PC-AGENT: A HIERARCHICAL AGENTIC FRAMEWORK FOR COMPLEX TASK AUTOMATION ON PC

Haowei Liu[1,2*], **Xi Zhang**[3*], **Haiyang Xu**[3†], **Yuyang Wanyan**[1,2], **Junyang Wang**[4], **Ming Yan**[3†], **Ji Zhang**[3], **Chunfeng Yuan**[1,2†], **Changsheng Xu**[1,2], **Weiming Hu**[1,2,5], **Fei Huang**[3]
[1]MAIS, Institute of Automation, Chinese Academy of Sciences, China
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences, China
[3]Alibaba Group [4]Beijing Jiaotong University
[5]School of Information Science and Technology, ShanghaiTech University, China
liuhaowei2019@ia.ac.cn, cfyuan@nlpr.ia.ac.cn
{shuofeng.xhy, ym119608}@alibaba-inc.com

ABSTRACT

MLLM-based GUI agents can assist humans in completing various tasks on smart devices automatically, demonstrating significant potential and application value. Unlike smartphones, the PC scenario not only features a more complex interactive environment with denser and more varied UI and text layouts, but also involves more intricate intra- and inter-app workflows, thus posing greater challenges for both perception and decision-making. To address the above issues, we propose a hierarchical agentic framework named PC-Agent. Specifically, from the perception perspective, we devise an Active Perception Module (APM) to overcome the inadequate abilities of current MLLMs in perceiving screenshot content. The APM integrates intention understanding and OCR to achieve fine-grained perception of the content and location of target text, and utilizes the accessibility (A11y) tree to obtain interactive element information. From the decision-making perspective, to handle complex user instructions and interdependent subtasks more effectively, we propose a hierarchical multi-agent collaboration architecture that decomposes decision-making processes into Instruction-Subtask-Action levels. Within this architecture, three agents (*i.e.*, Manager, Progress and Decision) are set up for instruction decomposition, progress tracking and step-by-step decision-making respectively. Additionally, a Reflection agent is adopted to enable timely bottom-up error feedback and adjustment. Alongside the PC-Agent framework, we introduce a new benchmark PC-Eval including 8 widely used applications and 25 real-world complex instructions. Empirical results on PC-Eval show that our PC-Agent achieves a 32% absolute improvement of task success rate over previous state-of-the-art methods. The code is available at https://github.com/X-PLUG/MobileAgent/tree/main/PC-Agent.

1 INTRODUCTION

Recently, Multi-modal Large Language Models (MLLM) (Bai et al., 2023; Ye et al., 2024; Chen et al., 2024; Li et al., 2024) have achieved impressive progress across various domains. Building on the powerful perception and reasoning abilities of MLLMs, researchers have extended them into multi-modal agents to assist humans in completing various tasks. In this field, graphical user interface (GUI) agents have garnered significant attention (Wang et al., 2024a; Agashe et al., 2024; Zhang et al., 2023; Wang & Liu, 2024), as the automation of smart devices (*e.g.*, smartphones, PCs) by agents holds vast application potential.

Compared to smartphones, the complexity of the PC scenario manifests in two aspects: **(1) More complex interactive environment.** PC's GUI encompasses denser and more diverse interactive elements (*i.e.*, icons and widgets), along with varied text layouts (*e.g.*, documents in *Word* and code in

*The first two authors contributed equally to this work.
†Corresponding authors.

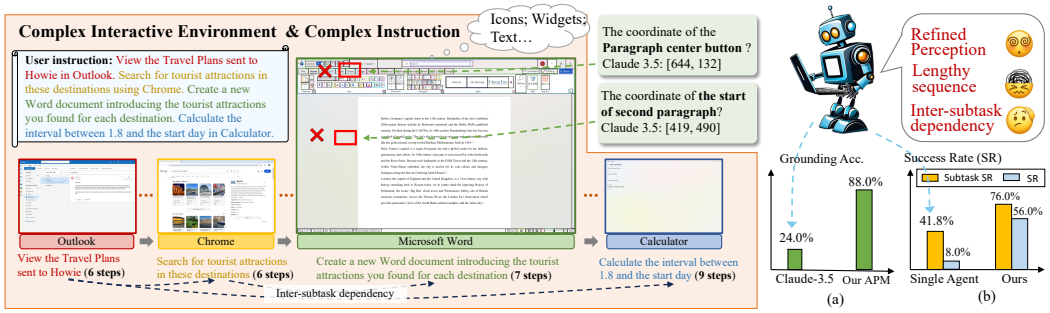

Figure 1: Illustration of the complexity of the PC scenario: (1) Complex interactive environment with dense and diverse elements. (2) Long and complex task sequences containing intra- and inter-software workflows. The black dotted lines denote the inter-subtask dependencies.

VS Code), posing significant challenges for screen perception. For example, as Figure 1 shows, the top ribbon of *Word* contains a plethora of icons and widgets, yet lacks textual labels indicating their functions. As a result, even state-of-the-art MLLMs (*e.g.*, Claude-3.5) exhibit inadequate abilities in perceiving and grounding icons and text on PC screens, and only achieves 24.0% accuracy on a GUI grounding dataset in Figure 1(a). More details about the grounding dataset can be found in Appendix A.5. **(2) More complex task sequences.** Compared to smartphones, PCs are generally used in productivity scenarios that involve more intricate intra- and inter-app workflows, and require longer and more intricate operation steps. Taking *making a travel plan* on PC (as in Figure 1) as an example, it might involve multiple subtasks across four applications. As a result, on the one hand, the lengthy operation sequences (*i.e.*, 28 steps in total) increase the difficulty of sensing the task progress. On the other hand, the existence of inter-subtask dependencies requires the agent to consider the execution results of preceding subtasks when making decisions, further increasing the decision-making difficulty. As Figure 1(b) shows, the instruction-level success rate (SR) of a single agent (GPT-4o Hurst et al., 2024) declines drastically from 41.8% to 8% compared to subtask SR, highlighting the challenge of completing real-world instructions on PC. To handle cross-app tasks, the previous work UFO (Zhang et al., 2024) designs a dual-agent framework, one for application selection and the other for specific control interactions. To tackle complex PC tasks, Agent-S (Agashe et al., 2024) combines online search and local memory for experience-augmented planning. However, these methods lack fine-grained perception and operation abilities of on-screen text, which is crucial in productivity scenarios (*e.g.*, Word document editing). Moreover, they generally overlook the complex dependencies between subtasks, thereby exhibiting limited abilities on realistic intra- and inter-app complex tasks.

