of position and size of application windows and irrelevant information

We continue to adopt the SoM setting and sample a subset of 28 tasks that agents relatively well perform (with a success rate of 50.79%) in OSWORLD. At the beginning of each task, we introduce disturbances to the windows by 1) changing the *position* of the window; 2) changing the *size* of the window to the minimal; 3) opening some irrelevant software and maximizing them to *clutter* the screen. This process generates several times more samples from the subset of tasks to observe their performance. We find current agents are not robust in handling all these changes, which leads to a performance drop to over 60% to even 80%. Surprisingly, we find agents can switch the window to a certain degree but fail to maximize the

5.3 Performance across Different Operating Systems

Another key challenge in building universal digital agents is ensuring that these agents can maintain efficient and consistent performance across different operating system environments. The differences between OS and their software ecosystems can significantly impact an agent’s observation and action spaces, leading to performance uncertainties. Here, we explore and analyze the correlation between the success of agents in completing tasks on Windows after migrating from Ubuntu using examples from OSWORLD.

We enhance the functionality of the OSWORLD environment to support setting up initial experiment states, final evaluations, and obtaining observations such as the ally tree and screenshots in Windows OS. Additionally, we have made example-wise fine-tuning modifications to the existing subset in OSWORLD for migration to Windows. We conduct evaluations using the GPT-4V screenshot-only method and present the correlation of performance across the two operating systems. As shown in Tab. 7, the model’s performance on Ubuntu and Windows is 4.88% and 2.55%, respectively, with a correlation coefficient of 0.7, despite the differences in their observation spaces. This implies that insights and methodologies developed within the OSWORLD framework can be effectively transferred to Windows environments with a high degree of reliability.

Table 7: Comparison of model performance and correlation across operating systems.

OS	SR (%)	Correlation Coefficient
Ubuntu	4.88	0.7
Windows	2.55	

5.4 Qualitative Analysis

In this section we highlight representative examples of success, failure, and surprising outcomes, alongside a comparative study between GPT-4V and Claude-3 agents, to elucidate the unique challenges and insights our environment introduces. See App. D for more details.

Success and failure cases We find agents, particularly based on GPT-4V, can successfully solve tasks that involve complex problem-solving or creative thinking, showcasing the advanced understanding and processing capabilities of the model already. One successful task is shown in the first row of Figure 9. The agent is requested to extract subtitle files from the video stream and save them locally. The agent first divides the screen into two parts, with the VLC application window on the

left and the terminal window open on the right, and uses the `ffmpeg` command twice. The first use removes the subtitles embedded in the original video, and the second use saves the extracted subtitles locally.

Task Instruction: I downloaded an episode of Friends to practice listening, but I don't know how to remove the subtitles. Please help me remove the subtitles from the video and export it as "subtitles.srt" and store it in the same directory as the video.

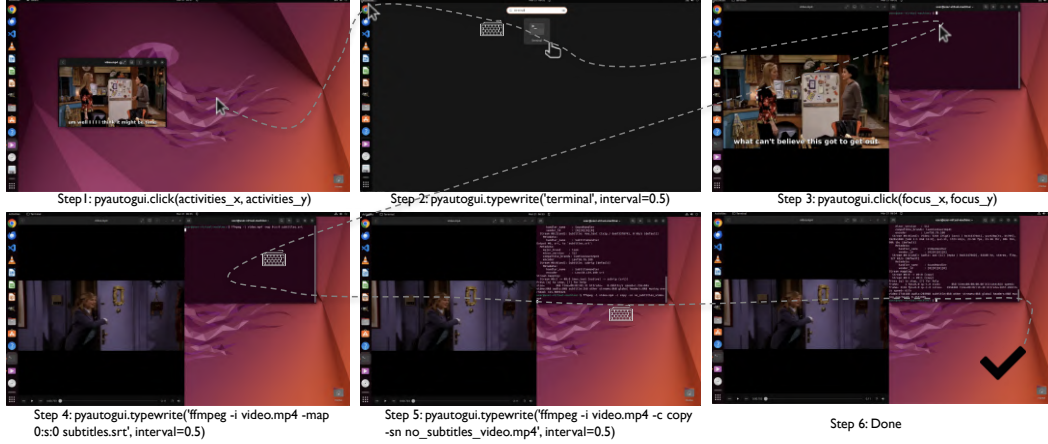


Figure 9: The agent successfully understood the complex task instructions, extracted the subtitle file from the video, and generated a pure video without embedded subtitles.

Despite the successes, there are notable failures that highlight the limitations of current models. In the task of “center-aligning the title of the document” (Fig. 10 line 1), the agent fails to ground the relatively simple requirement of “center alignment of texts”, performing many useless actions such as selecting irrelevant words, opening irrelevant menus, *etc.*

Moreover, we find that the agent lacks prior knowledge in using software, performing poorly in many specialized tasks (as shown in Fig. 16, with GIMP, LibreOffice Calc, and Chrome selected). Taking GIMP as an example, for the instruction “reduce brightness” the agent does not know which menu in the toolbar is for brightness adjustment and instead randomly tries until exhausting the maximum number of steps.

Common errors by GPT-4V agents Among the 550 failed examples from different settings in our sample, more than 75% exist *mouse click inaccuracies*, which is the most common error. The agent fails to click the correct coordinates despite planning detailed and accurate steps in their code comments, indicating strong planning but weak execution capabilities. Mouse click inaccuracies lead to two other frequent errors: *repetitive clicks* and *environmental noise dilemma*. Repetitive clicks occur when the agent repeatedly misclicks, adjusts, and fails, consuming too many steps. Environmental noise arises from clicking unintended objects, causing pop-ups, or opening unrelated applications. Due to a lack of prior knowledge about most professional software, it falls into a mismatch dilemma between the actions taken and the current state, and don’t know how to get back to normal. Moreover, the agent lacks basic human-like cognition of web pages, such as not closing pop-ups in real-world web pages or being attracted by advertisement content, which affects its original correct judgment. Failures also arise from *misinterpretation of instructions* and *visual oversight*, highlighting the need for improvement in language and visual processing. See App. D.2 for the specific execution process.

Discrepancies in task difficulty between agent and human We identify notable disparities in the perceived difficulty of tasks between humans and AI agents. Tasks that are intuitively simple for humans often present substantial challenges to agents, and conversely, tasks that humans find demanding can be more straightforward for agents to execute. You can find more details in Fig. 19 and App. D.3.

Tasks where humans outperform agents These tasks mainly involve text-based and design-related work, such as “bold the font on this slide and add notes” or “erase all the highlighted marks in this document” (Fig. 10 Line 2). Since the Internet lacks such fine-grained data as the software

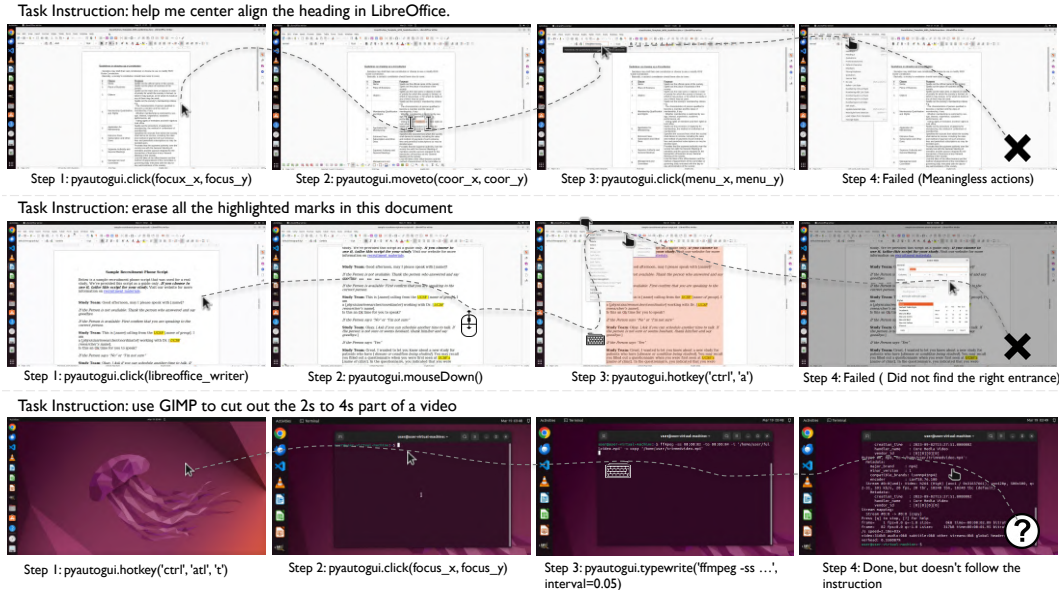


Figure 10: Screenshots of the three examples mentioned in the quality analysis. The first line is an example of GPT-4V failing at a very simple task, the second line is one example where agents face more difficulty than humans, and the third line is one example that is more difficult for humans than for agents.

execution process, the agent also lacks the corresponding training process, so its grounding ability is not good enough. The lack of understanding of GUI logic also causes poor performance on operations like selecting and scrolling.

Tasks where agents outperform humans Tasks that the agent considers simple but humans find difficult are concentrated in “code solvability tasks”, such as “monitor the system CPU for 30s and output the results” and “force close a process”. These tasks require little or no GUI interaction and can be completed by executing complex codes and instructions. It’s worth noting that completing through code sometimes mismatches with human instructions. In the task “use GIMP to cut out the 2s to 4s part of a video,(Fig. 10 Line 3)” the agent used “ffmpeg” command to complete the video cropping, ignoring the “use GIMP” requirement in the instructions.

Surprisingly, we discovered that agents are as prone to inefficiency in mechanically repetitive tasks, such as copying, pasting, and batch editing of Excel sheets, as humans. Humans frequently commit careless errors during execution. The shortcomings in agents stem either from the absence of an API or from insufficient training data related to the API, hindering their ability to efficiently process tasks in batches. Furthermore, sluggish response times can cause tasks to either time out or surpass the maximum allowed steps.

Comparative analysis: Claude-3 vs. GPT-4V Although Claude outperforms GPT-4 in many benchmarks such as GSM8K, HumanEval, *etc.*, in our main experiment, we find that Claude has an average lower accuracy rate compared to GPT-4V by 2.84% to 7.76%. We find that Claude can provide satisfactory high-level solutions, but its grounding ability contains hallucinations in detail. For instance, Claude would interpret double-clicking a file as selecting it instead of opening it, treat column B in LibreOffice Calc software as column C, and enter text in the VS Code text replacement box without clicking on global replace. This shows that Claude can align well with human planning in problem-solving, but lacks excellent grounding ability when it comes to execution. Details can be seen in Fig. 20 and App. D.4.

6 Related Work

Benchmarks for multimodal agents Testing digital interaction agents mainly spans coding environments, web scenarios, and mobile applications. In the coding domain, several works provide

frameworks and datasets for evaluating agents across programming languages and software engineering activities [53, 19, 23, 42]. For web browsing, platforms have been developed for agents to interact with web interfaces through keyboard and mouse actions, alongside datasets focusing on open-ended web tasks and realistic web navigation [41, 29, 54, 9, 62, 21]. Mobile device interaction research aims at improving accessibility, with simulators for mobile UI interactions and platforms dedicated to InfoUI tasks [26, 44, 47, 46, 38, 57, 49, 56, 48]. Further, environments connecting to real computers and datasets for GUI grounding, albeit without interactive capability, have emerged [12, 8, 36, 20]. Comprehensive task evaluation across different aspects also sees innovations [31, 34]. Differing from previous endeavors focusing on singular environments or lacking executability, OSWORLD integrates an interactive setup enabling agents to engage with operating systems openly, supported by a diverse array of tasks and precise evaluation scripts within a fully controllable setting, marking it as a competitive benchmarking realism and reliability, as well as an environment for learning and evaluating general-purpose digital agent (See Tab. 4 for comparison).

Vision-language models for multimodal agents Many existing works on GUI interaction utilize some form of structured data (such as HTML, accessibility trees, view hierarchies) as a grounding source [9, 14, 26, 35, 60, 43, 58, 62]. However, source code often tends to be verbose, non-intuitive, and filled with noise. In many cases, it is even inaccessible or unavailable for use, making multi-modality or even vision-only perception a must. To take screenshots as input, there are already specialized, optimized multi-modal models available that are suited for tasks on web [4, 11, 17, 22, 40] and mobile devices [16, 59]. Additionally, general-purpose foundation models [5, 25, 30, 63] also demonstrate significant potential for multi-modal digital agents. The development of prompt-based methods [12, 15, 51, 61], as well as visual reasoning paradigms, have also further facilitated the performance of digital agents in web pages, mobile apps, and desktop. To investigate how well do current models and methods perform in digital agent tasks, our paper evaluates the results of text-only, vision-only, and multi-modal input as well as across multiple methods, demonstrating that existing multi-modal models are far from capable computer agents. Specifically, there is ample room for improvement in long-horizon planning, screenshot details perception, pixel coordinate locating, and world knowledge.

