AppAgent: Multimodal Agents as Smartphone Users

Chi Zhang*†
School of Engineering
Westlake University
Hangzhou, Zhejiang, China
Tencent
Shanghai, Shanghai, China
chizhang@westlake.edu.cn

Zhao Yang*
Shanghai Supwisdom
Shanghai, China
Tencent
Shanghai, China
zhao.yang@supwisdom.com

Jiaxuan Liu*
Tencent
Shanghai, China
jiaxuanliu@tencent.com

Yanda Li*
University of Technology Sydney
Sydney, NSW, Australia
Tencent
Shanghai, Shanghai, China
Yanda.Li@student.uts.edu.au

Yucheng Han
Nanyang Technological University
Singapore, Singaore, Singapore
Tencent
Shanghai, Shanghai, China
yucheng002@e.ntu.edu.sg

Xin Chen
Tencent
Shanghai, Shanghai, China
chenxin2@shanghaitech.edu.cn

Zebiao Huang
Tencent
Shanghai, China
zebiaohuang@tencent.com

bin fu
Tencent
Shanghai, China
brianfu@vip.qq.com

Gang Yu
Tencent
shanghai, China
skicy@outlook.com

Abstract

Recent advancements in large language models (LLMs) have led to the creation of intelligent agents capable of performing complex tasks. This paper introduces a novel LLM-based multimodal agent framework designed to operate smartphone applications. Our framework allows the agent to mimic human-like interactions such as tapping and swiping through a simplified action space, eliminating the need for system back-end access and enhancing its versatility across various apps. Central to the agent's functionality is an innovative in-context learning method, where it either autonomously explores or learns from human demonstrations, creating a knowledge base used to execute complex tasks across diverse applications. We conducted extensive testing with our agent on over 50 tasks spanning 10 applications, ranging from social media to sophisticated image editing tools. Additionally, a user study confirmed the agent's superior performance and practicality in handling a diverse array of high-level tasks, demonstrating its effectiveness in real-world settings. Our project page is available at https://appagent-official.github.io/.

CCS Concepts

• **Human-centered computing** → **User interface management systems**.

*denotes co-first authorship.
†denotes corresponding author

CHI '25, Yokohama, Japan

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3713600

Keywords

Multimodal, smartphone, large language model

ACM Reference Format:
Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, bin fu, and Gang Yu. 2025. AppAgent: Multimodal Agents as Smartphone Users. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3706598.3713600

1 Introduction

The emergence of large language models, such as ChatGPT [24] and GPT-4 [25], marks a significant milestone in the field of artificial intelligence and natural language processing. These advanced models represent a fundamental change in how machines understand and generate human language, exhibiting a level of sophistication and versatility previously unattainable. One of the most exciting developments in this field is the capability of LLMs to function not just as language processors, but as agents capable of performing complex tasks. This evolution is evident in initiatives such as AutoGPT [45] and MetaGPT [15], which showcase the practical applications of LLMs in tasks requiring advanced cognitive functions like reasoning, planning, and collaboration. These developments extend their utility far beyond simple linguistic tasks and profoundly impact technology and everyday life.

However, a key limitation of these LLM-based agents has been their reliance solely on text-based information. This restriction has curtailed their perception and interaction with their environment. The introduction of models equipped with vision capabilities, such as the latest iteration of GPT-4, marks a pivotal breakthrough. By integrating the ability to process and interpret visual information, these models can now understand aspects of their surroundings that are difficult or impossible to convey through text alone. This extended capability enables LLMs to interpret context, recognize