In this paper, we propose the PC-Agent framework to handle the complex interactive environment and complex tasks in PC scenarios, which comprises three core designs: **(1) Active Perception Module.** To enhance the fine-grained perception and operation abilities of the agent, we propose an Active Perception Module (APM). For interactive elements, we use the accessibility tree to extract their locations and meanings. For text, we employ an MLLM-driven intention understanding agent for target text extraction, followed by OCR to obtain precise locations. **(2) Hierarchical Multi-agent Collaboration.** To improve the abilities of handling complex instructions, we adopt a divide-and-conquer approach and propose a Hierarchical Multi-agent Collaboration architecture. Specifically, we break down decision processes into three levels: **Instruction-Subtask-Action.** At the instruction level, a Manager Agent (MA) decomposes the user instruction into parameterized subtasks, with significantly fewer operation steps and lower decision-making difficulty. The MA also manages inter-subtask communication to handle complex dependencies between them. At the subtask level, a Progress Agent (PA) tracks and summarizes operation history for precise progress awareness. At the action level, a Decision Agent (DA) makes decisions step-by-step by combining the APM's perception information and PA's progress information, and interacts with the PC environment to complete the decomposed subtasks. **(3) Reflection-based Dynamic Decision-making.** Building on the above architecture, we also introduce a reflection-based dynamic decision-making mechanism for error detection in execution results, with timely feedback and adjustments. An additional Reflection Agent (RA) is set at the action level to observe screen changes before and after DA decisions, assessing the correctness of this step and conveying feedback to the DA and PA. Fig-

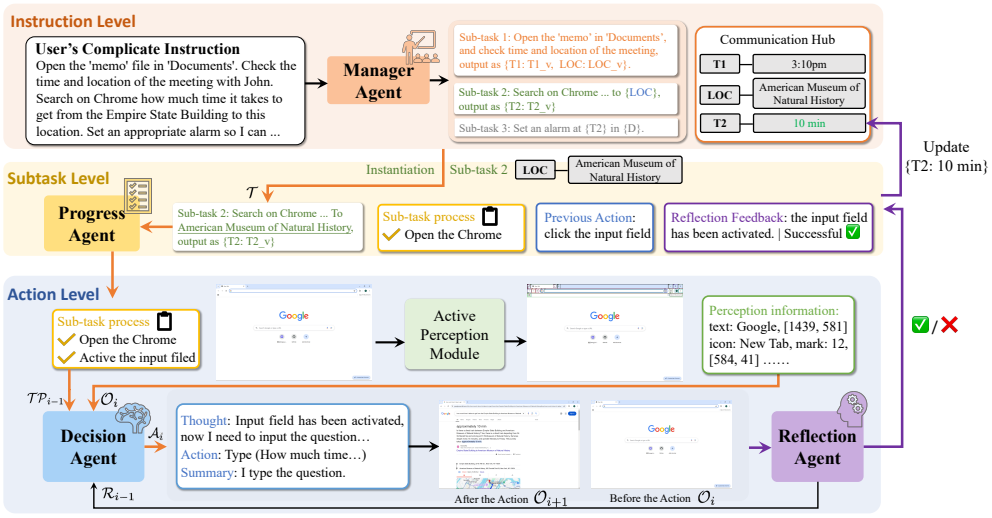

Figure 2: Overview of the proposed PC-Agent, which decomposes the decision-making process into three levels. The orange lines denote the top-down decision-making decomposition, and the purple lines represent the bottom-up reflection process.

ure 2 shows the entire process. Combining the hierarchical multi-agent collaboration architecture with reflection-based dynamic decision-making, our PC-Agent framework can **decompose complex user instructions from top to bottom** and **provide precise feedback from bottom to top during execution**. Consequently, the four agents collaborate to alleviate the difficulty of interactive environments and complex workflow tasks on PC.

To better evaluate the capabilities of agents on complex tasks, we present a new benchmark **PC-Eval** for PC productivity environments. PC-Eval comprises 8 popular applications and 25 complex user instructions, each consisting of several interdependent subtasks. It provides a challenging and realistic benchmark, by emphasizing complex workflows and long-horizon decision-making. Comparing our PC-Agent with advanced MLLM-based single agents and existing open-source PC agents on PC-Eval, we can find that PC-Agent achieves significant improvements in both instruction- and subtask-level success rates, demonstrating the effectiveness of the proposed framework.

Our contributions can be summarized as follows:

(1) We propose a PC-Agent framework to overcome the limitations of existing methods in handling complex interactive environments and complex tasks in PC scenarios. An Active Perception Module (APM) is devised to enable PC-Agent with refined perception and operation capabilities.

(2) To tackle complex PC tasks, we propose a hierarchical multi-agent collaboration architecture decomposing the decision process into three levels (*i.e.*, instruction-subtask-action), and introduce a reflection-based dynamic decision-making mechanism for timely error feedback and adjustments.

(3) We create a PC-Eval benchmark involving 8 commonly used PC applications to better assess the agent's capabilities in handling complex user instructions. Experimental results demonstrate that the proposed PC-Agent largely outperforms previous methods in completing complex PC tasks.