7 Conclusion and Future Work

In conclusion, the introduction of OSWORLD marks a significant step forward in the development of autonomous digital agents, addressing critical gaps in existing interactive learning environments. By providing a rich, realistic setting that spans multiple operating systems, interfaces, and applications, OSWORLD not only broadens the scope of tasks digital agents can perform but also enhances their potential for real-world application. Despite the promise shown by advancements in vision-language models, evaluations within OSWORLD reveal notable challenges in agents’ abilities, particularly in GUI understanding and operational knowledge, pointing to essential areas for future research and development.

We identify several potential directions for community development and progress toward general-purpose agents for computer operation:

Enhancing VLM capabilities for efficient and robust GUI interactions For foundation model development, we need to boost the efficiency of our models, enabling them to process much longer contexts and perform inference computations efficiently, akin to the robotics community [6, 7] to better handle real-world cases. Enhancements in VLMs’ GUI grounding capabilities that are robust to application windows changes are also sought, focusing on the accurate understanding and generation of precise actions aligned with given instructions. Moreover, amplifying VLMs’ ability to comprehend context in the form of images is a pivotal goal, since it is crucial to enable history encoding using images so that we can build memory and reflection upon that. These improvements may require more efforts in the upstream pre-training stage, downstream fine-tuning stage, and even in the model structure itself, as pointed out in previous work [9, 16, 32].

Advancing agent methodologies for exploration, memory, and reflection The next-level approach encompasses designing more effective agent architectures that augment the agents’ abilities to explore autonomously and synthesize their findings. The agents face challenges in leveraging lengthy raw observation and action records. It’s fascinating to explore novel methods for encoding this history,

incorporating efficient memory and reflection solutions to condense contextual information and aid the agent in extracting key information. Additionally, integrating knowledge grounding into (V)LLM agents through memory mechanisms is a promising avenue as well. Moreover, practice GUI assistants also require features of personalization and customization. These features rely on techniques such as user profiling and retaining memories from long-term user-assistant interactions. Additionally, crafting protocols specifically for digital agents operating within GUI and CLI interfaces aims at facilitating efficient actions is also an essential thing for the feasibility of general-purpose digital agents in the mid-short term.

Addressing the safety challenges of agents in realistic environments The safety of agents is a critical issue if applying a built agent in fully realistic environments, the developed universal digital agent could potentially be used to bypass CAPTCHA systems in the future, as noted in [39]. However, due to the currently limited capabilities of agents, we have not observed any harmful and damaging behaviors during our experiments, an automatic agent has the opportunity to damage patent rights, abuse accounts, attempt to exploit software vulnerabilities to create viruses, or engage in attacks. Currently, we adopt virtual machines to make it difficult for developing digital agents to cause irreversible damage to our host machines. However, there still lacks a reliable metric to assess the safety of an agent developed in an isolated environment. The current evaluation functions mainly focus on the results closely regarding the task instructions, assess only the correctness of task completion, and pay little attention to potential unnecessary damaging actions of agents. Owing to the complexity of a complete computer environment, we didn’t work out an efficient way to detect the latent side effects of the agent. Consequently, how to assess and control potential behaviors in open and real environments through environmental constraints and agent training is an important further direction of research.

Expanding and refining data and environments for agent development In terms of datasets and environments, we can broaden the scope to cover more specialized domains, including real-sector needs in healthcare, education, industry, transportation, and personalized requirements. Efforts can be made to ensure our environment’s seamless deployment across various hardware and software settings. The variance of all y tree quality across different applications is also noticed. Although the problem is not remarkable in the applications currently included, there is no guarantee of that the application developers obey the all y convention and offer clear and meaningful descriptions for GUI elements. More intelligent approaches to filtering redundant all y tree elements and handling latently missing elements deserve careful investigation as well. We also highlight the necessity of a painless data collection method, allowing for the effortless acquisition of computer operation data and its transformation into agent capabilities.

Acknowledgements

We thank Sida Wang, Peter Shaw, Chen Henry Wu, Pengcheng Yin, Shunyu Yao, Xing Han Lu, Siva Reddy, Ruoxi Sun, Zhiyuan Zeng, and Lei Li for their helpful feedback on this work.

References

- [1] Adept. ACT-1: Transformer for Actions. <https://www.adept.ai/act>, 2022.
- [2] Anthropic. Introducing the next generation of claude. <https://www.anthropic.com/news/claude-3-family>, 2023. Accessed: 2024-03-26.
- [3] Anthropic. The claude 3 model family: Opus, sonnet, haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf, 2024.
- [4] Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Cărbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. Screenai: A vision-language model for ui and infographics understanding. *arXiv preprint arXiv:2402.04615*, 2024.
- [5] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023.

- [6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.
- [7] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.
- [8] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024.
- [9] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023.
- [10] D. Dupont. GPT-4V-Act: GPT-4 Variant for Active Learning. GitHub repository, 2023. URL <https://github.com/ddupont808/GPT-4V-Act>.
- [11] Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*, 2023.
- [12] Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchun Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, et al. Assistgui: Task-oriented desktop graphical user interface automation. *arXiv preprint arXiv:2312.13108*, 2023.
- [13] Yiduo Guo, Zekai Zhang, Yaobo Liang, Dongyan Zhao, and Duan Nan. Pptc benchmark: Evaluating large language models for powerpoint task completion. *arXiv preprint arXiv:2311.01767*, 2023.
- [14] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*, 2023.
- [15] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.
- [16] Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. *arXiv preprint arXiv:2312.08914*, 2023.
- [17] Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*, pages 9466–9482. PMLR, 2022.
- [18] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- [19] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.
- [20] Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. Omniaact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*, 2024.
- [21] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.

- [22] Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning*, pages 18893–18912. PMLR, 2023.
- [23] Bowen Li, Wenhan Wu, Ziwei Tang, Lin Shi, John Yang, Jinyang Li, Shunyu Yao, Chen Qian, Binyuan Hui, Qicheng Zhang, et al. Devbench: A comprehensive benchmark for software development. *arXiv preprint arXiv:2403.08604*, 2024.
- [24] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models. *arXiv preprint arXiv:2305.19308*, 2023.
- [25] Lei Li, Zhihui Xie, Mukai Li, Shunian Chen, Peiyi Wang, Liang Chen, Yazheng Yang, Benyou Wang, and Lingpeng Kong. Silk: Preference distillation for large visual language models. *arXiv preprint arXiv:2312.10665*, 2023.
- [26] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*, 2020.
- [27] J. C. R. Licklider. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1(1):4–11, 1960. doi: 10.1109/THFE.1960.4503259.
- [28] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. NI2bash: A corpus and semantic parser for natural language interface to the linux operating system. *arXiv preprint arXiv:1802.08979*, 2018.
- [29] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.
- [30] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023.
- [31] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.
- [32] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 2024.
- [33] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*, 2024.
- [34] Grégoire Mialon, Clémentine Fourier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.
- [35] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- [36] Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. Screenagent: A vision language model-driven computer control agent. *arXiv preprint arXiv:2402.07945*, 2024.
- [37] R OpenAI. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2:13, 2023.
- [38] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023.
- [39] Andrew Searles, Yoshimichi Nakatsuka, Ercan Ozturk, Andrew Paverd, Gene Tsudik, and Ai Enkoji. An empirical study & evaluation of modern {CAPTCHAs}. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3081–3097, 2023.

- [40] Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. *arXiv preprint arXiv:2306.00245*, 2023.
- [41] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.
- [42] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering?, 2024.
- [43] Abishek Sridhar, Robert Lo, Frank F Xu, Hao Zhu, and Shuyan Zhou. Hierarchical prompting assists large language model on web navigation. *arXiv preprint arXiv:2305.14257*, 2023.
- [44] Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. Meta-gui: Towards multi-modal conversational agents on mobile gui. *arXiv preprint arXiv:2205.11029*, 2022.
- [45] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- [46] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibi Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.
- [47] Sagar Gubbi Venkatesh, Partha Talukdar, and Srinu Narayanan. Ugif: Ui grounded instruction following. *arXiv preprint arXiv:2211.07615*, 2022.
- [48] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024.
- [49] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Empowering llm to use smartphone for intelligent task automation. *arXiv preprint arXiv:2308.15272*, 2023.
- [50] Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, Leo Z. Liu, Yiheng Xu, Hongjin Su, Dongchan Shin, Caiming Xiong, and Tao Yu. Openagents: An open platform for language agents in the wild. *CoRR*, abs/2310.10634, 2023. doi: 10.48550/ARXIV.2310.10634. URL <https://doi.org/10.48550/arXiv.2310.10634>.
- [51] An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.
- [52] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023.
- [53] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *arXiv preprint arXiv:2306.14898*, 2023.
- [54] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.
- [55] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*, 2024.
- [56] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv e-prints*, pages arXiv–2312, 2023.

- [57] Danyang Zhang, Lu Chen, and Kai Yu. Mobile-env: A universal platform for training and evaluation of mobile interaction. *arXiv preprint arXiv:2305.08144*, 2023.
- [58] Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. Large language models are semi-parametric reinforcement learning agents. *Advances in Neural Information Processing Systems*, 36, 2024.
- [59] Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *arXiv e-prints*, pages arXiv–2309, 2023.
- [60] Zihan Zhao, Lu Chen, Ruisheng Cao, Hongshen Xu, Xingyu Chen, and Kai Yu. Tie: Topological information enhanced structural reading comprehension on web pages. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1808–1821, 2022.
- [61] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.
- [62] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.
- [63] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.

A Details of OSWORLD Environment

A.1 Environment Infrastructure

As compared to core commonly used techniques like Docker⁶, virtual machines can operate their own kernel and system, enabling compatibility with a wide variety of operating systems (such as Windows, macOS, Linux, etc.) across different CPU hardware types (x64, ARM, etc.), and supports training and evaluation in a multiprocess manner on both headless servers and personal computers.

A.2 Observation Space

We implement three kinds of observation: **complete screenshot**, **accessibility tree** and **terminal output**. We also implement a video recorder of the environment but don't put it into our modeling due to the agent's ability limitations. OSWORLD supports observation refactoring and extending if needed, such as getting data from certain opening applications that we want to focus on.

A.2.1 Screenshot

To align with the perception of a human user, we capture a screenshot of the entire computer screen. Including the mouse cursor also proves helpful in certain cases where mouse information is crucial. For screen resolution, we default to 1920×1080, as it is the most commonly used screen resolution according to Internet Users Screen Resolution Realtime Statistics for 2023⁷. This resolution also offers a 16:9 aspect ratio. OSWORLD also supports modifying the resolution of virtual machines to avoid potential memorization of absolute pixel values and to assist studies on topics like generalization across different resolutions.

A.2.2 Accessibility Tree

An accessibility tree (or a11y tree, same logic to kubernetes and k8s), refers to an intricate structure generated by the browser or OS accessibility APIs that renders a representative model of the web content, providing a means of interaction for assistive technologies. Each node within the accessibility tree hosts important information about a UI element. This could range from the nature of the object (a button, checkbox, or paragraph of text), its current state (checked or unchecked, for checkboxes), and even its spatial orientation on the screen.

Different operating systems employ varied accessibility APIs and tools to construct and manipulate the accessibility tree. These include Microsoft Active Accessibility (MSAA) and User Interface Automation (UIA) for Windows, NSAccessibility Protocol and macOS Accessibility Inspector for macOS, and Assistive Technology Service Provider Interface (ATSPI)⁸ for GNOME desktop used on Ubuntu. We adopt `pyatspi` to get the accessibility tree on Ubuntu and `pywinauto` on Windows. We further convert it into XML format for message passing. Partial pieces of the XML formatted accessibility tree are shown in Figure 11.

Tools such as Accerciser can help visualize the corresponding relationship of tree nodes and GUI components in the accessibility tree as shown in Figure 12.

A.3 Action Space

We implement two kinds of action space: `pyautogui` and `computer_13`. We mainly use `pyautogui` action space, since it saves tokens for describing action space definition in prompting, compared with self-designed actions.

A.3.1 PYAUTOGUI

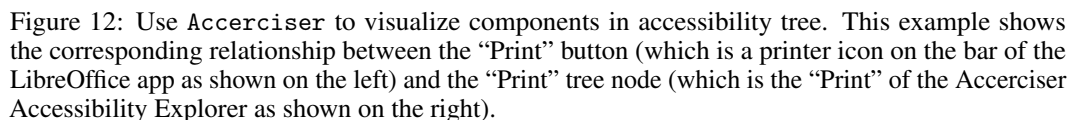
`pyautogui` is an open-source, cross-platform Python module utilized for programmatically controlling the mouse and keyboard. It can control simple movements, clicks, as well as keyboard inputs, and can provide screen capture features or locate where the screen-specific images reside that can

⁶<https://www.docker.com/>

⁷<https://www.screenresolution.org/year-2023/>

⁸<https://docs.gtk.org/atspi2/>

Figure 11: Raw XML formatted accessibility tree visualization.



Herein, we will demonstrate some use cases of `pyautogui` and illustrate how it can be wielded as an action space.