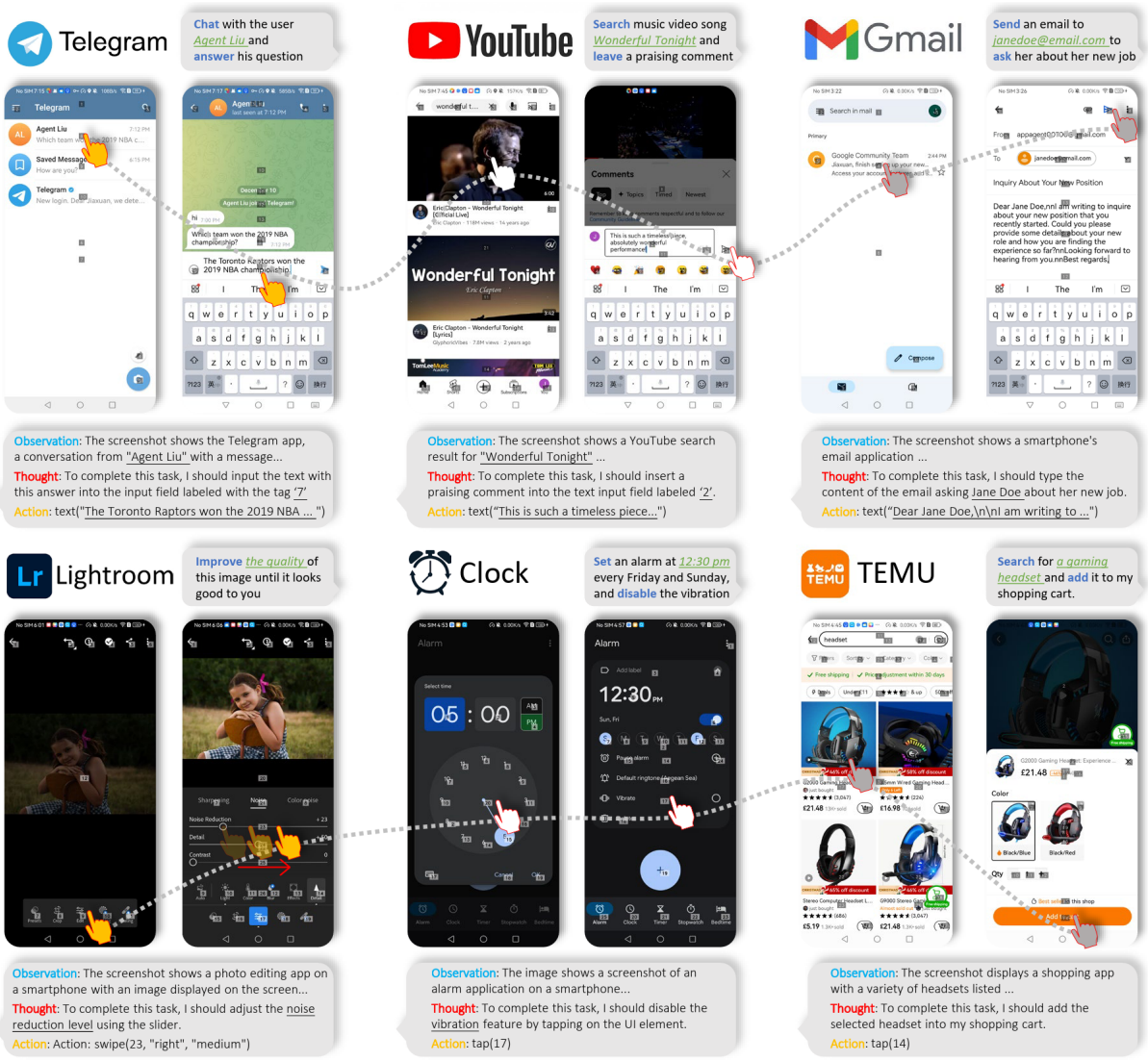

Figure 1: Diverse applications of our multimodal agent framework for smartphone App operation. We evaluate the effectiveness of our agent model on 50 tasks across 10 different Apps, highlighting its adaptability and effectiveness in a real-world context.

patterns, and respond to visual cues, thus providing a more holistic and interactive experience with the world.

In our work, we focus on building a multimodal LLM-based agent leveraging the vision capabilities of multimodal LLMs to undertake tasks previously unachievable by text-only agents. In particular, we explore an interesting but challenging application that builds an agent to operate any smartphone application (App) in the mobile operating system. Our approach differs significantly from existing intelligent phone assistants like Siri, which operate through system back-end access and function calls. Instead, our agent interacts with smartphone apps in a human-like manner, using low-level operations such as tapping and swiping on the graphical user interface (GUI). The proposed agent offers multiple advantages. Firstly, it eliminates the need for system back-end access, making our agent universally applicable across various applications. Additionally, this approach enhances security and privacy, as the agent does not require deep system integration. Furthermore, by operating on the GUI level, our agent can adapt to changes in app interfaces and updates, ensuring long-term applicability and flexibility.

However, creating a multimodal agent capable of interacting with diverse smartphone apps poses several challenges. First, performing GUI-level operations like tapping and swiping requires accurate prediction of parameters such as coordinates and movement directions, which is difficult for LLMs. Second, training an LLM to understand phone usage also presents a significant challenge. Existing research [3, 4] indicates that adapting current models for embodied tasks requires extensive training data, and collecting

a large dataset of app demonstrations is formidable. Most importantly, different apps have unique GUIs with varying icon meanings and operational logic. It is uncertain whether adapted models with supervised finetuning can effectively generalize to unseen apps.