2 PC-Agent

2.1 Task Formulation

Given an interactive GUI environment and a user instruction $\mathcal{I}$, the GUI Agent (denoted as ρ) obtains an observation $\mathcal{O}$ about the state of the environment (*e.g.*, the current screenshot). Based on internal reasoning and planning, it makes a decision $\mathcal{A}$ about the current step's action. Through operations such as clicking and typing, it interacts with the GUI environment and alters the environment's state. Since transitioning from the initial state to the target state in a GUI environment typically

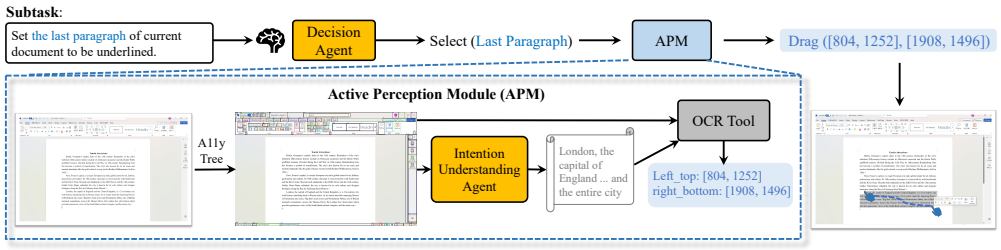

Figure 3: Illustration of the active perception module. For interactive elements, the A11y tree is adopted to obtain the bounding boxes and functional information. For text, an intention understanding agent and an OCR tool are utilized to perform precise selecting or editing.

requires multiple state transitions, this process occurs step-by-step. Each step's decision considers the operation history $\mathcal{H}$ of prior steps. The above process can be formalized as:

$$\mathcal{A}_i = \rho(\mathcal{I}, \mathcal{O}_i, \mathcal{H}_i), \tag{1}$$

where $\mathcal{A}_i$ and $\mathcal{O}_i$ represent the action and observation in the i-th step, and $\mathcal{H}_i$ is the operation history until the i-th step.

Compared to the mobile scenario, the PC scenario presents more complex interactive environments and task sequences, increasing the complexity of $\mathcal{I}$, $\mathcal{O}$ and $\mathcal{H}$ in Equation 1. This necessitates designing an agent framework tailored for complex scenarios. To address this, we propose a PC-Agent framework, which is depicted in Figure 2 and will be introduced in detail below.

2.2 Active Perception Module

MLLM-based agents struggle to accurately perceive the position and meaning of interactive elements and text. To address this and enable refined perception and operation, we propose an active perception module (APM).

Interactive Elements. We use the pywinauto API to extract the A11y tree of the GUI interface, filtering and parsing the coordinates and descriptions of interactive elements. Then we annotate the elements' bounding boxes on screenshots in an SoM (Yang et al., 2023) manner to help MLLM understand the position and meaning of the interactive elements.

Text. Text information cannot be obtained through the A11y tree, and user instructions often vaguely reference text, making it difficult to directly acquire the target text's content and position information. For instance, *bold the last two paragraphs of this Word document*. To overcome this issue, we propose utilizing active perception to obtain the content and position of the target text. As shown in Figure 3, for tasks involving refined text operations (such as *selection* or *editing*), the decision agent will first output the *Select (target text)* action. Then the APM employs an MLLM-driven intention understanding agent to determine the start and end range of the target text, followed by the use of OCR tools to precisely locate the target text for subsequent detailed operations such as *drag*.

2.3 Hierarchical Multi-agent Collaboration

PC scenarios often involve intra- and inter-app workflows, increasing the complexity of user instructions. To address this, we adopt a divide-and-conquer approach, breaking down the decision-making process into three levels: Instruction, Subtask and Action. As Figure 2 shows, based on this top-down hierarchical decomposition, we design a multi-agent collaboration architecture.

(1) Instruction-level: A manager agent (MA) is set up for high-level task management, which includes decomposing instructions into subtasks, communication among subtasks, and overall progress. **(2) Subtask-level:** A progress agent (PA) is established to manage the progress of subtasks. **(3) Action-level:** A decision agent (DA) is designated to complete subtasks. Given a specific subtask, the DA makes decisions for each step iteratively, based on the perception of the environment and the operation history provided by the PA.

Through this hierarchical multi-agent collaboration, complex user instructions are decomposed into several interdependent subtasks. The collaborative efforts of manager, progress and decision agents effectively reduce the overall decision-making difficulty and improve the success rate.

2.3.1 MANAGER AGENT

In the proposed hierarchical multi-agent collaboration architecture, the LLM-driven manager agent (MA) plays a crucial role in high-level task management:

(1) Instruction decomposition. As illustrated in Figure 2, given a complex user instruction, the MA first decomposes it into a series of parameterized subtasks. Each subtask, once instantiated, can be independently executed by the progress agent and decision agent, thus effectively reducing the complexity of individual tasks.

(2) Communication among subtasks. In a PC productivity scenario, user instructions often involve complex workflows. Therefore, the decomposed subtasks often have complex interdependencies. Specifically, there are four types of subtasks:

- The execution result of the subtask can be used to instantiate subsequent subtasks (***e.g.***, Subtask 1 in Figure 2);
- The subtask depends on the execution results of preceding subtasks for instantiation (***e.g.***, Subtask 3 in Figure 2);
- The subtask both depends on preceding subtasks for instantiation and produces execution results for subsequent subtasks (***e.g.***, Subtask 2 in Figure 2);
- The subtask is independent of other subtasks (***e.g.***, *set an alarm at 10am in the Clock app*).

As Figure 2 shows, during the whole process, the manager agent manages communication among subtasks and complex parameter transmission relationships. It maintains a communication hub, updates the output of successfully executed subtasks into this hub, and uses the hub to instantiate subsequent subtasks.

(3) Overall progress management. The manager agent also updates the overall task progress based on the execution results of subtasks reported by the progress agent.

2.3.2 PROGRESS AGENT

After the Manager Agent completes instruction decomposition and necessary inter-subtask communication to instantiate parameterized subtasks, the current independently executable subtask is handed over to the Progress Agent (PA). The PA, also driven by LLM, is responsible for tracking and summarizing the progress of subtasks based on the decisions of the decision agent and the feedback from the reflection agent (will be introduced in Section 2.4). Once the current subtask is completed, the PA feeds back the output results to the MA.