Mouse Controlling Functions

⁹<https://github.com/jordansissel/xdotool>
¹⁰<https://github.com/boppreh/mouse>
¹¹<https://github.com/boppreh/keyboard>

```

pyautogui.click(x=moveToX, y=moveToY, clicks=num_of_clicks, button='left')
pyautogui.rightClick(x=moveToX, y=moveToY)
pyautogui.middleClick(x=moveToX, y=moveToY)
pyautogui.doubleClick(x=moveToX, y=moveToY)
pyautogui.tripleClick(x=moveToX, y=moveToY)
pyautogui.scroll(amount_to_scroll, x=moveToX, y=moveToY)
pyautogui.mouseDown(x=moveToX, y=moveToY, button='left')
pyautogui.mouseUp(x=moveToX, y=moveToY, button='left')

```

Keyboard Controlling Functions

```

import pyautogui

# useful for entering text, newline is Enter
pyautogui.typewrite('Hello world!\n', interval=secs_between_keys)

pyautogui.typewrite(['a', 'b', 'c', 'left', 'backspace', 'enter', 'f1'], interval=secs_between_keys)
pyautogui.hotkey('ctrl', 'c') # ctrl-c to copy
pyautogui.hotkey('ctrl', 'v') # ctrl-v to paste
pyautogui.keyDown(key_name)
pyautogui.keyUp(key_name)

```

pyautogui as an Action Space Given the various controls it provides, pyautogui can readily be used as an action space in building automation software or testing interfaces with minor adjustments. More formally, an action is within the action space when it meets the syntax of pyautogui or is one of three special actions `WAIT`, `FAIL`, and `DONE`. This might include actions like clicking at a certain location, entering text or key sequences, or even resting for a span (Pause). Each action could be mapped to an identifying label or number, forming a discrete action space. For example:

```

import pyautogui

def perform_action(action):
    if action == 0:
        pyautogui.moveTo(100, 100)
    elif action == 1:
        pyautogui.write('Hello world!', interval=0.25)
    else:
        pyautogui.pause(1)

```

In this scheme, the "perform_action" function constitutes the action space, where each unique action is associated with a unique integer (its action ID). The function interprets these action IDs and performs the corresponding action, forming a rudimentary discrete action space.

One interesting finding is that language models generate screenshot locate functions like:

```

pyautogui.locateOnScreen('Apple.png')

```

When there is insufficient grounding evidence (such as when no screenshot is inputted, the accessibility tree lacks a specific element, or the multimodal model cannot comprehend the user interface), employing this function to retrieve the correct icon image could present an interesting method.

A.3.2 COMPUTER_13

To facilitate potential reinforcement learning applications, we have created a variant of pyautogui, which we named `computer_13`. In this variant, we wrap pyautogui into a finite action class with parameterized enumeration, such that it features 13 action types, excluding three special ones for task process control. Utilizing this structured approach allows more effective reinforcement learning by providing a distinct and finite set of actions to be learned and optimized. As summarized in Table 8, each action type has certain parameters, detailed in the collection, confirming the type, range, and whether each parameter is optional for that action.

Table 8: Action types and parameters defined in action space `computer_13`, a variance we created for the potential reinforcement learning research based on our environment.

Action Type	Parameters	Note
MOVE_TO	x, y	Move the cursor to the specified position
CLICK	<i>button</i> , x, y , <i>num_clicks</i>	Click the left button if the button not specified, otherwise click the specified button; click at the current position if x and y are not specified, otherwise click at the specified position
MOUSE_DOWN	<i>button</i>	Press the left button if the button not specified, otherwise press the specified button
MOUSE_UP	<i>button</i>	Release the left button if the button not specified, otherwise release the specified button
RIGHT_CLICK	x, y	Right click at the current position if x and y are not specified, otherwise right click at the specified position
DOUBLE_CLICK	x, y	Double click at the current position if x and y are not specified, otherwise double click at the specified position
DRAG_TO	x, y	Drag the cursor to the specified position with the left button pressed
SCROLL	dx, dy	Scroll the mouse wheel up or down
TYPING	<i>text</i>	Type the specified text
PRESS	<i>key</i>	Press the specified key and release it
KEY_DOWN	<i>key</i>	Press the specified key
KEY_UP	<i>key</i>	Release the specified key
HOTKEY	<i>keys</i>	Press the specified key combination
WAIT	-	Wait until the next action
FAIL	-	Decide the task cannot be performed
DONE	-	Decide the task is done

B Details of OSWORLD Benchmark

B.1 Operating System Selection

As of 2023, the most popular desktop operating systems are Windows (69.5%), macOS (20.4%), ChromeOS (3.2%), and Linux (3.1%)¹². While Windows and macOS dominate the market share and boast the richest software ecosystems, their closed-source nature raises potential copyright concerns for direct usage. ChromeOS, being a web-based operating system, heavily depends on a Google account for its functionalities, rendering it less suitable for a public benchmark.

In contrast, Linux desktops offer a wealth of open-source software for most daily tasks, supported by an active community for both basic and advanced use. Essential applications such as Office Suite, browsers, email clients, multimedia apps, and thousands of games and applications are either pre-installed or readily available through the software center of the distribution. Consequently, we select Ubuntu, the most representative Linux desktop OS, as the foundation for the main part of our benchmark intended for public use. Additionally, we have developed components to facilitate agent interaction on a Windows virtual machine and have created a relatively small set of examples focusing on the Microsoft Office suite, including Excel, PowerPoint, and Word. This serves as a counterpart to the LibreOffice suite available on Ubuntu. These components can be utilized in-house or officially with the purchase of a license. Regarding macOS, theoretically, it is illegal to install macOS on non-Apple devices, which leads us to refrain from developing our benchmark on this platform to avoid copyright issues.

B.2 Software Selection

Due to the high cost of obtaining operation and evaluation script annotation data, we have chosen a representative set of software for the examples of Ubuntu part. We adopt standards that consider: 1) Availability - the software must be available on Ubuntu 22.04; 2) Open-source - the software should be open-sourced with an appropriate license to prevent copyright issues; 3) Popularity - the software should take a high download number and frequency of recommendations in blogs and tutorials; 4) Strong user community and good support resources - it is preferable to have an active and robust user community as well as official documents, which can serve as ample resources for task collection and agent learning; 5) Diversity of categories - the software should be diverse to adequately represent and cover a wide range of real-world cases.

¹²<https://www.statista.com/statistics/218089/global-market-share-of-windows-7/>

As a result, we have shortlisted the software into two categories: general usage and professional usage. For general usage, we have VLC for media playback, Chrome for web browsing, and Thunderbird for email management. For professional usage, we have VS Code as a coding IDE, and LibreOffice (Calc, Writer, and Impress) for handling spreadsheets, documents, and presentations respectively, along with GIMP for image editing. This brings our total to eight different types of software.

B.3 Task Example Sources

We detail the task example sources in Table 9.

Table 9: Task Example Resources

App	Resources	Link
OS	Ubuntu Documentations	https://help.ubuntu.com/
	Ask Ubuntu	https://askubuntu.com/
	Super User	https://superuser.com/
	Stack Overflow	https://stackoverflow.com
	YouTube	https://www.youtube.com/
Calc	LibreOffice Help	https://help.libreoffice.org/
	Microsoft Tech Community	https://techcommunity.microsoft.com/
	libreofficehelp.com	https://www.libreofficehelp.com/
	Reddit r/LibreOfficeCalc	https://www.reddit.com/r/LibreOfficeCalc/
	Reddit r/Excel	https://www.reddit.com/r/Excel/
	Super User	https://superuser.com/
	Medium	https://medium.com/
	Quora	https://www.quora.com/
	YouTube	https://www.youtube.com/
	Ryan and Debi & Toren Personal Site	https://www.ryananddebi.com/
Writer	LibreOffice Help	https://help.libreoffice.org/
	LibreOffice Forum	https://ask.libreoffice.org/
	libreofficehelp.com	https://www.libreofficehelp.com/
	Super User	https://superuser.com/
	Stack Overflow	https://stackoverflow.com
	Ask Ubuntu	https://askubuntu.com/
	Quora	https://www.quora.com/
	YouTube	https://www.youtube.com/
	SearchStar Personal Site	https://seekstar.github.io/
Impress	LibreOffice Help	https://help.libreoffice.org/
	libreofficehelp.com	https://www.libreofficehelp.com/
	Reddit r/LibreOffice	https://www.reddit.com/r/LibreOffice/
	Super User	https://superuser.com/
	Stack Overflow	https://stackoverflow.com
	Technical Tips	https://technical-tips.com/
	Just Click Here	https://justclickhere.co.uk/
	TikTok	https://www.tiktok.com/
VLC	VLC Documentation	https://docs.videolan.me
	VLCHelp.com	https://www.vlchelp.com/
	VideoLAN's Wiki	https://wiki.videolan.org/
	Ubuntu Documentations	https://help.ubuntu.com/
	Reddit r/Fedora	https://www.reddit.com/r/Fedora/
	Super User	https://superuser.com/
	Medium	https://medium.com/
	YouTube	https://www.youtube.com/
	Dedoimedo	https://www.dedoimedo.com/index.html
Thunderbird	Thunderbird Support	https://support.mozilla.org/en-US/products/thunderbird
	Reddit r/Thunderbird	https://www.reddit.com/r/Thunderbird/
	Reddit r/Automation	https://www.reddit.com/r/automation/
	Super User	https://superuser.com/
	WikiHow	https://www.wikihow.com/
	Quora	https://www.quora.com/
	BitRecover	https://www.bitrecover.com/
	AdSigner	https://www.adsigner.com/
Chrome	Google Chrome Help	https://support.google.com/chrome

Continued on next page

Table 9 – continued from previous page

App	Resources
	Reddit r/Chrome https://www.reddit.com/r/Chrome/
	Super User https://superuser.com/
	WikiHow https://www.wikihow.com/
	in5steps.com https://in5stepstutorials.com/
	How-To Geek https://www.howtogeek.com/
	Medium https://medium.com/
	Quora https://www.quora.com/
	YouTube https://www.youtube.com/
VS Code	Laptop Mag https://www.laptopmag.com
	Super User https://superuser.com/
	Stack Overflow https://stackoverflow.com
	Quora https://www.quora.com/
	YouTube https://www.youtube.com/
GIMP	Campbell Muscle Lab GitHub https://campbell-muscle-lab.github.io/
	Reddit r/GIMP https://www.reddit.com/r/GIMP/
	Super User https://superuser.com/
	Stack Overflow https://stackoverflow.com
	Quora https://www.quora.com/
	Make-Use-Of https://www.makeuseof.com/
Workflow	YouTube https://www.youtube.com/
	UniPath Marketplace https://marketplace.uipath.com/
	sync.blue https://www.sync.blue/
	Device Tests https://devicetests.com/
	Make Tech Easier https://www.maketecheasier.com/
	Unix & Linux Stack Exchange https://unix.stackexchange.com/
	Geeks for Geeks https://www.geeksforgeeks.org/
	I Love Free Software https://www.ilovefreesoftware.com/
	The Geek Diary https://www.thegeekdiary.com/
	Zyxware https://www.zyxware.com/
	GNOME Discourse https://discourse.gnome.org/
	It's FOSS https://itsfoss.com/
	Super User https://superuser.com/
	Stack Overflow https://stackoverflow.com
	LibreOffice Forum https://ask.libreoffice.org/
	ImpressExtractNotes https://github.com/danielrcollins1/ImpressExtractNotes
	Medium https://medium.com/
	YouTube https://www.youtube.com/
	Kelvin Smith Library https://case.edu/library/

B.4 Task Examples Collection

Here we show the detailed statistics of OSWORLD benchmark, including the main set on Ubuntu (369 in total) and the analytic set on Windows (43 in total).

Table 10: Detailed statistics of OSWORLD benchmark suite about examples number, average instruction tokens, infeasible instructions and integrated instructions.

	OS	Calc	Impress	Writer	VLC	Thunderbird	Chrome	VSCode	GIMP	Workflow	Overall
Examples	24	47	47	23	17	15	46	23	26	101	369
Avg. Inst. Tokens	22.38	33.30	25.19	35.30	35.82	34.07	22.07	20.78	16.23	51.24	33.36
#Infeasible	5	1	0	1	3	1	3	5	10	1	30
#Integrated	7	19	30	0	0	0	26	0	0	2	84

Table 11: Detailed statistics of Windows analytic set benchmark suite. This set contains no infeasible tasks and integrated tasks.

	Excel	Word	PPT	Workflow	Overall
Examples	11	9	7	16	43
Avg. Inst. Tokens	19.45	21.44	21.86	47.57	32.48

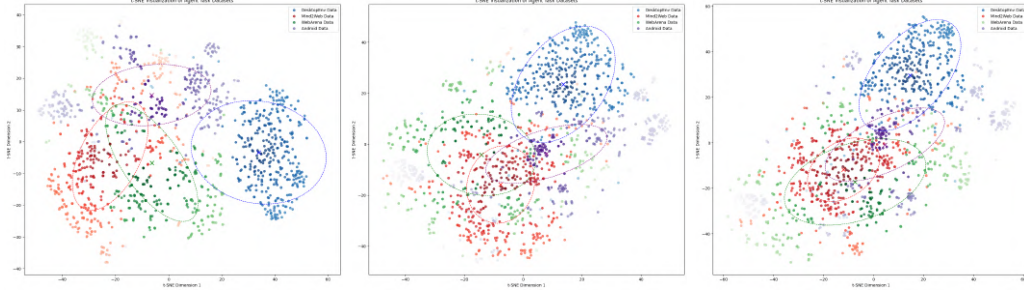


Figure 13: Comparison of instructions distribution. All datasets are sampled to 300 to make a fair comparison. The hyper-parameters of t-SNE are randomly sampled for each plot.