In this paper, we introduce a multimodal agent framework aimed at operating any smartphone app like human users. Inspired by how humans quickly learn to use new apps, either through trial-and-error exploration or by observing demonstrations from others, the learning of our framework involves an exploration phase where the agent interacts autonomously with apps and learns from their outcomes in a few-shot manner. These interactions are documented, which serves as external knowledge to assist the agent in navigating and operating the apps during deployment. This in-context learning process can be accelerated by observing a few human demonstrations. Following this exploratory phase, the agent can operate the app by consulting the constructed document based on its current state, eliminating the need to collect extensive training data for adapting the parameters of the LLMs. The document-driven few-shot learning approach offers several advantages, including the ability to enhance agent performance by improving document quality—a relatively straightforward process. Additionally, updates to apps can be accommodated by modifying the document, obviating the need for re-training LLMs.

To validate its effectiveness, we tested our agent on 10 different apps, ranging from social media and messaging to email, maps, shopping, and even complex image editing apps. Both quantitative results and user studies underscore the advantages of our design, particularly its adaptability, user-friendliness, and efficient learning and operating capabilities across a wide range of applications. This underlines the potential of our agent as a versatile and effective tool in the realm of smartphone app operation. Fig. 1 showcases application cases of our approach across various apps.

In summary, this paper makes the following contributions:

- We open-source a multimodal agent framework, focusing on operating smartphone applications with our developed action space.
- We propose an innovative in-context learning strategy, which enables the agent to learn to use novel apps quickly.
- Through extensive experiments across multiple apps, we validate the advantages of our framework, demonstrating its potential in the realm of AI-assisted smartphone app operation.

2 Related Work

2.1 Large language models

The development of ChatGPT [24] and GPT-4 [25] represents a crucial advancement in natural language processing. Unlike earlier LLMs, these new models [32–34, 49, 51] enable multi-round conversations and have the impressive ability to follow complex instructions. The integration of vision capabilities in GPT-4V [46] is a further milestone, enabling the language model to process and interpret visual data. This addition has broadened the scope of potential AI applications, allowing GPT-4 to undertake diverse tasks such as problem-solving, logical reasoning, tool usage, API calls, and coding. Recent studies [44, 47] have shown that GPT-4V can understand various types of images, including simple user interfaces (UIs) in popular smartphone apps. However, challenges arise when the apps are new and their UIs are less typical, which highlights a major problem that our work aims to address. Among open-source efforts from the industry and the research community, the LLaMA series [33, 34] are the most popular equivalents and have been fine-tuned to acquire conversational abilities, employing a decoder-only architecture similar to ChatGPT [32, 51]. Building upon LLaMA, many multimodal LLMs, such as LLaVA [21, 22], ChartLlama [14], and StableLLaVA [20], also demonstrate vision understanding capabilities akin to those of GPT-4V. Nevertheless, a performance gap persists between these open-source models and GPT-4V, suggesting potential areas for further development.

2.2 LLMs as agents

The use of LLMs as agents for executing complex tasks has gained increasing attention. Initiatives like AutoGPT [45], HuggingGPT [30], and MetaGPT [15] illustrate this trend, and many projects demonstrate impressive capabilities, moving beyond basic language tasks to engaging in activities requiring higher cognitive functions, such as software development [6, 27] and gaming [10, 26, 43]. In this context, Yao *et al.* [48] introduce an innovative approach that synergizes reasoning and acting in LLMs, significantly enhancing their decision-making and interactive capabilities. LLM-based agents are designed to utilize the advanced language and reasoning skills of LLMs to interact with and manipulate their environment [12, 23, 38, 42]. This includes performing tasks that require understanding context, making decisions, and learning from interactions [17, 40]. Such agents are pivotal in applications where human-like cognitive abilities are essential.

Recent advancements have also integrated LLMs with traditional search and planning methods to enhance autonomous task completion. Language Agent Tree Search [52] combines Monte Carlo Tree Search with LLMs to improve multi-step reasoning and planning for autonomous agents, achieving state-of-the-art results in tasks like programming and web navigation. Koh *et al.* [19] introduce a best-first tree search for LLMs, enhancing their performance on complex tasks by improving success rates and reducing steps in interactive environments. Agent Workflow Memory [37] allows agents to learn and reuse task workflows from past experiences, significantly boosting performance on long-horizon tasks such as web navigation and shopping.