The purpose of setting up an independent PA between the MA and DA is twofold:

(1) It achieves more precise progress tracking by divide-and-conquer. PA tracks the progress of each subtask individually. This avoids summarizing the entire instruction-level history, which can be lengthy and cumbersome. **(2) It facilitates decision-making** by providing the decision agent with a clearer understanding of the operation history and which parts of the subtask remain incomplete. This avoids interference from lengthy history information in the decision-making process.

Specifically, the input for the PA at the i-th step includes four parts: (1) the current subtask $\mathcal{T}$ assigned by the MA; (2) the previous task progress $\mathcal{TP}_{i-1}$; (3) the action $\mathcal{A}_i$ output by the i-th step's DA; and (4) the reflection $\mathcal{R}_i$ after executing the i-th step's action. Based on this information, the PA outputs the updated progress $\mathcal{TP}_i$. The above process can be formalized as:

$$\mathcal{TP}_i = PA(\mathcal{T}, \mathcal{TP}_{i-1}, \mathcal{A}_i, \mathcal{R}_i). \tag{2}$$

2.3.3 DECISION AGENT

Driven by MLLM, the Decision Agent (DA) is the core agent within the entire PC-Agent framework that generates action decisions and directly interacts with the environment. Given a subtask $\mathcal{T}$, at each step, DA first obtains an observation $\mathcal{O}_i$ of the current environment using the perception module. It then combines this with the progress information $\mathcal{TP}_{i-1}$ output by PA in the previous step, and the reflection information $\mathcal{R}_{i-1}$ output by RA, to generate the decision for the current step $\mathcal{A}_i$. This process can be formalized as:

$$\mathcal{A}_i = DA(\mathcal{T}, \mathcal{O}_i, \mathcal{TP}_{i-1}, \mathcal{R}_{i-1}). \tag{3}$$

Here, decisions are generated in a Chain-of-Thought (Wei et al., 2022) manner. An inner monologue for the current step is first generated, followed by the corresponding action decision. This approach not only aids the MLLM in making better decisions but also helps the RA to judge whether the execution results meet expectations.

After obtaining the decision for the current step, we convert the decision information into a specific action type and corresponding parameters, and then use *pyautogui* to execute the corresponding keyboard and mouse operations. To simplify operations and make decisions easy to parse, we define a constrained action space, which includes *click, double click, type, select, drag, scroll, shortcut* and *stop* (detailed in Appendix A.3). This constrained action space ensures that the DA can effectively generate and execute decisions, leading to efficient and accurate task completion.

2.4 REFLECTION-BASED DYNAMIC DECISION-MAKING

Due to factors such as hallucinations and limited reasoning capabilities, even the most advanced MLLMs (*e.g.*, GPT-4o Hurst et al., 2024, claude-3.5 Anthropic, 2024) find it challenging to avoid errors in perception and decision-making. This issue becomes more pronounced with long operation sequences required by tasks, as a single error in any step can lead to the failure of the entire task.

To detect potential errors in execution results and provide timely feedback and adjustments, we design a reflection-based dynamic decision-making mechanism. Built on the hierarchical architecture introduced in Section 2.3, the dynamic decision-making mechanism operates in a bottom-up manner with the Reflection Agent at its core.

2.4.1 REFLECTION AGENT

In the action-level of the hierarchical architecture, we set up a reflection agent (RA) parallel to the decision agent (DA). After the DA makes a decision and executes the corresponding action, the RA observes the change in the system's state before and after the action to determine whether the outcome of this step meets expectations. This process can be formalized as:

$$\mathcal{R}_i = RA(\mathcal{T}, \mathcal{A}_i, \mathcal{O}_{i-1}, \mathcal{O}_i). \tag{4}$$

Depending on the execution results, the RA makes three types of judgments:

(1) The execution of the action resulted in changes to the screenshot that did not meet expectations. This may be due to incorrect action type or position parameters in DA's decision, requiring replanning to correct the mistake.

(2) No effective response was produced on the screenshot after executing the action. This might be because the action was executed on a position with no interactive elements, or the element (such as an input box) was not yet activated, necessitating an adjustment in the action execution position.

(3) The action execution produced the correct result, allowing the DA to proceed with the next decision based on this.

In the first two scenarios, the RA's output will be fed back to the next step's DA, enabling the DA to produce decisions based on reflection information to correct errors or avoid repeating ineffective actions. The RA's reflection information will also be fed back to the progress agent (PA), allowing the PA to detect errors and achieve more accurate progress tracking.

Table 1: Examples of complex instructions in PC-Eval.

Applications	Instruction	Steps
File Explorer Notepad Clock Calculator	In the Notepad app, open the 'travel_plan' file in 'Documents', and check the time and location of the travel plans. Add the travel destination to the World Clock list on the Clock app. Calculate the interval between February 18 and the start time of the travel on the Calculator.	20
Chrome Excel	Search on Chrome for the total population of China, the United States, and India in 2024 respectively. Create a new spreadsheet in Excel, write the three countries' names in column A in descending order of population, and the corresponding populations in column B.	23
File Explorer Word	Open the 'test_doc1' file in 'Documents' in File Explorer, set the title to be bold, and set the line spacing of the first two paragraphs to 1.5x in Word.	8

3 EXPERIMENTS

3.1 PC-EVAL

Existing benchmarks in real computer environments (*e.g.*, OSWorld Xie et al., 2024 and WindowsAgentArena Bonatti et al., 2024) involve relatively basic tasks that don't align with practical workflow requirements, such as *Open Paint and draw a red circle*. To better evaluate the capabilities of agents on complex PC tasks, we propose a new benchmark PC-Eval, which consists of 25 complex instructions involving 8 commonly used PC applications (*i.e.*, Chrome, Microsoft Word, Microsoft Excel, Notepad, Clock, Calculator, Outlook, and File Explorer). Each instruction comprises several interdependent subtasks, and emphasizes precise operations, practical workflows, and long-horizon. Three annotators with AI education backgrounds annotated and checked these instructions to ensure they are realistic and challenging. Table 1 shows three example instructions, with the complete list available in the Appendix A.4. Since different subtasks correspond to different pages and success criteria, creating separate scripts for automatic evaluation of each subtask would be prohibitively costly. Therefore, we employ human evaluation in this study, and we adopt the following two metrics for evaluation:

- Success Rate (SR): The success rate metric refers to the proportion of successfully completed instructions by the agents.
- Subtask Success Rate (SSR): To comprehensively evaluate the ability of agents, we annotated the subtasks of the PC-Eval instructions, and calculate the success rate of the subtasks completed by the agents.