We also visualize the intent distribution (We obtain sentence embeddings for instructions using OpenAI’s embedding model, and then apply t-SNE to reduce the dimensionality to two dimensions for visualization.) and compare it with other benchmarks which also focus on the digital agent. We randomly sample 300 examples from each dataset and randomly choose three different hyperparameters for t-SNE. Visualization results are shown in Figure 13. From the figure, we can observe that the semantic distribution of the instructions alone has reached the most comprehensive level. Additionally, our environment remains controllable and executable, offering a more reliable evaluation. It is also noticeable that the clustering centers of the other three are closely positioned, whereas the points in our distribution approaches are inconsistent with theirs, indicating that we can serve as a unique choice for a more comprehensive assessment of the capabilities of future intelligent agents.

B.5 Initial State Setup Details

The setup of the initial state contains three stages: 1) Start emulator. The specified virtual machine is activated and automatically reverted to the corresponding snapshot, which records the initial system settings of the machine. 2) Prepare files (Optional). The file or software that specifies the initial state of the task to be executed is downloaded to the virtual machine and opened. The system is configured to first download the files to the host through a direct link and then upload them to the VM via a LAN connection. Specifically, some initial files are set up for OS-related tasks by manipulating the file system directly from the command line. 3) Execute reprocessing commands (Optional). For tasks that require additional preprocessing, task-specific operations are executed after the completion of the first two phases. For example, taking the currently open LibreOffice Impress file to page five, clicking in the center of the screen to return to the main interface, *etc.* We provide convenient APIs to configure initial conditions and world settings, standardizing our tasks to make this process user-friendly and easily extendable.

B.6 Evaluation Configuration Details

In this section, we will show details of preparations for the evaluation of the selected apps (LibreOffice – Calc, Writer and Impress, Thunderbird, VLC Media Player, Chrome, VS Code, GIMP) and OS (Ubuntu and Windows).

B.6.1 Ubuntu

LibreOffice: Calc, Writer, and Impress LibreOffice is a popular open-source fully-featured office suite for Linux desktops. Our benchmark is built upon version 7.3.7.2, the version pre-installed in Ubuntu 22.04. Calc, Writer, and Impress are selected to build tasks on them. As the majority of tasks are to conduct a little revision to a file, we evaluate these tasks mainly by checking the final result file (in *xlsx*, *docx*, or *pptx* format). The check can be done by comparing the result file with a golden reference, or inspecting some particular attributes or settings of the file, *e.g.*, page style, freezing, and locale. Usually, the *xlsx*, *docx*, and *pptx* files are mainly accessed through `openpyxl`¹³, `python-docx`¹⁴, and `python-pptx`¹⁵. For some properties not supported by the current libraries, we also look them up directly via parsing the Office Open XML format¹⁶.

Thunderbird Thunderbird is a popular open-source fully-featured email client for Linux desktops. Version 115.6.0 of Thunderbird is pre-installed in Ubuntu 22.04. We crafted an account profile to set up a feasible initial state. Evaluation for Thunderbird configurations is mainly performed by reading various configurations or data

¹³<https://openpyxl.readthedocs.io/en/stable/>

¹⁴<https://github.com/python-openxml/python-docx>

¹⁵<https://github.com/scanny/python-pptx>

¹⁶<https://learn.microsoft.com/en-us/office/open-xml/about-the-open-xml-sdk>

files in the profile folder. An open-source reverse engineering tool Firefox Decrypt¹⁷ is adopted to decrypt the stored account information for evaluation. The involved account information is just for examples and contains no information about the real person. Besides, there are tasks instructing to help to compose a new email. In these cases, the accessibility tree is leveraged to inspect the contents in the composing window before really sending it.

VLC Media Player VLC Media Player is a popular open-source cross-platform multimedia player and framework that plays most multimedia files. The evaluation for VLC Media Player is multifold, ranging from utilizing VLC HTTP interface¹⁸, reading the VLC configuration file, comparing final result files, and leveraging accessibility tree to inspect the desired content.

Chrome Google Chrome is one of the most popular and powerful cross-platform web browsers developed on Google's free and open-source software project Chromium. The evaluation of Chrome is mainly based on the utilization of Playwright¹⁹, a browser automation library to control Chromium, Firefox, and WebKit with a single API. To connect Playwright running on host machine with Chrome running on virtual machine, port transferring tool socat²⁰ is leveraged. Additional information such as the HTML source codes of websites is also leveraged in the evaluation of some tasks.

VS Code VS Code is a popular open-source multi-functional cross-platform editor for source-code editing. The evaluation of VS Code tasks is primarily divided into two different categories. One subset of tasks is predominantly oriented towards file manipulation. In the context of these tasks, a comparative analysis is conducted between the resultant file and an anticipated reference gold file. Another subset of tasks is centered around how to utilize the intrinsic functionalities of the VS Code software itself, such as modifying color themes, initiating workspace sessions, and modifying settings. In these instances, it becomes important to extract relevant internal information and configurations from the VS Code environment.

In the course of this research, we principally leverage the capabilities offered by the VS Code Extension API²¹ and information in the settings JSON file²² to obtain the requisite internal signal for the evaluation process. Our methodology involves the development of a custom VS Code extension, its installation within the VS Code software deployed on our virtual machine, and the subsequent invocation of the extension's command each time an evaluation is required, as well as checking whether the settings JSON has the correct value for a specific key.

GIMP GIMP is an open-source raster graphics editor used for image manipulation, editing, free-form drawing, format transcoding, and more specialized tasks. The evaluation for GIMP tasks is also mainly divided into two different categories, just like the VS Code evaluation. One type of task is mainly oriented to file operations. In these tasks, the resulting files are compared and analyzed with the expected reference golden files, mainly relying on some powerful image processing libraries such as pillow²³. Another category of tasks revolves around taking advantage of the inherent capabilities of the GIMP software itself. In these instances, we primarily read GIMP's configuration files to obtain internal information to evaluate the tasks.

B.7 Windows

Microsoft Office: Excel, Word, and PowerPoint Microsoft Office is the most popular office suite on Windows desktops. These three components share the same functions with the corresponding LibreOffice components by and large. They are used to edit xlsx, docx, and pptx files, respectively. Thus, the evaluation for LibreOffice tasks can be reused for Microsoft Office tasks.

Thunderbird Thunderbird is a cross-platform email client. Only the structure of profile folder on Windows is slightly different from that on Linux. We thus revised the account profile and reuse it to set up the same initial state on Windows.

Chrome Chrome is a cross-platform web browser. To evaluate tasks on Chrome, only the port transferring tool needs to be replaced with Ncat²⁴. Other configurations and the evaluations can be shared with Linux-version tasks.

¹⁷https://github.com/unode/firefox_decrypt

¹⁸https://wiki.videolan.org/Control_VLC_via_a_browser/

¹⁹<https://playwright.dev/>

²⁰<http://www.dest-unreach.org/socat/>, <https://linux.die.net/man/1/socat>

²¹<https://code.visualstudio.com/api>

²²https://code.visualstudio.com/docs/getstarted/settings#_settingsjson

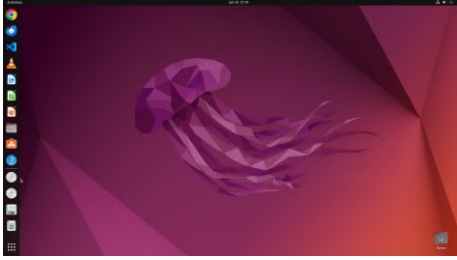
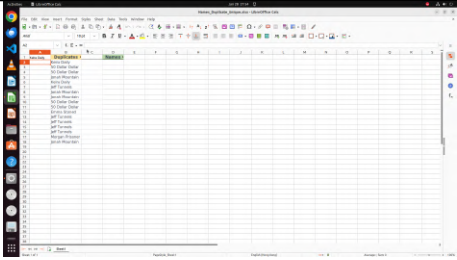
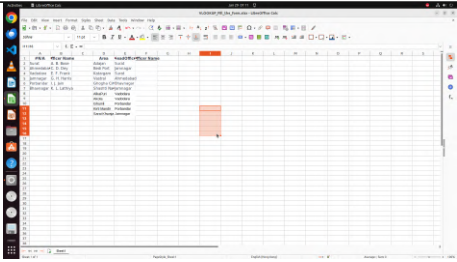
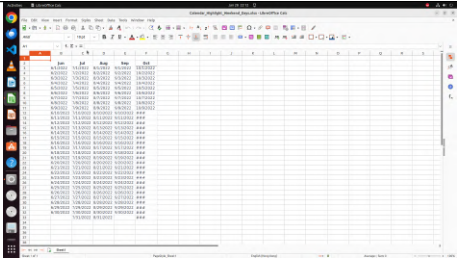
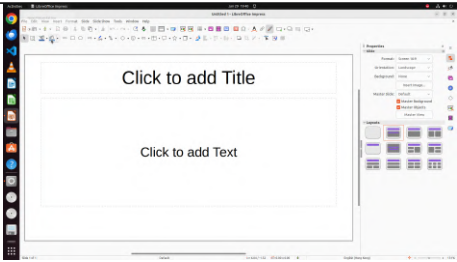
²³<https://pypi.org/project/pillow/>

²⁴<http://www.dest-unreach.org/socat/>

B.8 More Task Examples

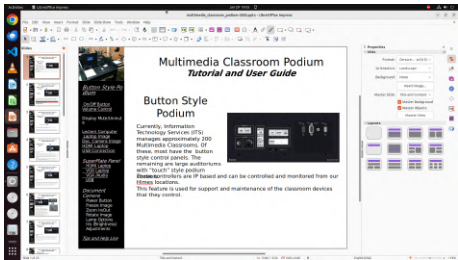
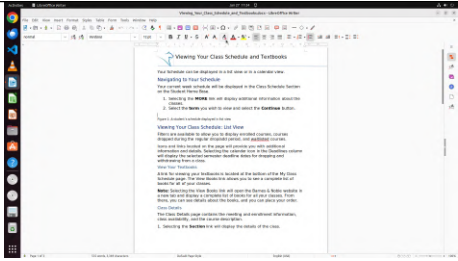
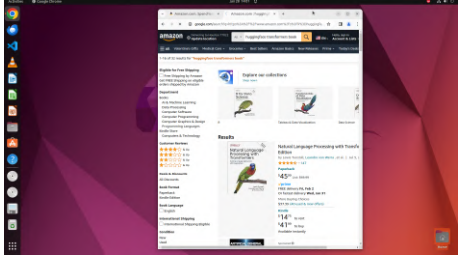
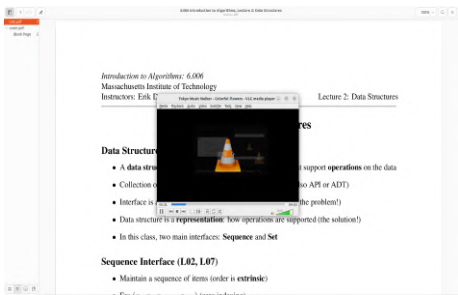
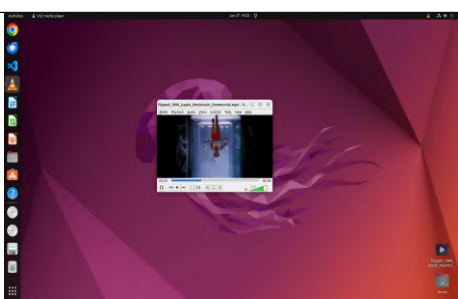
In this section, we curate a collection of examples from various app sets, each characterized by distinct operational logic and requiring different capabilities. These examples are carefully chosen to illustrate the diverse challenges and requirements encountered when interacting with different types of applications.

Table 12: More Example Showcase from Each Subset of Domains.