The emergence of multimodal LLM agents [3, 4, 11, 29, 36, 50], capable of processing various inputs including text, images, audio, and video, has further broadened the scope of LLM applications. This versatility is particularly beneficial for LLM-based agents, enabling them to interact more effectively with their environment and complete more complex tasks, such as completing household tasks in a physical world [1], generating 3D assets via procedural tool use [31], or mastering over 600 tasks across different domains at the same time [29]. A growing body of work has explored the application of multimodal LLMs in different contexts, such as web interaction and app control. Recent systems like Mind2Web [8], WebArena [53], WorkArena [9], and VisualWebArena [18] have leveraged LLMs to operate and interact with web pages, while others, such as OSWorld [41], extend similar capabilities to desktop PCs. Our research contributes to this area by focusing on an agent

designed to operate smartphone applications. This agent's ability to interpret screenshots from the operating system demonstrates its flexibility and adaptability, making it a valuable tool in a wide range of applications.

Recent studies have also focused on training LLMs for GUI understanding. These methods often rely on large-scale datasets, which enable high generalization but require substantial computational resources. For example, CogAgent [16] uses millions of instances from high-resolution GUI data and VQA datasets. The Android in the Wild (AITW) [28] dataset contains 715,000 operation instances across 30,000 tasks. Unlike these large-scale methods, AppAgent relies on online dynamic data gathered through exploration and demonstrations, making it more data-efficient. This approach is distinct from previous studies that focused on datasets or models for UI understanding and enables quick adaptation to new tasks without the need for extensive pre-collected datasets, offering a more scalable solution.

2.3 Programming by Demonstration

Our work is also related to Programming by Demonstration (PbD), a method where agents learn tasks by observing user demonstrations rather than explicit programming [7, 13]. PbD has been widely applied in robotics, enabling robots to perform complex tasks by replicating demonstrated actions [2, 5]. In our approach, the agent learns from human demonstrations or its own exploration, aligning with the objective of PbD by enabling task learning through examples, thus simplifying task automation. The related work of Wu *et al.* [39] introduces the Never-ending UI Learner, an autonomous app crawler that learns UI interaction tasks by programmatically tapping UI elements and observing the effects, avoiding the need for costly human labeling.

3 Method

This section details the methodology behind our innovative multimodal agent framework. This framework enables an agent to interact with smartphone applications in a manner akin to human behavior. We first describe the experimental environment and action space, which are foundational elements of our system. Next, we discuss the high-level design of the exploration phase, where the agent learns app functionalities either through autonomous interactions or by observing human demonstrations. Finally, we outline the deployment phase, explaining how the agent applies its acquired knowledge to execute high-level tasks.

We strongly encourage readers to refer to Appendix A and supplementary material for detailed prompts related to the different functionalities and designs described in this section for better understanding.

3.1 Environment and action space

Experimental environment: Our experimental environment is built on a command-line interface (CLI), allowing the agent to interact with smartphone apps. We chose the Android operating system for our experiments. The agent receives two key inputs: a real-time screenshot showing the app's interface and an XML file detailing the interactive elements. The XML file can be parsed from Android apps using off-the-shelf tools, without accessing the back-end. To enhance the agent's ability to identify and interact with these elements seamlessly, we assign each element a unique identifier. These identifiers are derived either from the resource IDs in the XML file (if provided) or are constructed by combining the class name, size, and content of the element. On the app screenshot, we dynamically overlay these identifiers as semi-transparent numbers on the corresponding elements, as shown in Fig. 1. This helps the agent to interact accurately without needing to specify exact positions on the screen and enhances the agent's precision in controlling the phone.

Action space: Our agent's action space mirrors common human interactions with smartphones: taps and swipes. We designed six basic functions:

- `Tap(element : int)` : This function simulates a tap on the UI element numbered on the screen. For example, `tap(5)` would tap the element labeled '5' on the screen.
- `Long_press(element : int)` : This function emulates a long press (for 1 second) on a UI element.
- `Swipe ( element : int, direction : str, dist : str)`: It allows the agent to swipe on an element in a specified direction (up, down, left, right) and distance (short, medium, long). For instance, `swipe(21, "up", "medium")` would swipe up on element "21" for a medium distance.
- `Text(text : str)` : To bypass inefficient virtual keyboard typing of each character, this function inputs text directly into an input field when a virtual keyboard is visible. For example, `text("Hello, world!")` inputs the string "Hello, world!".
- `Back()` : This system-level function enables the agent to return to the previous UI screen. It is useful for backtracking, such as when the agent needs to undo a navigation step or recover from an incorrect action.
- `Exit()` : This specialized function is employed to conclude processes, typically invoked upon successful task completion.