3.2 RESULTS

Experimental setup. In the experiments, unless otherwise specified, we use GPT-4o as the foundation model for the manager, progress, decision and reflection agents within our PC-Agent framework. And we use the OpenOCR tool for OCR in the APM. We compare the proposed PC-Agent with a wide range of single- and multi-agent methods, including advanced MLLMs such as GPT-4o (Hurst et al., 2024), Gemini-2.0 (Team et al., 2023), Claude-3.5 (Anthropic, 2024), Qwen2.5-VL 72B (Team, 2025), as well as existing PC agent methods such as UFO (Zhang et al., 2024) and Agent-S (Agashe et al., 2024). To ensure as fair a comparison as possible, for MLLMs, we set the same action space via prompting, enabling them to operate as a single decision agent. As for UFO and Agent-S, we also adopt GPT-4o as their foundation model.

Results of single agents. Table 2 presents the performance comparison of PC-Agent against other methods on PC-Eval. It can be seen that those MLLM-based single agents have almost failed on all the instructions. Even the best-performing Qwen2.5-VL achieves merely a 12% success rate. This result indicates that relying solely on the abilities of a single decision agent to fulfill complex user instructions on PC is extremely challenging for the current MLLMs. Meanwhile, the success rate of these models is significantly lower than the subtask success rate. This verifies that, due to the lengthy operation sequences and complex dependencies between subtasks, completing the entire instruction is far more difficult than completing individual subtasks.

Table 2: Dynamic evaluation results on the PC-Eval benchmark.

Model	Type	Subtask Success Rate (%) ↑	Success Rate (%) ↑
Claude-3.5	Single-Agent	15.2%	0.0%
Gemini-2.0		35.4%	0.0%
GPT-4o		41.8%	8.0%
Qwen2.5-VL		46.8%	12.0%
UFO (Zhang et al., 2024)	Multi-Agent	43.0%	12.0%
Agent-S (Agashe et al., 2024)		55.7%	24.0%
PC-Agent (Ours)		**76.0%**	**56.0%**

Table 3: The results of the ablation study on the APM module, Manager agent and Reflection Agent.

Ablation study			Subtask Success Rate	Success Rate
APM	Manager Agent	Reflection Agent		
	✓	✓	58.2%	20.0%
✓		✓	50.6%	12.0%
✓	✓		48.1%	12.0%
✓	✓	✓	76.0%	56.0%

Results of multi-agent methods. UFO and Agent-S are two agent frameworks tailored for PC scenarios. However, on PC-Eval, UFO only achieves a slight advantage over the single agent using GPT-4o. While Agent-S shows an improvement in SSR over single agents, its instruction-level SR remains low. A detailed analysis reveals their problems in both perception and decision-making: **(1)** Existing methods have limited fine-grained perception and operation abilities. For instance, in Excel scenarios such as the one shown in Figure 4, UFO may input multiple pieces of information into the same cell. In Word scenarios such as the one shown in Figure 6, both UFO and Agent-S are unable to perform editing operations (*e.g.*, *"underline the last paragraph"*). **(2)** Existing methods are insufficient in handling the dependency between subtasks in complex instructions, especially in scenarios where the execution of later subtasks depends on the results of earlier ones. For example, in the instruction "*... and write down the translation of the content*", Agent-S would directly write down the text *"The translation of the content"*, rather than the translated content obtained earlier.

In contrast, our proposed APM enables the PC-Agent to have refined operation abilities. Additionally, through hierarchical multi-agent collaboration, PC-Agent achieves effective instruction decomposition, inter-subtask communication, progress management, and error reflection, which significantly improves the performance on complex tasks. As a result, our PC-Agent largely outperforms all previous methods, surpassing UFO and Agent-S by 44% and 32% respectively in terms of SR. Certainly, the fact that PC-Agent is currently unable to complete some of the instructions in PC-Eval highlights the challenges of PC-Eval and the necessity for further research on complex PC tasks.

3.3 Ablation Study

Table 3 shows the results of the ablation study on different components of the PC-Agent framework. From the ablation results we can conclude:

(1) The active perception module has a significant impact on PC-Agent's performance. Comparing the first and fourth lines, it can be seen that after removing APM, the SSR decreases by nearly 20%, while the SR decreases drastically by over 30%. On the one hand, without APM, the Decision Agent is unable to grasp the meaning of interactive elements and thus makes more errors. On the other hand, the PC-Agent loses the ability to precisely perceive and manipulate the referred text. As a result, the instruction completion rate has significantly declined.

(2) The manager agent effectively improves PC-Agent's abilities in complex workflow scenarios. Comparing the second and fourth lines, it can be seen that removing MA causes SR to significantly decline to 12%. This is because without MA, a complex instruction will be treated

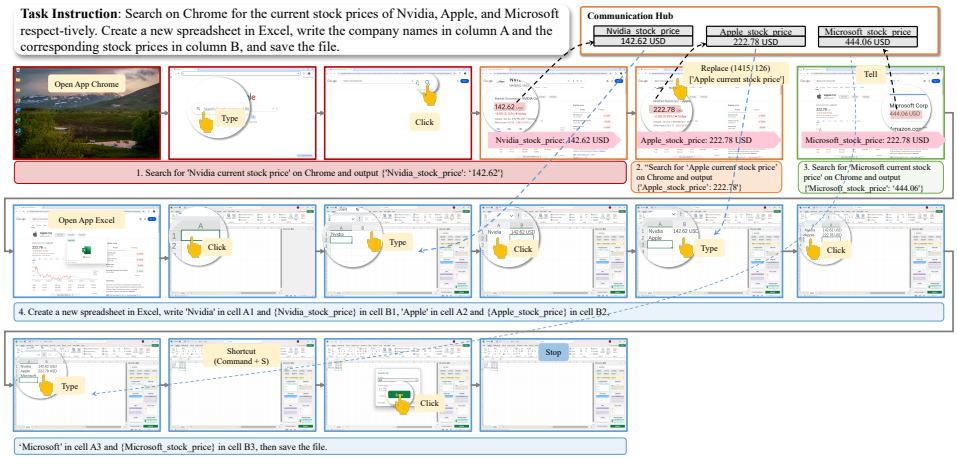

Figure 4: A case of searching for information multiple times and build an Excel sheet accordingly.

as a single task for PA and DA to execute. The lengthy operation sequences and complex dependency between the subtasks pose great challenges to progress tracking and also interfere with DA's decision-making.