Related App(s)	Instruction(s)	Screenshot	Abilities Needed
OS	<i>I want to install Spotify on my current system. Could you please help me?</i>		knowledge of OS; omit distractions
Calc	<i>Check the names in column "Names with duplicates" and put the unique ones in column "Unique Names". Keep the original order.</i>		massive elements; knowledge tricks or reasoning over long actions
Calc	<i>I have a lookup table for the officers of each branch. Please, here is another table in which I need to fill with the officer names according the headoffice (i.e., the branch name). Help me to complete this.</i>		massive elements; knowledge of formulas and functions
Calc	<i>Given a partial calendar, please highlight all the weekends (Saturday & Sunday) by setting the cell background as red (#ff0000).</i>		massive elements; commonsense reasoning; software tricks
Impress	<i>I closed the slide pannel on the left and idk how to get it back please help</i>		software knowledge; imagine about UI layouts; overcome typos in instruction

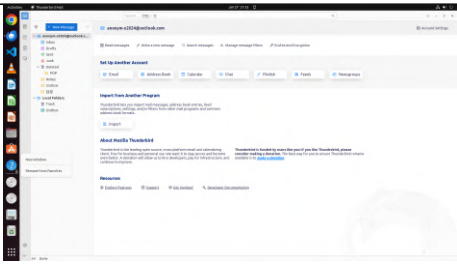
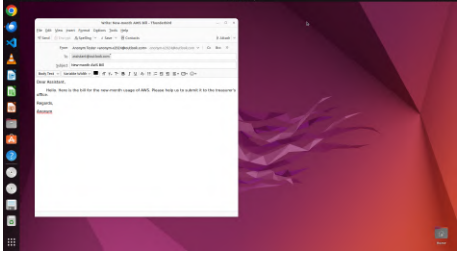
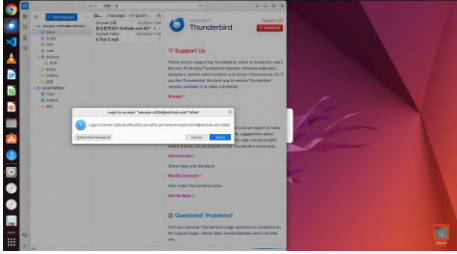
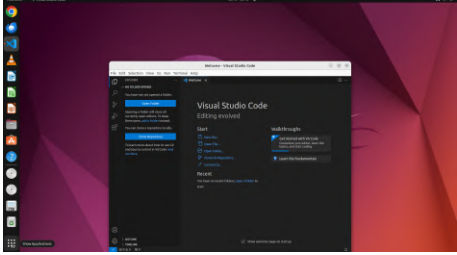
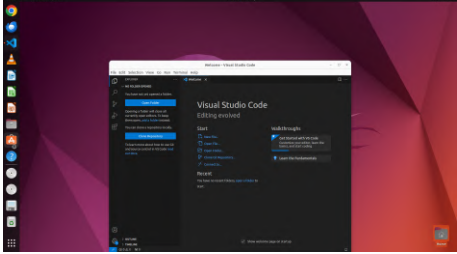
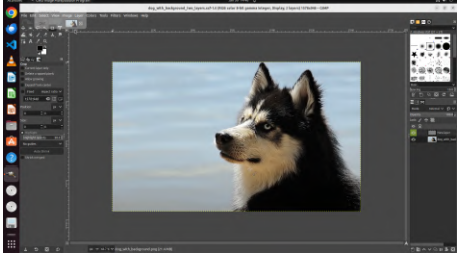
Continued on next page

Table 12 – continued from previous page

Related App(s)	Task Instruction	Screenshot of Initial State	Abilities Needed
Impress	<i>On it Whenever I launch a LibreOffice Impress, it uses both screens, one for current slide and next slide and another for actual presentation. What I want is to use only one monitor which shows presentation. I dont want the screen with Current slide and Next slide so that it can be used for other purposes. How should I achieve this?</i>		reason from unprofessional phenomenon expression
Writer	<i>Copy the screenshot 1.png from the desktop to where my cursor is located</i>		locate the position of cursor; switch from desktop and app
Chrome	<i>Can you help me clean up my computer by getting rid of all the tracking things that Amazon might have saved? I want to make sure my browsing is private and those sites don't remember me.</i>		understanding the unprofessional expression
VLC	<i>I am reading lecture note in PDF while a music video is running in VLC media player. But I find I need to switch to the player every time I need to pause/start. Could you help me change the setting to allow pausing the video using keyboard shortcut without minimizing the PDF reader? I want to focus on the lecture note and don't be disturbed by the app switching.</i>		understanding the reference from unprofessional expression; software knowledge
VLC	<i>Hey, could you turn this video the right way up for me? And once it's flipped around, could you save it for me with the name '1984_Apple.mp4' on the main screen where all my files are?</i>		software knowledge; spatial judgment ability

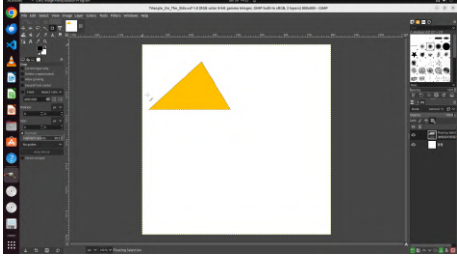
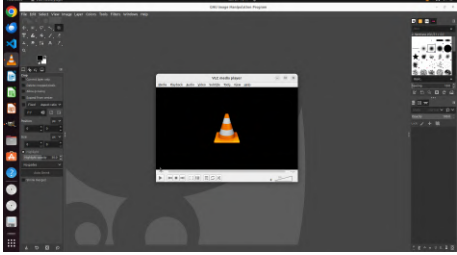
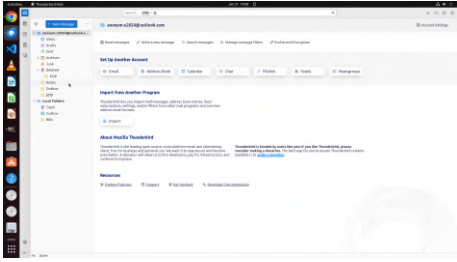
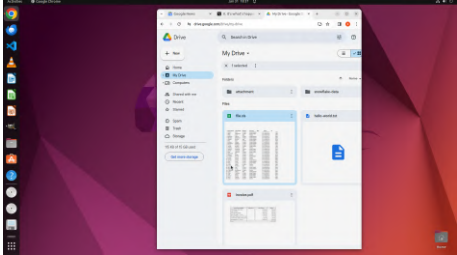
Continued on next page

Table 12 – continued from previous page

Related App(s)	Task Instruction	Screenshot of Initial State	Abilities Needed
Thunderbird	Create a local folder called "Promotions" and create a filter to auto move the inbox emails whose subject contains "discount" to the new folder		software knowledge
Thunderbird	Attach the my AWS bill to the email. The bill is stored at /aws-bill.pdf. Don't close it or send it. I haven't finish all the contents.		file management; extra requirement
Thunderbird	I've got a bunch of email accounts in Thunderbird, and it's a hassle to check them one by one. Can you show me how to set up a unified inbox so I can see all my emails in one place?		deep-hidden feature, need to be explored even by human users; pop-up window
VS Code	Please modify VS Code's settings to disable error reporting for Python missing imports.		software knowledge to deal with settings; reasoning to understand the cause and solution of the error
VS Code	Please help me install the autoDocstring extension in VS Code.		software knowledge to deal with Extensions; reasoning to search and install the extension successfully
GIMP	Could you make the background of this image transparent for me?		precise and intricate operations

Continued on next page

Table 12 – continued from previous page

Related App(s)	Task Instruction	Screenshot of Initial State	Abilities Needed
GIMP	<i>Help me choose the yellow triangle and position it at the center of my picture.</i>		spatial perception and reasoning, as well as precise control of actions
Multiple (VLC+GIMP)	<i>Could you help me create an Animated GIF from a video file using VLC and GIMP from the source of video "src.mp4", 5-second clip beginning at 00:03?</i>		software knowledge to undergo sophisticated processes and ability to process multi-step procedure successfully
Multiple (ThunderBird+Writer+Chrome)	<i>Help me export charts, graph or other images from docx files received in email "Lecture Document" in Notes folder and upload these png files to the figures/ folder in Google Drive for later use (use numbers to name them).</i>		ability to selectively export charts, graphs and images from docx file; software knowledge for google drive file upload
Multiple (Chrome+Calc)	<i>Could you help me extract data in the table from a new invoice uploaded to my Google Drive, then export it to a Libreoffice calc .xlsx file in the desktop?</i>		ability to do table data extraction; export data to .xlsx file

C Details of Baseline Methods

C.1 Hyper-Parameter of the Baseline Agents

We utilize the versions of gpt-3.5-turbo-16k, gpt-4-0125-preview, and gpt-4-vision-preview, respectively for GPT results, need to be noted that result could be changed from time since it is close-sourced. We also employ the gemini-pro and gemini-pro-vision versions for the Gemini models. For all language models, we set the temperature parameter to 1.0, and top_p to 0.9, and the maximum number of tokens for generation is set to 1500. We set the maximum steps of interaction to 15 and the maximum time limits to 30 minutes for all tasks since the agent could lead to a stuck environment under some unexpected cases.

C.2 Prompt Details

C.2.1 Prompt for A11y Tree, Screenshot and their Combination Setting

```
You are an agent which follow my instruction and perform desktop computer
→ tasks as instructed.
You have good knowledge of computer and good internet connection and assume
→ your code will run on a computer for controlling the mouse and keyboard.
For each step, you will get an observation of an image, which is the
→ screenshot of the computer screen and you will predict the action of the
→ computer based on the image.

You are required to use `pyautogui` to perform the action grounded to the
→ observation, but DONOT use the `pyautogui.locateCenterOnScreen` function
→ to locate the element you want to operate with since we have no image of
→ the element you want to operate with. DONOT USE `pyautogui.screenshot()`
→ to make screenshot.
Return one line or multiple lines of python code to perform the action each
→ time, be time efficient. When predicting multiple lines of code, make
→ some small sleep like `time.sleep(0.5);` interval so that the machine
→ could take; Each time you need to predict a complete code, no variables
→ or function can be shared from history
You need to to specify the coordinates of by yourself based on your
→ observation of current observation, but you should be careful to ensure
→ that the coordinates are correct.
You ONLY need to return the code inside a code block, like this:
```python
your code here
```

Specially, it is also allowed to return the following special code:
When you think you have to wait for some time, return ```WAIT```;
When you think the task can not be done, return ```FAIL``` , don't easily say
→ ```FAIL``` , try your best to do the task;
When you think the task is done, return ```DONE``` .

My computer's password is 'password', feel free to use it when you need sudo
→ rights.
First give the current screenshot and previous things we did a short
→ reflection, then RETURN ME THE CODE OR SPECIAL CODE I ASKED FOR. NEVER
→ EVER RETURN ME ANYTHING ELSE.
```

For ally tree setting and ally tree + screenshot setting, the prompts are basically the same, just replace the screenshot words with ally tree words.

C.2.2 Prompt for SoM Setting

```
You are an agent which follow my instruction and perform desktop computer
→ tasks as instructed.
You have good knowledge of computer and good internet connection and assume
→ your code will run on a computer for controlling the mouse and keyboard.
For each step, you will get an observation of the desktop by 1) a screenshot
→ with interact-able elements marked with numerical tags; and 2)
→ accessibility tree, which is based on AT-SPI library. And you will
→ predict the action of the computer based on the image and text
→ information.

You are required to use `pyautogui` to perform the action grounded to the
→ observation, but DONOT use the `pyautogui.locateCenterOnScreen` function
→ to locate the element you want to operate with since we have no image of
→ the element you want to operate with. DONOT USE `pyautogui.screenshot()`
→ to make screenshot.
```

You can replace x, y in the code with the tag of the element you want to
 ↳ operate with. such as:

```
```python
pyautogui.moveTo(tag_3)
pyautogui.click(tag_2)
pyautogui.dragTo(tag_1, button='left')
```
```

When you think you can directly output precise x and y coordinates or there
 ↳ is no tag on which you want to interact, you can also use them directly.
 But you should be careful to ensure that the coordinates are correct.
 Return one line or multiple lines of python code to perform the action each
 ↳ time, be time efficient. When predicting multiple lines of code, make
 ↳ some small sleep like `time.sleep(0.5);` interval so that the machine
 ↳ could take; Each time you need to predict a complete code, no variables
 ↳ or function can be shared from history
 You need to to specify the coordinates of by yourself based on your
 ↳ observation of current observation, but you should be careful to ensure
 ↳ that the coordinates are correct.
 You ONLY need to return the code inside a code block, like this:

```
```python
your code here
```
```

Specially, it is also allowed to return the following special code:
 When you think you have to wait for some time, return ```WAIT```;
 When you think the task can not be done, return ```FAIL``` , don't easily say
 ↳ ```FAIL``` , try your best to do the task;
 When you think the task is done, return ```DONE```.

My computer's password is 'password', feel free to use it when you need sudo
 ↳ rights.
 First give the current screenshot and previous things we did a short
 ↳ reflection, then RETURN ME THE CODE OR SPECIAL CODE I ASKED FOR. NEVER
 ↳ EVER RETURN ME ANYTHING ELSE.

C.3 Accessibility Tree Filtering

Since the original tree is large (usually over 1 million tokens in XML format), we filter the accessibility tree nodes by their tags, visibilities, availabilities, *etc.* The concrete rules are illustrated in the following Table 13.

Table 13: Criteria for keeping all y tree nodes on Ubuntu and Windows platforms

| Condition | Ubuntu | Windows |
|--|--|----------------|
| Node Tags | document*, *item, *button, *heading, *label, *scrollbar, *searchbox, *textbox, *link, *tabelement, *textfield, *textarea, *menu, alert, canvas, checkbox, combo-box, entry, icon, image, paragraph, scroll-bar, section, slider, static, table-cell, terminal, text, netuiribbontab, start, trayclockwclass, traydummysearchcontrol, uiimage, uiproperty, uiribboncommandbar | Same as Ubuntu |
| Showing | True | Not Applicable |
| Visible | True | True |
| Enabled or Editable or Expandable or Checkable | True | True |
| Has Name or Text or Image | True | True |

| Condition | Ubuntu | Windows |
|-----------|----------|----------|
| Position | ≥ 0 | ≥ 0 |
| Size | > 0 | > 0 |

C.4 Set-of-Mark Implementation Details

Our methodology involves an initial transformation of the original observational image acquired from our environment into the standardized *SoM* format and be putted into VLMs together with a table of the marks with metadata information such as tags and names. This format consists of bounding boxes that bound the sub-images of interest, each associated with a corresponding integer mark. Notably, our approach incorporates the utilization of the accessibility tree to identify the bounding boxes associated with all clickable buttons within the current image observation, instead of using segmentation models like the original *SoM* paper. Some examples of screenshots after applying SoM are shown in Figure 14. We can observe the emergence of some low-quality, unmodelable tasks, and even misleading bounding boxes, depending on the level of support from the software ecosystem. This could be another reason for the poor performance of SoM. Future improvements could be made in this regard.