These predefined actions are designed to simplify the agent's interactions, particularly by eliminating the need for predicting precise screen coordinates, which significantly challenges the multimodal language models' abilities.

3.2 Exploration phase

Exploring by autonomous interactions. The Exploration Phase is a critical component of our proposed framework. In this phase, the agent autonomously interacts with smartphone apps to learn their functionalities and UI elements, mimicking how humans explore and familiarize themselves with new apps. The agent begins by being assigned a task, and it then interacts with the app's graphical user interface (GUI) elements through trial and error. The agent's task can be automatically generated by a large language model and is designed to mirror real-world deployment scenarios. The agent chooses actions based on its understanding of the task and the UI. For example, when tasked with sending an email, the agent may attempt to identify the text box and the send button based on its understanding of typical UI patterns. Then, the agent, driven by an LLM, attempts to figure out the functions of UI elements and the effects of specific actions by analyzing screenshots before and after each action, as shown in Fig. 2. For instance, the prompt

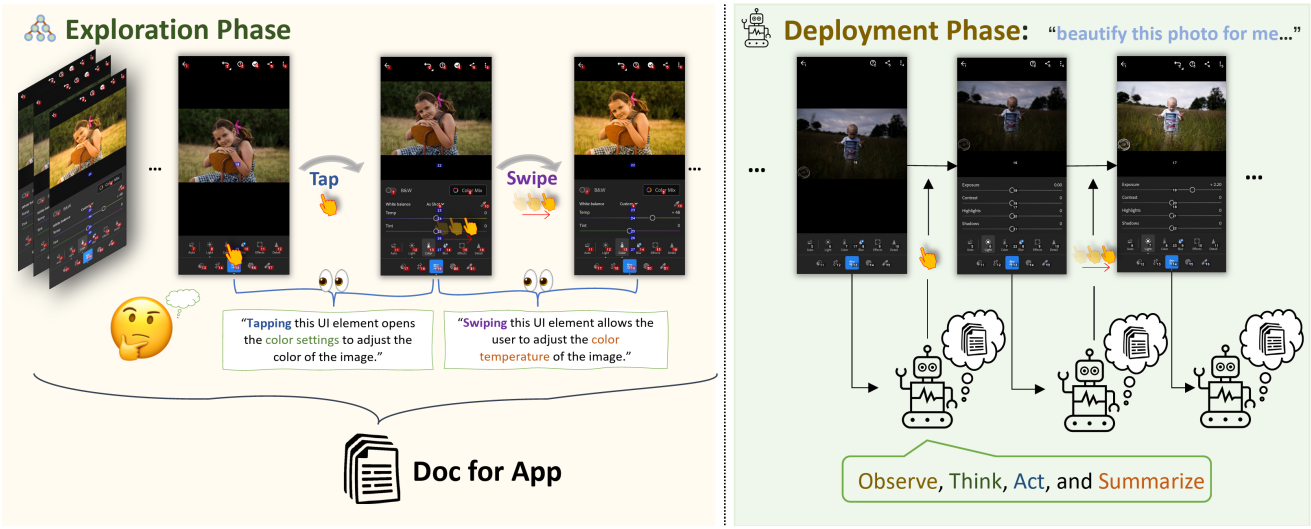

Figure 2: Overview of our multimodal agent framework designed to operate smartphone applications. The figure illustrates the two-phase approach of our framework. In the exploration phase, the agent interacts with a smartphone application and learns from their outcomes to create a comprehensive reference document. In the deployment phase, the agent utilizes the information compiled in this document to operate and navigate the apps effectively.

may include instructions such as "describe the functionality of the UI element concisely in one or two sentences by observing the differences between the two screenshots." As the agent explores, it compiles a document recording the effects of actions on different UI elements. When an element is operated on multiple times, the agent updates the document after each operation by combining past records with current observations, thereby improving accuracy over time.

Fig. 3 illustrates an example of how the document records functional descriptions of UI elements. As shown, the document contains concise summaries of the interactions with each UI component in JSON format. These descriptions guide the agent in making informed decisions during subsequent task executions in the deployment phase. This facilitates task generalization by providing reusable functional descriptions, even for unseen tasks.