(3) The reflection-based dynamic decision-making mechanism helps the model recover from errors. Comparing the third and fourth lines, it can be seen that removing RA leads to a very significant performance decrease (*i.e.*, 27.9% in SSR and 44.0% in SR). This is because during the execution of complex instructions, errors in perception and decision-making are inevitable. Removing RA causes the model to lack awareness and timely correction of errors, which predisposes it to getting stuck in meaningless repetition or incorrect steps.

3.4 Case Study

Figure 4 illustrates a complete operation process of our PC-Agent framework. Given a complex user instruction, the Manager Agent first breaks it down into four subtasks. For the first three subtasks, when each is successfully executed, the corresponding search result is updated in the communication hub. Then the MA uses the hub to instantiate the fourth subtask, which reduces the difficulty of the long-horizon decision-making process. Besides, the precise click and type operations in Excel demonstrate the effectiveness of our proposed APM in perceiving complex screen elements. We also provide a case study on reflection-based dynamic decision-making. See Appendix A.2 for details.

4 Related Work

Recent advances in MLLMs (Hurst et al., 2024; Liu et al., 2024; Wang et al., 2024b) have inspired research to extend these models to intelligent agents in various domains. Among these, there's significant focus on GUI Agents for task automation on smart devices. Currently, research in this field is more concentrated on the Mobile (Zhang et al., 2023; Wang et al., 2024a; Hong et al., 2024) and Web (Gur et al., 2023; Zheng et al., 2024) scenarios. In the PC scenario, Cradle (Tan et al., 2024) focuses on employing MLLM's reasoning abilities to realize operations in AAA games, while PC Agent (He et al., 2024) aims to enable agents to create and modify PowerPoint presentations. Despite the notable progress, their versatility remains relatively limited. To handle cross-app tasks, UFO (Zhang et al., 2024) designs a dual-agent framework, where one agent is responsible for application selection, and the other agent handles the specific control interactions. To inject PC task knowledge into decision-making, Agent-S (Agashe et al., 2024) combines online search and local memory for experience-augmented planning. Compared to previous methods, our PC-Agent focuses on complex PC tasks. We achieve more refined perception and operation (*e.g.*, editing Word documents) via the devised APM. And the proposed hierarchical framework realizes a divide-and-conquer pipeline for complex instructions, which effectively addresses the inter-subtask dependencies and significantly improves performance on complex tasks.

5 CONCLUSION

In this work, we proposed a PC-Agent framework to handle complex interactive environments and tasks in PC scenarios. An Active Perception Module was devised for refined perception and operation capabilities. And we proposed a hierarchical multi-agent collaboration architecture to decompose the decision-making process into three levels, and adopted reflection-based dynamic decision-making for timely error feedback and adjustments. We created a PC-Eval benchmark of realistic and complex user instructions. Experimental results demonstrate that the proposed PC-Agent exhibits superior performance over previous methods in completing complex PC tasks.

ACKNOWLEDGEMENT

This work is supported by Beijing Natural Science Foundation (L243015, L223003), the National Key Research and Development Program of China (No. 2020AAA0105802), the Natural Science Foundation of China (No. 62036011, 62192782), the Project of Beijing Science and Technology Committee (No. Z231100005923046).

REFERENCES

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

Anthropic. Claude 3.5 sonnet. *https://www.anthropic.com/news/ 3-5-models-and-computer-use.*, 2024.

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.

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

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24185–24198, 2024.

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.

Yanheng He, Jiahe Jin, Shijie Xia, Jiadi Su, Runze Fan, Haoyang Zou, Xiangkun Hu, and Pengfei Liu. Pc agent: While you sleep, ai works–a cognitive journey into digital world. *arXiv preprint arXiv:2412.17589*, 2024.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Gang Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering foundation agents towards general computer control. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

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.

Qwen Team. Qwen 2.5 vl. *https://qwenlm.github.io/blog/qwen2.5-vl.*, 2025.

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*, 2024a.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024b.

Xiaoqiang Wang and Bang Liu. Oscar: Operating system control via state-aware reasoning and re-planning. *arXiv preprint arXiv:2410.18963*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

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.

Qinghao Ye, Haiyang Xu, Jiabo Ye, Ming Yan, Anwen Hu, Haowei Liu, Qi Qian, Ji Zhang, and Fei Huang. mplug-owl2: Revolutionizing multi-modal large language model with modality collaboration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13040–13051, 2024.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. UFO: A UI-Focused Agent for Windows OS Interaction. *arXiv preprint arXiv:2402.07939*, 2024.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023.

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.

A Appendix

A.1 Ablation on Foundation Models

Table 4 compares the performance of different MLLMs. Here we introduce two metrics besides SR and SSR to compare the results of using different MLLMs as the foundation model:

- **Recovery Rate:** It measures the proportion of instructions where recovery occurred. The recovery behavior indicates that the agent detects an error and corrects it via reflection (no matter whether the instruction is ultimately completed).

Table 4: Performance results of PC-Agent with different foundation models on PC-Eval.