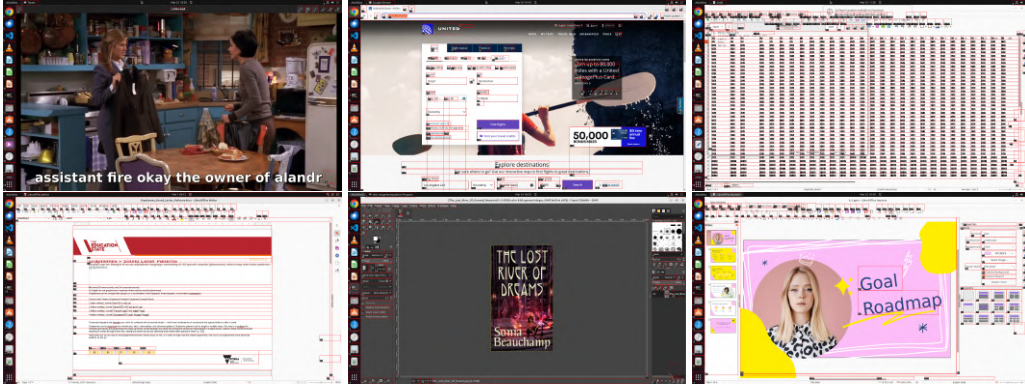


Figure 14: Showcase of example screenshots marked by SoM across different app GUI.

C.5 Full Results of Baseline Methods

Here we show the break-down results of baseline methods from different LLMs and VLMs for follow-up reference.

We have also compiled the distribution of steps taken by the GPT-4V model under our four settings: Accessibility Tree (A11y Tree), Screenshot, Screenshot combined with Accessibility Tree (Screenshot+A11y Tree), and Set-of-Mark. This data (as shown in Fig. 15) provides potential clues for future work. Overall, there are observable differences in how many steps the agent chooses to execute and when it decides to terminate under different settings. More detailed control and analysis of these behaviors can be explored in subsequent research efforts.

D Examples of Qualitative Analysis

Here we give the specific code execution process for the examples in the qualitative analysis in §5.4 and add more error cause analysis, trying to get insights for further improvement.

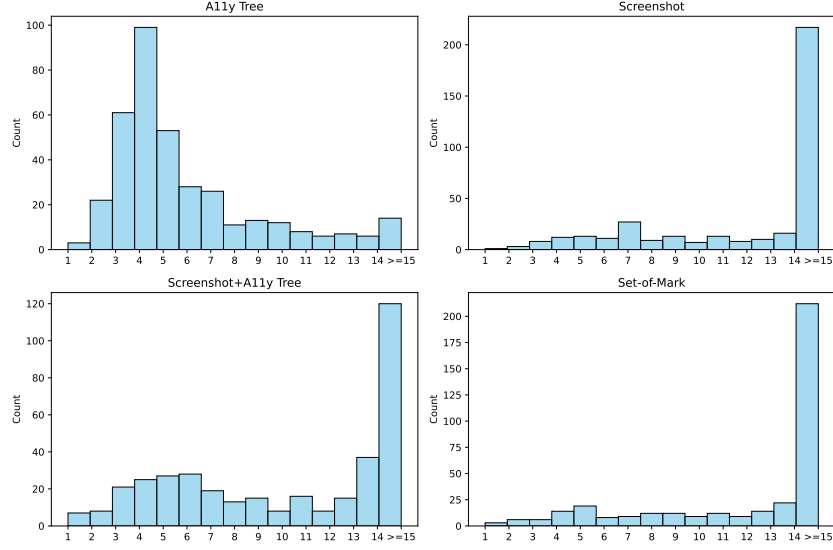


Figure 15: Distribution of steps taken by the GPT-4V based agents across four different settings.

Table 14: Detailed success rates of baseline LLM and VLM agents on OSWORLD, divided by apps (domains): OS, LibreOffice Calc, LibreOffice Impress, LibreOffice Writer, Chrome, VLC Player, Thunderbird, VS Code, GIMP and Workflow which is comprehensive with multiple apps, for gaining insights from interfaces and operation logics.

| Inputs | Model | Success Rate (\uparrow) | | | | | | | | | |
|-------------------|---------------|-----------------------------|-------|---------|--------|-------|-------|--------|-------|-------|----------|
| | | OS | Calc | Impress | Writer | VLC | TB | Chrome | VSC | GIMP | Workflow |
| A11y | Mixtral-8x7B | 12.50 | 0.00 | 0.39 | 4.34 | 10.22 | 6.67 | 2.17 | 8.69 | 3.85 | 0.10 |
| | GPT-3.5 | 4.17 | 2.13 | 6.77 | 4.35 | 6.53 | 0.00 | 2.17 | 0.00 | 0.00 | 1.62 |
| | Gemini-Pro | 4.17 | 0.00 | 2.13 | 4.35 | 12.41 | 0.00 | 2.17 | 0.00 | 7.69 | 0.63 |
| | GPT-4 | 20.83 | 0.00 | 6.77 | 4.35 | 23.53 | 26.67 | 26.09 | 30.43 | 23.08 | 2.97 |
| Screen | CogAgent | 4.17 | 0.00 | 0.00 | 4.34 | 6.53 | 0.00 | 2.17 | 0.00 | 0.00 | 0.00 |
| | Gemini-ProV | 8.33 | 0.00 | 6.77 | 4.35 | 12.41 | 0.00 | 6.52 | 8.70 | 23.08 | 2.08 |
| | GPT-4V | 12.50 | 0.00 | 2.52 | 4.35 | 18.34 | 0.00 | 6.52 | 0.00 | 7.69 | 6.04 |
| | Claude-3-Opus | 4.17 | 0.00 | 2.52 | 4.34 | 6.53 | 0.00 | 2.17 | 0.00 | 3.84 | 2.61 |
| Screen + A11y | CogAgent | 4.17 | 2.17 | 0.00 | 4.35 | 6.53 | 0.00 | 2.17 | 0.00 | 0.00 | 0.10 |
| | Gemini-ProV | 4.17 | 2.13 | 6.77 | 4.35 | 18.30 | 0.00 | 4.35 | 0.00 | 0.00 | 1.52 |
| | GPT-4V | 16.67 | 0.00 | 6.77 | 21.73 | 24.18 | 33.33 | 21.74 | 21.74 | 15.38 | 4.59 |
| | Claude-3-Opus | 12.5 | 2.13 | 4.65 | 4.34 | 18.30 | 0.00 | 2.17 | 8.69 | 7.69 | 0.99 |
| SoM | CogAgent | 4.17 | 2.17 | 0.00 | 4.34 | 6.53 | 0.00 | 2.17 | 0.00 | 0.00 | 0.00 |
| | Gemini-ProV | 4.17 | 0.00 | 0.39 | 4.34 | 6.53 | 0.00 | 0.00 | 0.00 | 0.00 | 0.63 |
| | GPT-4V | 8.33 | 8.51 | 6.38 | 13.04 | 10.66 | 33.33 | 23.91 | 17.39 | 11.54 | 6.57 |
| | Claude-3-Opus | 12.5 | 2.13 | 0.39 | 8.70 | 6.53 | 13.33 | 17.39 | 0.00 | 11.54 | 4.49 |
| Human Performance | | 75.00 | 61.70 | 80.85 | 73.91 | 70.59 | 46.67 | 78.26 | 73.91 | 73.08 | 73.27 |

D.1 Success and Failure Cases

Success Task: *I downloaded an episode of Friends to practice listening, but I don't know how to remove the subtitles. Please help me remove the subtitles from the video and export it as "subtitles.srt" and store it in the same directory as the video.*

```
# Action 1
import pyautogui
import time
# Move the mouse to the top bar of the media player window to access the menu
pyautogui.moveTo(800, 10)
# Assuming the menu is at the top of the screen
pyautogui.click()
time.sleep(0.5)

# Action 2
# Move the mouse to the Activities corner to open the GNOME Activities Overview
pyautogui.moveTo(10, 10)
# Assuming the Activities corner is at the top-left
pyautogui.click()
# Type `terminal` to search for the Terminal application
pyautogui.write('terminal')
time.sleep(0.5)
# Press Enter to open the Terminal application
pyautogui.press('enter')

# Action 3
# Focus the Terminal window and type the command to extract subtitles
pyautogui.click(640, 360)
# Clicking in the center of the screen to focus the Terminal
time.sleep(0.5)
pyautogui.write('ffmpeg -i video.mp4 -map 0:s:0 subtitles.srt')
pyautogui.press('enter')

# Action 4
# Focus the Terminal window and type the command
# to create a new video without subtitles
pyautogui.click(1100, 350)
# Clicking inside the Terminal window to focus it
pyautogui.write('ffmpeg -i video.mp4 -c copy -sn no_subtitles_video.mp4')

Done!
```

Failure Task: *Help me center align the heading in LibreOffice.*

```
# Action 1
import pyautogui
import time
# Move the mouse to the LibreOffice Writer document icon and double-click
pyautogui.moveTo(1100, 720, duration=1)
pyautogui.doubleClick()

# Action 2
# Move the mouse to the beginning of the heading text
# and click and drag to select the heading
pyautogui.moveTo(200, 230, duration=1)
pyautogui.mouseDown()
pyautogui.moveTo(800, 230, duration=1)
pyautogui.mouseUp()
# Move the mouse to the center align button in the toolbar and click
pyautogui.moveTo(400, 150, duration=1)
```

```

pyautogui.click()

# Action 3
# Assuming the text is still selected,
move the mouse to the center align button in the toolbar and click
pyautogui.moveTo(400, 150, duration=1)
pyautogui.click()

# Action 4
# Move the mouse to the center align button in the toolbar and click
pyautogui.moveTo(340, 80, duration=1)
pyautogui.click()

Failed.

```

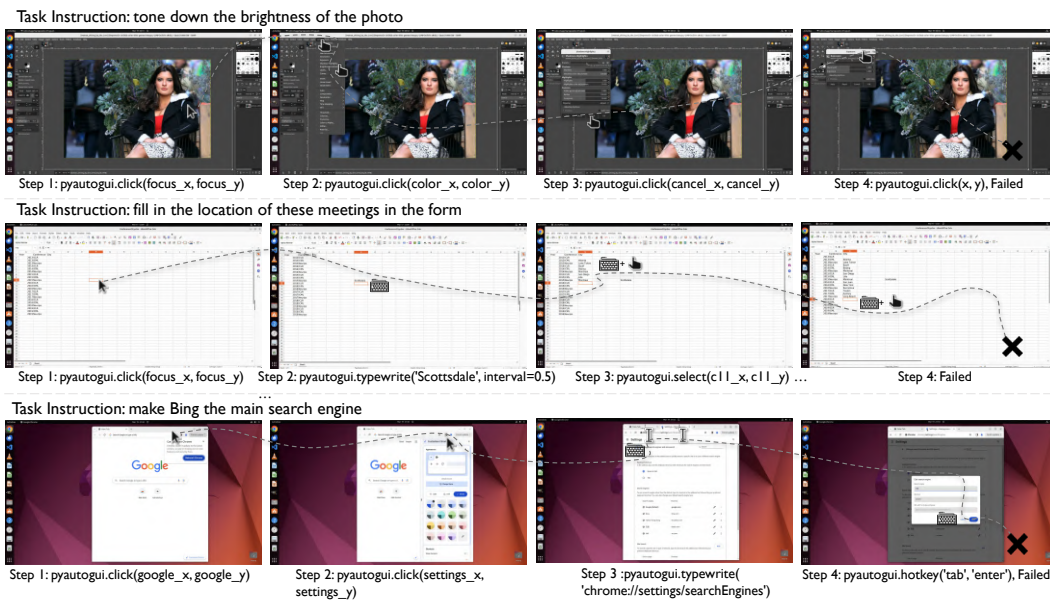


Figure 16: The figure shows examples of usage errors in three types of software. The first row is for GIMP, the second row is for LibreOffice Calc, and the third row is for Google Chrome. Detailed code execution processes are provided below.