During exploration, the agent can backtrack and correct its actions when it encounters unintended results. If a particular action led to an irrelevant page, such as an advertisement or a pop-up, the agent can use the system's back function to return to a more relevant screen. This capability is crucial for efficient task completion, ensuring that the agent doesn't waste time on non-essential UI elements. Compared to random search methods like Depth-First Search (DFS) or Breadth-First Search (BFS), our task-oriented exploration approach leverages the language model's knowledge of common UI patterns. This targeted exploration increases efficiency by focusing on elements that are likely to be crucial for the task at hand, avoiding unnecessary interactions with irrelevant components. The exploration phase concludes when the agent either completes the assigned task or reaches the maximum number of allowed exploration steps. In Appendix A, we provide the detailed prompts used by the agent to guide action selection, understand UI functionality, and handle backtracking.

Exploration through observation of demonstrations. An alternative exploration methodology, which has often proven more effective in our experiments, involves the agent observing and learning from human demonstrations. These demonstrations serve as a valuable resource for the agent, offering exemplary instances of efficient application usage. These instances are particularly valuable for understanding complex functionalities that may be challenging to discover through independent interactions. In this method, a human user operates the applications, while the agent assumes the role of an observer. The agent's primary responsibility during these demonstrations is to discern the functions of UI elements. This process is similar to auto-exploration in that both aim to summarize documents based on UI changes. The only difference lies in the fact that auto-exploration decides where to click and observes the changes independently, while human demonstrations provide a more directed learning experience for the agent. It's worth noting that the document only records the functions of UI elements, not the sequence of actions. Therefore, during deployment, the agent effectively avoids overfitting to specific tasks or operational sequences demonstrated by humans. This strategy effectively reduces the agent's exploration space. By focusing only on the elements and actions used by the human user, the agent can avoid navigating through irrelevant application pages. This makes the exploration process more streamlined and efficient, eliminating unnecessary steps and interactions. As a result, the agent can quickly grasp the essential functionalities of the application, reducing the time and effort required for comprehensive exploration.

3.3 Deployment Phase

After completing the exploration phase, the agent is well-prepared to handle complex tasks using the knowledge it has gained. When

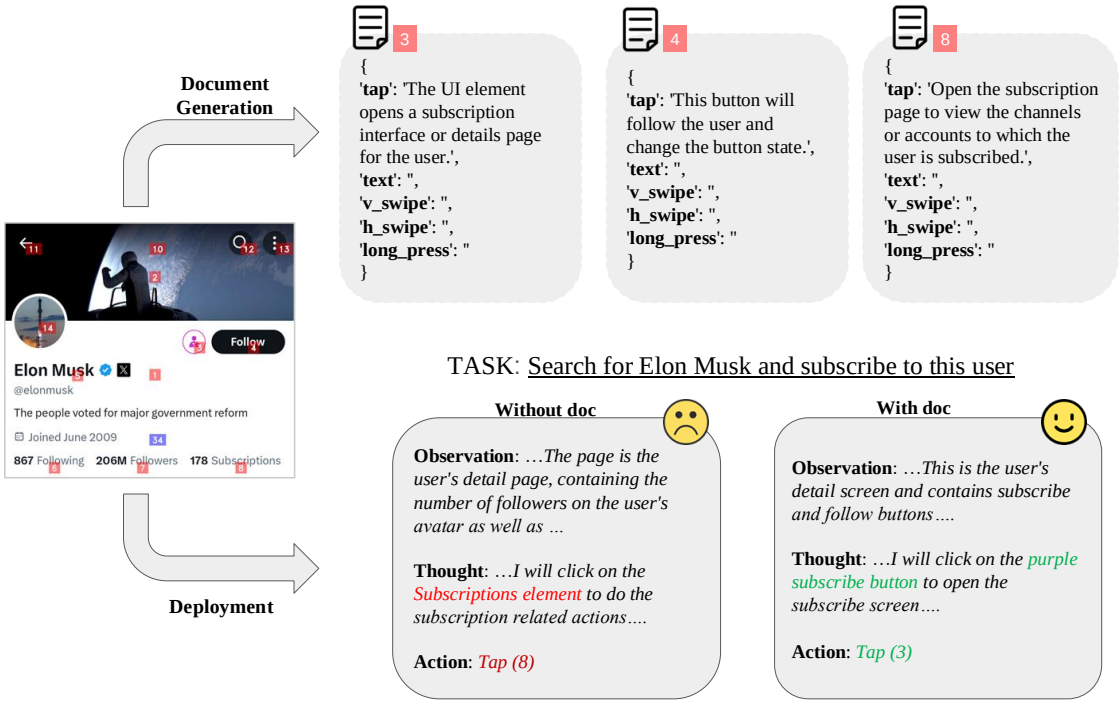

Figure 3: Examples of generated documents. These documents contain the functionalities of various UI elements and are dynamically incorporated into the prompts to enhance decision-making during deployment.