Model	Subtask SR (%) ↑	Success Rate (%) ↑	Recovery Rate (%) ↑	Manager SR (%) ↑
Qwen2.5-VL	32.9%	12.0%	40.0%	88.0%
Gemini-2.0	55.7%	28.0%	24.0%	84.0%
Claude-3.5	63.3%	40.0%	48.0%	88.0%
GPT-4o	**76.0%**	**56.0%**	**64.0%**	**96.0%**

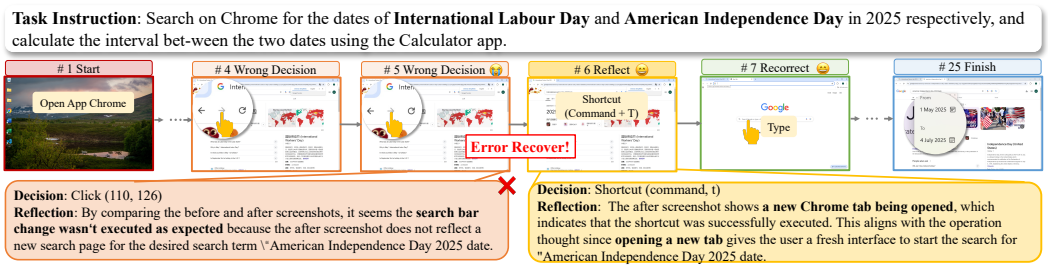

Figure 5: A case of reflection when performing multiple successive searches in Chrome.

- **Manager SR:** It assesses the ability of the Manager Agent to correctly decompose the user instructions.

From the table, we can observe that both the SSR and SR of the PC-Agent driven by GPT-4o are significantly better than the results using Gemini-2.0, Claude-3.5 and Qwen2.5-VL. And GPT-4o leads Gemini-2.0 by 40% in terms of the Recovery Rate. It may be benefit from the better perception and reasoning abilities of GPT-4o. Besides, it is worth noting that, compared to the single agent using Qwen2.5-VL, the SSR and SR of the PC-Agent using Qwen2.5-VL actually decreased. Detailed analysis reveals that this is due to Qwen2.5-VL's limited textual ability to follow the format of the output action and unsatisfactory ability to judge whether the task is completed. And the latter issue becomes more severe after the instruction is decomposed into subtasks. In conclusion, the result in Table 4 highlights that the abilities of MLLMs are the foundation of the framework's effectiveness.

A.2 More Case Study

Figure 5 shows an example within the PC-Agent framework where the proposed reflection mechanism prevents repetitive invalid operations. As can be seen, after the Decision Agent (DA) clicked the *forward button* of the Chrome browser without producing a valid response, the Reflection Agent detected this error and fed it back to the DA. Based on this feedback, the DA reconsidered in the next step and executed the correct operation (*i.e., use the Shortcut Command + T to open a new tab*).

A.3 Action Space

We define the action space as follows:

- Open App (name): Open a specific app using the system's search function.
- Click (x, y): Click the mouse at position (x, y).
- Double Click (x, y): Click the mouse twice at position (x, y).
- Select (text): Acquire the content and position of the target text by invoking the active perception module (APM).
- Type (x, y) [text]: Input text content at position (x, y).
- Drag (x1, y1) (x2, y2): Select a specific area of text content by dragging.
- Scroll (x, y) (value): Scroll the page up or down at position (x, y).

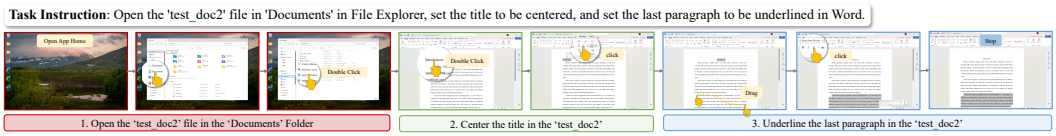

Figure 6: A case of refined text editing operations in the Word application.

- Shortcut (key list): Use shortcut keys, such as saving through ctrl+s.
- Stop: All the requirements have been met, end the current process.

A.4 Instructions in PC-Eval

We show the complete instruction list of PC-Eval as follows:

- In the Notepad app, open the ‘memo’ file in ‘Documents’, and check the second event in the morning. Set an alarm 1 hour before this event in the Clock app.
- In the Notepad app, open the ‘memo’ file in ‘Documents’, and check the location of the meeting with John. Search on Chrome how much time it takes to get from the Empire State Building to this location.
- In the Notepad app, open the ‘memo’ file in ‘Documents’, and check the time and location of the meeting with John. Search on Chrome how much time it takes to get from the Empire State Building to this location, and set an appropriate alarm on the Clock app so that I can leave the Empire State Building in time to arrive at the meeting location punctually.
- In the Notepad app, open the ‘travel_plan’ file in ‘Documents’, and check the travel destination. Use Chrome to search if the traffic at the destination drives on the left or the right.
- Search on Chrome for the dates of International Labour Day and American Independence Day in 2025 respectively, and calculate the interval between the two dates using the Calculator app.
- Open the ‘travel_plan2’ file in ‘Documents’ in the Notepad app, and check the three candidate destinations for the travel plan. Search on Chrome for the flight time from Beijing to each destination, and tell me which candidate destination has the shortest flight time.
- Search on Chrome for the current stock prices of Nvidia, Apple, and Microsoft respectively. Create a new spreadsheet in Excel, write the company names in column A and the corresponding stock prices in column B.
- Search on Chrome for the total population of China, the United States, and India in 2024 respectively. Create a new spreadsheet in Excel, write the three countries’ names in column A in descending order of population, and the corresponding population numbers in column B.
- Create a new document in Word. Write down two paragraphs introducing Alibaba and OpenAI respectively. Save the document as ‘TechCompanies’.
- Search for the paper ‘Attention is all you need’ on Chrome, download the paper and record its abstract. Create a new document in Word, write down the abstract of the paper, and save it as ‘Transformer’.
- Search for the ratings of ‘Interstellar’ and ‘12 Angry Men’ on imdb.com on Chrome. Open the ‘movie_rate’ excel file in ‘Documents’ in File Explorer, and fill in the corresponding movie ratings.
- Open the ‘test_doc1’ file in ‘Documents’ in File Explorer, set the title to be bold, and set the line spacing of the first two paragraphs to 1.5x in Word.
- Open the ‘test_doc2’ file in ‘Documents’ in File Explorer, set the title to be centered, and set the last paragraph to be underlined in Word.
- Open the ‘test_doc3’ file in ‘Documents’ in File Explorer, write down the translation of the content below the main text.