GIMP Task: Could you tone down the brightness of my photo?

```

# Action 1
import pyautogui
import time
# Coordinates for the image icon on the desktop
# Adding half of the width and height to get the center
image_icon_x, image_icon_y = 1793 + 60, 920 + 17
# Double-click on the image icon
pyautogui.doubleClick(image_icon_x, image_icon_y)

# Action 2
# Coordinates for the `Colors` menu
# Adding half of the width and height to get the center
colors_menu_x, colors_menu_y = 344 + 28, 64 + 12
# Click on the `Colors` menu
pyautogui.click(colors_menu_x, colors_menu_y)

```



```

# Action 3
# Coordinates for the `Brightness-Contrast...` option in the `Colors` dropdown
# Assuming each menu item is approximately 28 pixels apart
brightness_contrast_option_x, brightness_contrast_option_y
= 344 + 28, 64 + 12 + 28 * 6
# Click on the `Brightness-Contrast...` option
pyautogui.click(brightness_contrast_option_x, brightness_contrast_option_y)

# Action 4
# Coordinates for the `Cancel` button in the `Shadows-Highlights` dialog
# Adding half the width and height to get the center
cancel_button_x, cancel_button_y = 375 + 42, 625 + 16
# Click the `Cancel` button
pyautogui.click(cancel_button_x, cancel_button_y)
# Coordinates for the `Brightness-Contrast...` option in the `Colors` dropdown
# Adjusted index for Brightness-Contrast option
brightness_contrast_option_x, brightness_contrast_option_y
= 344 + 28, 64 + 12 + 28 * 5
pyautogui.click(brightness_contrast_option_x, brightness_contrast_option_y)

# Action 5
# Coordinates for the `Cancel` button in the `Exposure` dialog
# Adding half the width and height to get the center
cancel_button_x, cancel_button_y
= 375 + 42, 391 + 16
pyautogui.click(cancel_button_x, cancel_button_y)
# Coordinates for the `Brightness-Contrast...` option in the `Colors` dropdown
# Adjusted index for Brightness-Contrast option
brightness_contrast_option_x, brightness_contrast_option_y
= 344 + 28, 64 + 12 + 28 * 6
pyautogui.click(brightness_contrast_option_x, brightness_contrast_option_y)

# Action 6
# Coordinates for the `Cancel` button in the `Color Balance` dialog
# Adding half the width and height to get the center
cancel_button_x, cancel_button_y
= 375 + 42, 588 + 16
pyautogui.click(cancel_button_x, cancel_button_y)
# Since the `Brightness-Contrast...` option was not correctly opened before,
# we will use an estimated position based on the dropdown menu.
# Assuming `Brightness-Contrast...` is the second option in the `Colors` menu.
brightness_contrast_option_x, brightness_contrast_option_y
= colors_menu_x, colors_menu_y + 28 * 2
pyautogui.click(brightness_contrast_option_x, brightness_contrast_option_y)

# Action 7
# Coordinates for the `Cancel` button in the `Color Temperature` dialog
# Adding half the width and height to get the center
cancel_button_x, cancel_button_y = 375 + 42, 403 + 16
pyautogui.click(cancel_button_x, cancel_button_y)
# Since the `Brightness-Contrast...` option was not correctly opened before,
# we will use an estimated position based on the dropdown menu.
# Assuming `Brightness-Contrast...` is the second option in the `Colors` menu.
brightness_contrast_option_x, brightness_contrast_option_y
= colors_menu_x, colors_menu_y + 28 * 2
# Click on the `Brightness-Contrast...` option
pyautogui.click(brightness_contrast_option_x, brightness_contrast_option_y)

# ... More Wrong Duplicate Actions; Can not find the right coordination.

# Failed

```

Libreoffice calc Task: *I now want to count the meeting cities of the three machine learning conferences in the past ten years from 2013 to 2019(including 2013 and 2019). I have listed the names and years of the conferences in excel. Please fill in the vacant locations.*

```
# Action 1
# Move to cell C2 which is the first empty cell under `City` for ICLR 2013
pyautogui.click(319, 222)
# Coordinates of cell C2
time.sleep(0.5)
# Type the city name for ICLR 2013
pyautogui.write(`Scottsdale`)
# Press Enter to move to the next cell
pyautogui.press('enter')

# Action 2
# Move to cell C3 which is the city cell for ICML 2013
pyautogui.click(319, 268)
# Coordinates of cell C3
# Type the city name for ICML 2013
pyautogui.write(`Atlanta`)
# Press Enter to move to the next cell
pyautogui.press('enter')

# Action 3
# Move to cell C4 which is the city cell for NeurIPS 2013
pyautogui.click(319, 289)
# Coordinates of cell C4
# Type the city name for NeurIPS 2013
pyautogui.write(`Lake Tahoe`)
# Press Enter to move to the next cell
pyautogui.press('enter')

... More copy-and-paste operations

# Action 15
# Move to cell C16 which is the city cell for NeurIPS 2017
pyautogui.click(319, 541)
# Coordinates of cell C16
# Type the city name for NeurIPS 2017
pyautogui.write(`Long Beach`)
# Press Enter to move to the next cell
pyautogui.press('enter')

Failed.
```

Chrome Task: *Can you make Bing the main search thingy when I look stuff up on the internet?*

```
# Action 1
pyautogui.click(x=1280, y=50)
# Approximate coordinates for the `More` button
pyautogui.move(0, 150)
# Moving the cursor down to `Settings` in the drop-down menu
pyautogui.click()
# Clicking on `Settings`

# Action 2
pyautogui.click(tag_31)
# Click on the `Finish update` button to
approximate the location of `More` button
pyautogui.move(-50, 0)
# Move left from the `Finish update` button to approximate the `More` button
```

```

pyautogui.click()
# Click on the `More` button
time.sleep(0.5)
pyautogui.move(0, 150)
# Move down to the `Settings` option in the drop-down menu
pyautogui.click()
# Click on `Settings`

# Action 3
# Click on the `Customise Chrome` button which should be the `More` button
time.sleep(1)
# Wait for the dropdown menu to appear
# The `Settings` option is usually located towards the bottom of
the dropdown menu, so I'll move the cursor down to reach it
pyautogui.move(0, 200)
# Move down to the `Settings` option in the drop-down menu
pyautogui.click()
# Click on `Settings`

# Action 4
pyautogui.click(tag_47)
# Click on the `Close` button to close the `Customise Chrome` menu
pyautogui.click(tag_25)
# Click on the `New Tab` button to open a new tab
time.sleep(1)
# Wait for the new tab to open
pyautogui.typewrite(`chrome://settings/searchEngines`)
# Type the direct URL to the search engine settings page
pyautogui.press(`enter`)

# Action 5
# Simulate pressing the `Tab` key multiple times to navigate through the options
for _ in range(10):
# Assuming it might take around 10 tabs to reach the search engine list
    pyautogui.press(`tab`)
    time.sleep(0.5)
# Wait for the focus to move to the next element
# Simulate pressing the `Enter` key to select an option,
which would be done once we reach the Bing option
pyautogui.press('enter')

# Action 6
pyautogui.click(tag_42)
# Click on the `Cancel` button to exit the edit mode for the `360` search engine

Failed.

```

D.2 Common Errors by GPT-4V Agents

In the following two figures (Figure 18 and Figure 17), we take the web page tasks as examples to show the two most common types of errors in GPT4-V: *mouse click inaccuracies* and *inadequate handling of environmental noise*.

Task Instruction: On next Monday, look up a flight from Mumbai to Stockholm.

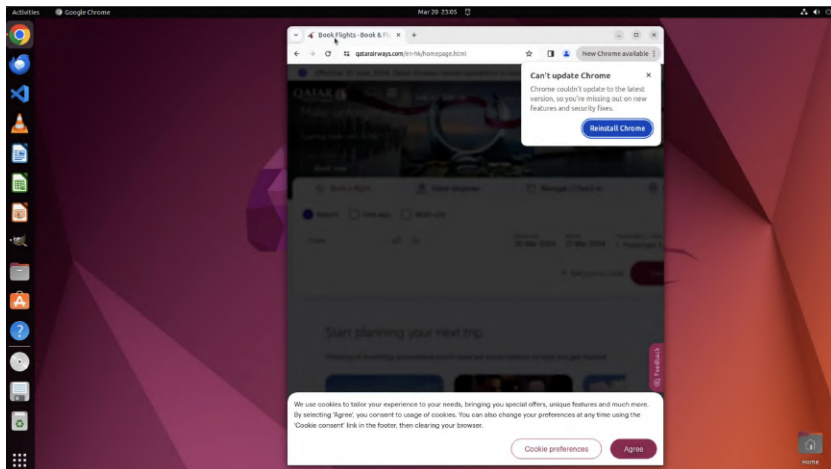


Figure 17: The error shown in the figure is due to mouse click inaccuracies. The agent was supposed to click on the product category images of the Nike official store and follow the instructions to search for women's jerseys priced over 60 dollars. However, due to a coordinate calculation error, it mistakenly clicked on the favorite button, creating a significant discrepancy between the current state and the target state. Under these circumstances, the agent is unable to backtrack to the previous state and start over.

Task Instruction: Browse the list of women's Nike jerseys over \$60.

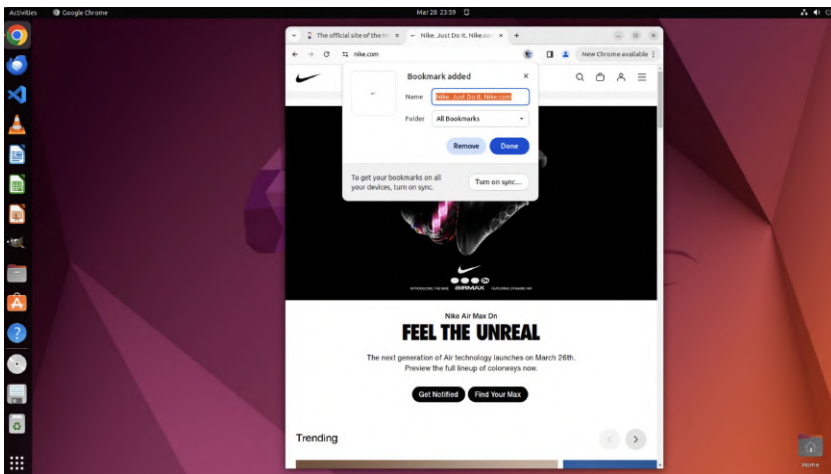


Figure 18: The error shown in the figure is due to inadequate handling of environmental noise. When the agent encounters web pages with cookies or pop-up ads, its unfamiliarity with the structure of such pages leads to attempts to interact directly with the web page without bypassing the cookies or pop-ups. Since the pop-ups or cookie requests are not closed, all actions are ineffective.

D.3 Discrepancies in Task Difficulty between Agent and Human

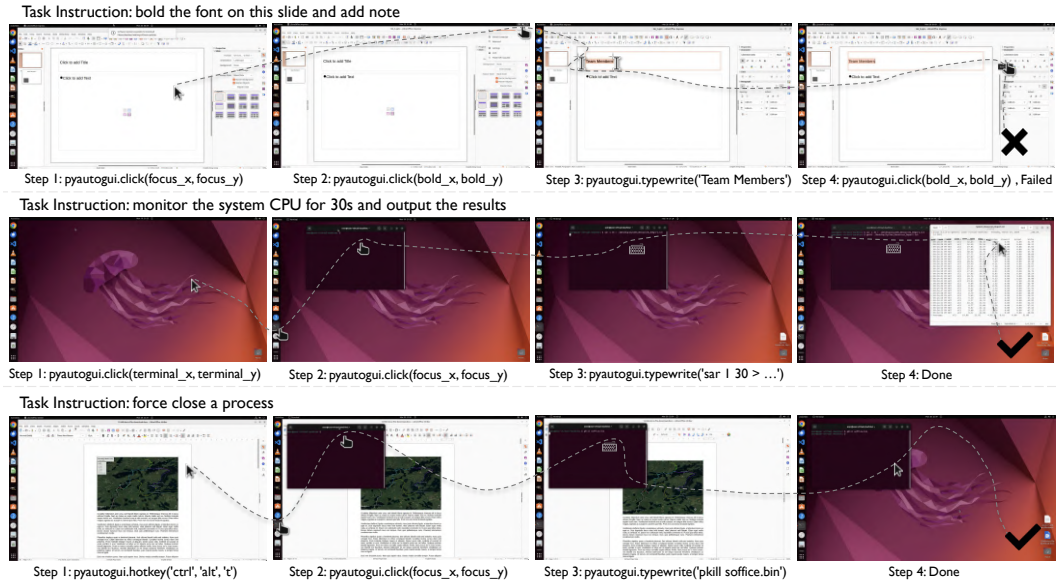


Figure 19: Supplementary examples of tasks performed by humans and agents.

Human > Agent Task: *erase all the highlighted marks in this document*

```
# Action 1
import pyautogui
import time
# Move to the 'Character Highlighting Color' button and click
pyautogui.moveTo(1020, 122)
pyautogui.click()

# Action 2
# Assuming the 'No Color' option is the first option in the dropdown
# Move the mouse slightly below the 'Character Highlighting Color' button
# to click on 'No Color'
pyautogui.moveTo(1020, 180)
time.sleep(0.5)
pyautogui.click()

# Action 3
# Move to the 'Save' button and click
pyautogui.moveTo(184, 108)
time.sleep(0.5)
pyautogui.click()

# Failed.
```

Human > Agent Task: *On page 2, add a note same to the title into the slide. Make the font of title bold.*

```
# Action 1
# Click on the vertical scrollbar to scroll down and reveal more slides
pyautogui.click(tag_43)

# Action 2
# Click on the second slide thumbnail to select it
```

```

pyautogui.click(tag_2)

# Action 3
# Click on the text box to add text
pyautogui.click(tag_42)
pyautogui.typewrite('Team Members', interval=0.1)
# Type the title
pyautogui.hotkey('ctrl', 'a')
# Select all text
pyautogui.hotkey('ctrl', 'b')
# Make the text bold

# Action 4
# Click on the Bold button to apply bold formatting
pyautogui.click(tag_68)

Failed.