presented with a task, the agent follows a detailed, step-by-step approach. At each step, the agent receives a screenshot of the current user interface (UI) along with a dynamically updating document as input for the LLM. This document describes the functionalities of the UI elements on the current UI page and the effects of specific actions. We also add explanations of all possible actions into the prompt, as described in Section 3.1, to assist the agent in making informed decisions.

The agent has four main responsibilities at each step. Initially, it must observe the current UI by analyzing the screenshot and the dynamically generated document to understand the task and its context. Following this, the agent is required to articulate its thought process concerning the task in light of the observations it has made. Then, the agent proceeds to execute actions by invoking the appropriate action functions available to it. Finally, the agent is asked to summarize the interaction history. This summarized information is subsequently incorporated into the prompt in the next step, thereby providing the agent with a form of memory that allows it to learn from past interactions and make more informed decisions in the future. This systematic approach greatly augments the reliability and interpretability of the agent's actions. Consequently, it helps facilitate more informed and effective decision-making by the agent. The process ends when the agent determines that the task is completed. At this point, the agent can terminate the process by invoking the `Exit()` action. This action signifies the completion of the task, which marks the task's completion.

4 Experiments

In this section, we will present our evaluation of the multimodal agent framework through a combination of quantitative and qualitative experiments. Our primary goal is to assess the agent's performance and its ability to operate a diverse set of smartphone apps effectively.

4.1 Experimental Setup

As our work proposes an exploration-deployment paradigm rather than training a novel multimodal model, many existing benchmarks, such as those focusing on UI understanding evaluation, are not applicable to the evaluation of our framework. Therefore, we directly test our framework in real smartphone application environments, which present significant challenges due to distracting pages and special cases, such as advertisement pages, that can hinder the agent's performance. To comprehensively evaluate our method, we construct a benchmark that includes 10 popular apps, each serving various purposes. These apps include Google Maps, Twitter, Telegram, YouTube, Spotify, Yelp, Gmail, TEMU, Clock, and Lightroom. We have intentionally chosen this diverse set of apps to test the agent's adaptability across various functions and interfaces. In particular, to gain a more comprehensive insight into the vision capabilities of our agent, we conducted an in-depth case study using Adobe Lightroom, an image-editing application. This specific case study allowed us to evaluate the agent's proficiency in handling visual tasks and its ability to interpret and manipulate images within the app. For exploration, we prepared a set of tasks that are similar

but not identical to the evaluation tasks, allowing the model to learn effectively. These tasks are designed to resemble the test tasks in nature, ensuring the model encounters a variety of UI elements and functionalities during exploration. For example, if the testing task is to navigate to a subway station in Google Maps, an exploration task might involve navigating to a different location. The task details are provided in Appendix A. It's important to note that the specific operations performed during exploration are not recorded—only the inferred functionalities of UI elements are documented. This approach prevents data contamination and ensures the integrity of the test phase. For these experiments, unless otherwise specified, we utilized the state-of-the-art multimodal LLM, GPT-4. GPT-4 is designed to effectively process interleaved image and text inputs. This unique capability enables our agent to seamlessly interpret and interact with both visual and textual information within the apps.

4.2 Ablative Analysis

Baselines. To comprehensively evaluate our multimodal agent framework, we considered various design choices and their impact on performance. We conducted experiments using different configurations to provide valuable insights into the agent's behavior. We started with GPT-4 without any reference documents during testing and examined its performance both with the raw action API in the command-line interface and our simplified action space. Next, we explored different ways to generate guiding documents for the agent. These included documents generated through autonomous exploration, watching human demonstrations, and the manually crafted document as an oracle benchmark. Specifically, manually crafted oracle documents are refined versions of UI documents generated during human demonstrations. Experts review and optimize these to ensure precise and complete functional descriptions in the doc. In the autonomous exploration phase, we capped the maximum number of steps at 40, while during testing, we limited it to 20.