- Access https://arxiv.org/ in Chrome, search for papers related to 'multimodal agent', and download the first paper.
- Read the sent mail 'Travel' to Howie in Outlook, record the departure, destination and start date of the journey. Search for a one-way flight on booking.com on Chrome.
- Search in Chrome for the IMDb ratings of 'Leon: The Professional', 'The Shawshank Redemption', and '2001: A Space Odyssey'. Record them in a new .txt file using Notepad, sorted from highest to lowest.
- Check the sent mail 'Code' to Howie in Outlook, download the attachment 'homework.py' and open it in Visual Studio Code. Fix the error in this python code.
- Create a new Python file in Visual Studio Code, write a function that takes a list as input and outputs the k-th largest number in the list. Send this code file to Howie via Outlook.
- Search for tourist attractions in Tokyo and Kyoto respectively in Chrome, and record the information in a new Word document.
- Open the 'test_doc3' file located in 'Documents' in File Explorer, note its Chinese content, create a new Word document, and write down the English translation of the Chinese content from test_doc3.
- Open the 'test_doc1' file located in 'Documents' in File Explorer, increase the font size of the title by one level.
- In the Notepad app, open the 'travel_plan' file in 'Documents', and check the time and location of the travel plans. Add the travel destination to the World Clock list on the Clock app. Calculate the interval between February 18 and the start time of the travel on the Calculator.
- Search on Chrome for the total population of China, the United States, and India in 2024 respectively. Create a new spreadsheet in Excel, write the three countries' names in column A in descending order of population, and the corresponding populations in column B.
- Open the 'test_doc1' file in 'Documents' in File Explorer, set the title to be bold, and set the line spacing of the first two paragraphs to 1.5x in Word.
- Compare the prices of Amazon, Walmart, and Best Buy for a new Nintendo Switch console in Chrome, and write the site with the cheapest price and the price on Notepad.
- Read the mail 'Travel' in Outlook, record the departure, destination and date of the journey. Search for a round-trip flight on booking.com on Chrome.

A.5 Our GUI Grounding Dataset

On the webpage of Booking.com:

- Click to book flights
- Click to select one-way
- Click to select departure location
- Click to select destination
- Click to select date
- Click to select March 21st
- Click to select April 1st
- Click to select previous month
- Click to select next month

On the Excel page:

- Click to select A3
- Click to select E5
- Click to select top align

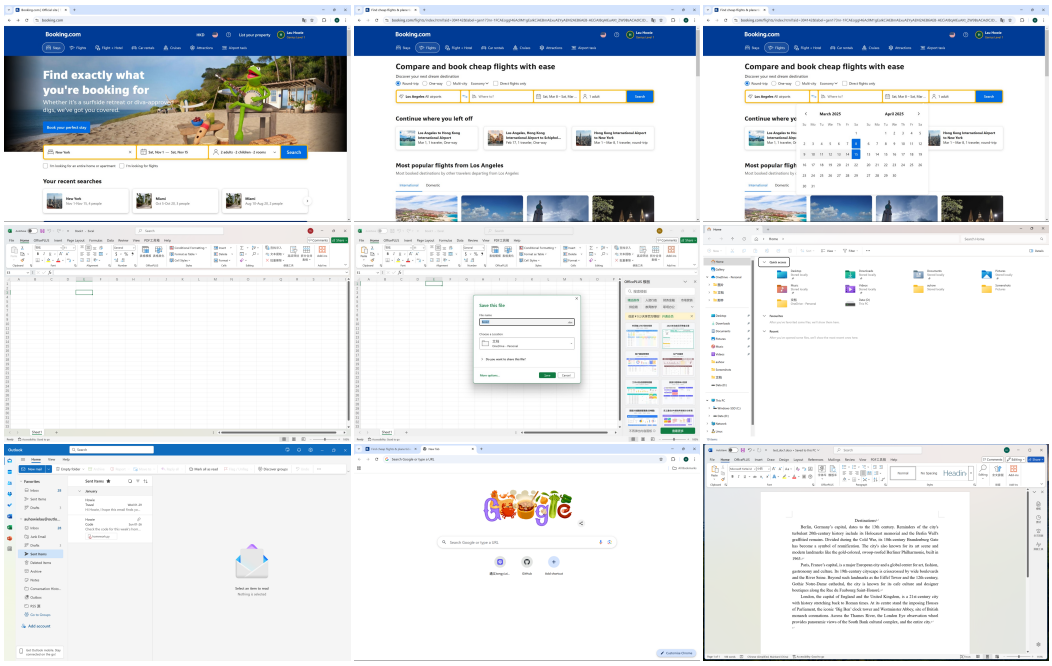

Figure 7: Example screenshots from the GUI grounding dataset we built for commonly used applications in PC scenarios.

- Click to select bottom align
- Click to select left align
- Click to select right align
- Click to save
- Click to change file name
- Click to change save location

On the File Explorer page:

- Click the Downloads folder
- Click the Documents folder
- Click the Pictures folder
- Click the Music folder

On the Outlook page:

- Click to view inbox
- Click to view spam/junk email
- Click to view sent emails
- Click to view the Travel email sent to Howie
- Click to view the Code email sent to Howie
- Click to search
- Click to create a new email
- Click to mark as read

On the Chrome page:

- Click the search bar
- Click the search box
- Click to open a new tab
- Click to bookmark
- Click settings
- Click refresh
- Click to switch to Booking.com tab

On the Word page:

- Click for bold
- Click for italic
- Click to add underline
- Click to change text color
- Click to center text
- Click to increase font size
- Click to decrease font size
- Click to adjust line spacing
- Click the top-left corner of the title
- Click the bottom-right corner of the title
- Click the top-left corner of the second-to-last paragraph
- Click the bottom-right corner of the second-to-last paragraph
- Click the bottom-right corner of the last line