```

Agent > Human Task: *use GIMP to cut out the 2s to 4s part of a video*

```

# Action 1
import pyautogui
import time
# Move the mouse to the assumed terminal icon position
# at the bottom of the dock and click to open it
# Assuming the terminal icon is at the bottom of the dock
pyautogui.moveTo(70, 1350, duration=1)
pyautogui.click()
# Type the command to use ffmpeg to trim the video
ffmpeg_command = `ffmpeg -ss 00:00:02 -to 00:00:04 -i /home/user/fullvideo.mp4
-c copy /home/user/trimmedvideo.mp4`
pyautogui.write(ffmpeg_command, interval=0.05)
pyautogui.press('enter')

# Action 2
# Close the search dialog
pyautogui.press('esc')
# Open the terminal using a keyboard shortcut (Ctrl+Alt+T is a common shortcut
# for opening a terminal in many Linux distributions)
pyautogui.hotkey('ctrl', 'alt', 't')

# Action 3
# Move the mouse to the terminal and click to focus on the terminal window
pyautogui.moveTo(260 + 722/2, 103 + 434/2, duration=1)
pyautogui.click()
# Type the command to use ffmpeg to trim the video
pyautogui.write(ffmpeg_command, interval=0.05)
pyautogui.press('enter')

# Done, but not follow the demanding in instruction.

```

Agent > Human Task: *I want to understand the resource usage of my Ubuntu system under normal workloads. Please use the 'sar' command in the 'sysstat' toolkit to monitor system activity, evaluate the status once every second for 30 seconds, output the results to "System Resources Report.txt" under Desktop.*

```

# Action 1
import pyautogui
import time

```

```

# Press the keyboard shortcut to open the terminal
pyautogui.hotkey('ctrl', 'alt', 't')
time.sleep(1)
# Wait a moment for the terminal to open

# Action 2
# Click inside the terminal to focus on it
pyautogui.click(600, 300)
# Assuming the terminal is in the upper left area of the screen
n# Type the command to monitor system activity
pyautogui.typewrite('sar 1 30 > ~/Desktop/System Resources Report.txt')

# Action 3
# Click on the 'System Resources Report.txt' file to select it
pyautogui.click(1850, 940)
# Coordinates taken from the label position and size
time.sleep(0.5)
# Right-click to open the context menu
pyautogui.click(1850, 940, button='right')
# Move to the 'Open With Other Application' option in the context menu
pyautogui.move(0, 70)
# Assuming 'Open With Other Application' is 70 pixels
down from the right-click position
# Click on 'Open With Other Application'
pyautogui.click()
time.sleep(0.5)
# Type 'gedit' to select the text editor
pyautogui.typewrite('gedit')
# Press 'Enter' to open the file with gedit
pyautogui.press('enter')

# Action 4
# Click inside the terminal to ensure it's in focus
pyautogui.click(500, 200)
# Coordinates are estimated to be within the terminal window
# Type the command to open the file with gedit
pyautogui.typewrite('gedit ~/Desktop/System Resources Report.txt')

Done!

```

Agent > Human Task: *Hey, my LibreOffice Writer seems to have frozen and I can't get it to close normally. Can you help me force quit the application from the command line? I'm on Ubuntu and I don't want to restart my computer or lose any other work I have open.*

```

# Action 1
import pyautogui
import time
# Open a terminal window
pyautogui.hotkey('ctrl', 'alt', 't')
time.sleep(1)
# Type the command to kill all instances of LibreOffice Writer
pyautogui.typewrite('killall soffice.bin')

# Action 2
# Open a terminal window
pyautogui.hotkey('ctrl', 'alt', 't')
time.sleep(1)
# Type the command to kill all instances of LibreOffice Writer
pyautogui.typewrite('killall soffice.bin')

# Action 3

```



```
# Ensure the terminal window is focused
pyautogui.click(500, 200)
# Assuming the terminal is at this position
# Type the pkill command to forcefully kill the LibreOffice Writer application
pyautogui.typewrite('pkill -9 soffice.bin')
```

Done!

D.4 Comparative Analysis: Claude-3 vs. GPT-4V

Task Instruction: find the path of the file named "secret.docx" and copy it to the clipboard.

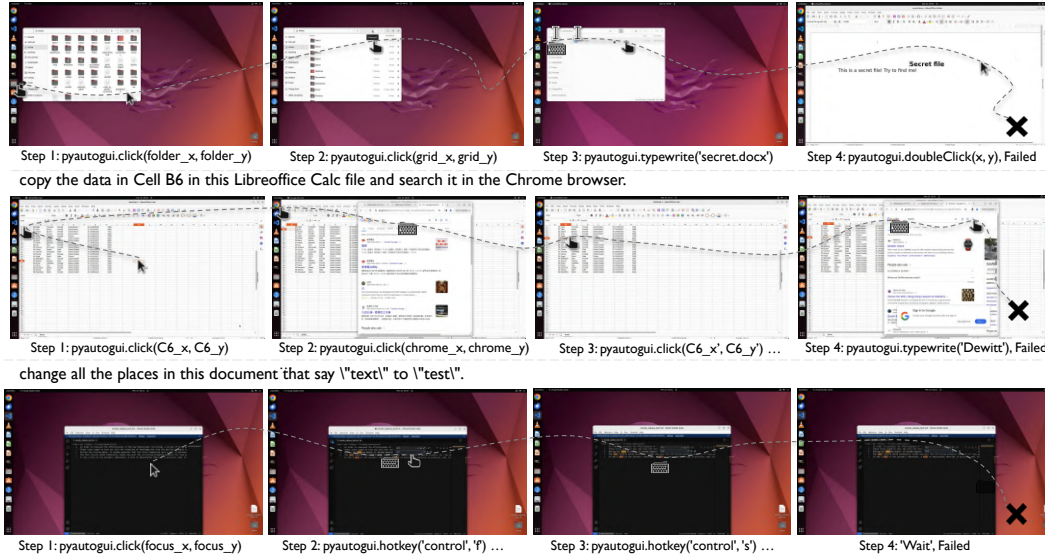


Figure 20: Screenshots of error example analysis of Claude-3 Opus.

Claude3 Failed Task1: *I remember there is a file named "secret.docx" on this computer, but I can't remember where it is. Please find the path where this file is stored and copy it to the clipboard.*

```
# Action 1
pyautogui.click(tag_10)
# click the Files icon to open the file manager

# Action 2
pyautogui.click(tag_21)
# click the search icon to open the search bar

# Action 3
pyautogui.typewrite('secret.docx')
# type the file name to search for
pyautogui.press('enter')
# press enter to start the search

# Action 4
pyautogui.doubleClick(tag_52)
# double click the secret.docx file to select it
pyautogui.hotkey('ctrl', 'c')
# press Ctrl+C to copy the selected file path to clipboard

Failed.
```


Claude3 Failed Task2: *copy the data in Cell B6 in this Libreoffice Calc file and search it in the Chrome browser.*

```
# Action 1
import pyautogui
import time
# Scroll down to view more search results if needed
pyautogui.press('pagedown')
time.sleep(1)
pyautogui.press('pagedown')
time.sleep(2)
# The answer appears to be in the snippet under the first search result
# `TV 31. u76f4u64adu4e2d.` which translates to `TV 31. Live.`
# So channel 31 in Hong Kong is RTHK TV 31
print(`Found the answer. Channel 31 in Hong Kong is RTHK TV 31.`)
# The original task of copying cell B36 from the spreadsheet
and searching it in the browser is complete
print(`Task complete.`)

# Action 2
# Activate the Chrome browser window
pyautogui.click(tag_3)
time.sleep(1)
# Click on the address/search bar
pyautogui.click(tag_30)
# Paste the copied value `Dewitt`
pyautogui.hotkey('ctrl', 'v')
# Press Enter to perform the search
pyautogui.press('enter')
print(`Searched for `Dewitt` in the Chrome browser.`)

# Action 3
print(`DONE`)

# Action 4
print(`DONE`)

Failed.
```

Claude3 Failed Task3: *Please help me change all the places in this document that say 'text' to 'test'.*

```
# Action 1
pyautogui.hotkey('ctrl', 'h')
pyautogui.write('text')
pyautogui.press('tab')
pyautogui.write('test')
pyautogui.press('enter')

# Action 2
pyautogui.press('enter')

# Action 3
pyautogui.hotkey('ctrl', 's')

Failed.
```

D.5 Analysis on SoM Effectiveness

Task w/o SoM: turning off the code checking function of the python plugin in vscode

```
# Action 1
import pyautogui
import time
# Move the mouse to the gear icon (Manage) and click it
pyautogui.moveTo(472, 1034, duration=0.5)
pyautogui.click()

# Action 2
# Move the mouse to the 'Settings' menu item and click it
# A rough central point of the 'Settings' menu item
pyautogui.moveTo(497 + 385/2, 832 + 25/2, duration=0.5)
pyautogui.click()

# Action 3
# Move the mouse to the search settings bar and click it
# A rough central point of the search settings bar
pyautogui.moveTo(783 + 587/2, 390 + 19/2, duration=0.5)
pyautogui.click()
# Type the query to search for Python-related settings
pyautogui.write('python', interval=0.05)

# Action 4
# Move the mouse to the 'Edit in settings.json' button and click it
# A rough central point of the button
pyautogui.moveTo(990 + 116/2, 624 + 15/2, duration=0.5)
pyautogui.click()

# Action 5
# Move the cursor to the correct position
pyautogui.moveTo(889 + 1/2, 432 + 1/2, duration=0.5)
pyautogui.click()
# Type the configuration to disable Python missing imports error reporting
pyautogui.write('reportMissingImports: `none`', interval=0.05)
pyautogui.press('enter')
# Save the changes
pyautogui.hotkey('ctrl', 's')

# Done
```

Task w/ SoM: turning off the code checking function of the python plugin in vscode

```
# Action 1
import pyautogui
import time
pyautogui.hotkey('ctrl', 'shift', 'p')

# Action 2
pyautogui.write('Preferences: Open Settings (UI)', interval=0.05)
time.sleep(0.5)
pyautogui.press('enter')

# Action 3
# Click on the search settings input field
pyautogui.click(x=600, y=175)
# These coordinates may need to be adjusted for your screen resolution
# Type 'Python' into the search field
pyautogui.write('Python', interval=0.05)
```

```

pyautogui.press('enter')

# Action 4
# Click on the search settings bar and type in the query again
pyautogui.click(x=800, y=150)
# Click on the search bar, the coordinates may need adjustment
pyautogui.typewrite('Python missing imports', interval=0.25)

# ... Can not find the right click area

# Failed

```

D.6 Analysis on SoM Effectiveness

The SoM, effective for web tasks focusing only on labeled bounding boxes, **did not** excel in our diverse tasks that required more expertise and flexibility. It shortens the action space of the agent and thus hinders the agent's exploration and adaptability. For example in Figure 21, in the task of "turning off the code checking function of the python plugin." in VS Code, agents without SoM succeeded by editing the settings.json, whereas SoM-equipped agents struggled with finding and deselecting the checkbox in the settings. The latter has a longer action path and fails due to incorrect mouse clicks. Furthermore, SoM's labeling of elements in professional software also has such errors, that agents without SoM can use a lly tree to calculate and adjust coordinates, while the elements' blocks under SoM can be hard to change during the task.

Task Instruction: turning off the code checking function of the python plugin in vscode

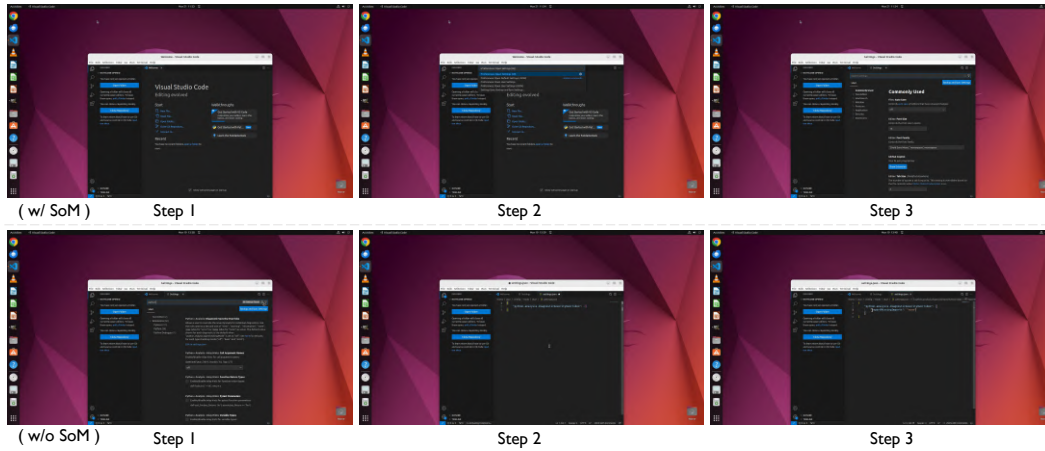


Figure 21: A task showcase where the SoM-equipped agent tried to find interactive settings, while the screen-allytree-equipped agents directly modified the value in the setting.json file.