Metrics. To effectively compare the performance of different methods, we employed three human-evaluated metrics:

- **Successful Rate (SR):** This metric measures the average rate at which the agent successfully completes tasks within an app. If the agent fails to finish the task in 10 steps, it is considered a failure. As the tasks we created have clear evaluation criteria, the operation process and results are manually checked to determine task completion.
- **Reward:** A fine-grained reward strategy was implemented to measure partial task completion. Each UI page encountered during a task was assigned a score based on its proximity to the objective, ensuring a detailed assessment of the agent's progression. Specifically, for each step that matched the expert demonstration's ground-truth sequence, the agent received a reward point. This scoring system provided a nuanced evaluation of the agent's ability to follow expected actions, even if the task was not fully completed. The maximum reward for a task corresponds to the total number of steps in the expert demonstration. While expert demonstrations provide a ground-truth sequence, it is important to acknowledge that multiple valid approaches may exist for completing a given task.
- **Normalized Reward (NR):** To ensure equal contribution from all apps, this metric normalized the reward by dividing it by the total number of ground-truth steps of each app. Average normalized rewards across all apps were reported, mitigating bias toward complex tasks.
- **Average Steps:** We also reported the average number of steps required to successfully finish tasks across the selected apps.

Results. The comparison of our experimental results is presented in Table 1. We report the average performance of 45 tasks on 9 of the 10 previously described apps. Notably, we excluded Lightroom from this evaluation, as assessing task completion in this application presented inherent ambiguities. As demonstrated, our simplified action space significantly improves the performance of the GPT-4 baseline. Our observations indicate that LLM struggles with producing accurate xy coordinates, while our simplified action space eliminates this challenging requirement (2.2% *vs.* 48.9%). Additionally, documents generated through autonomous exploration and observing human demonstrations proved to be highly effective. Their results consistently outperformed the GPT-4 baseline with large improvements and are comparable to the results of human-written documents, which highlights the efficacy of our design in enhancing the agent's performance across a diverse set of apps. This also indicates that the performance of the agent can be further improved by manually improving the quality of the documents, *i.e.*, the accuracy of the element functionality descriptions recorded in the document, which also allows iterative optimization of the agent

Analysis of Task Complexity. To further investigate the agent's performance, we analyzed its ability to handle tasks of varying complexity. In particular, we compare the success rates (SR) of the baseline and document-assisted methods across tasks with different ground-truth step requirements. We used documents generated via autonomous exploration in this experiment for a fair comparison. Given the limited number of tasks at certain step levels in our benchmark, we repeated each experiment five times to mitigate the effects of randomness and ensure statistical robustness. The results, shown in Fig. 4, reveal that the performance of all methods declined as task complexity increased. The document-assisted methods consistently outperformed the baseline, with the performance gap widening as the number of task steps grew.

Qualitative results. In Fig. 5, we present a collection of examples that illustrate the agent's process of executing a variety of tasks. The purpose of this qualitative analysis is to offer a snapshot of the agent's capabilities – its capacity to accurately perceive the task at hand, reason about the appropriate course of action, and then act accordingly. These examples highlight not only the agent's ability to understand and interpret the tasks but also its proficiency in carrying out the corresponding actions effectively.

4.3 Comparison with other works

To better demonstrate the superiority of our agent, we conducted a comparison with MobileAgent [35]. MobileAgent is also a multimode agent for smartphone control, built upon AppAgent. Instead of designing action space based on the XML file, it employs a vision-based method for operating elements on the UI. Since the comparison here focuses on action space design, we do not use

Table 1: Evaluating Design Choices in AppAgent Performance. This table contrasts different design elements within AppAgent. Key findings include: our custom-developed action space surpasses the raw action space in efficiency; the exploration phase, incorporating both autonomous interaction and observation of human demonstrations, significantly enhances agent performance; and the auto-generated documentation yields outcomes on par with those derived from manually crafted documents.

Method	Document	Action Space	SR ↑	Reward ↑	NR ↑	Avg. Steps
GPT4 (Baseline)	None	Raw	2.2%	0.6	0.10	4.0
	None	**Ours**	48.9%	3.5	0.60	6.9
AppAgent	**Auto. Exploration**	**Ours**	73.3%	5.1	0.86	4.4
	Watching Demos	**Ours**	84.4%	4.7	0.87	5.1
	Manually Crafted	**Ours**	95.6%	5.5	0.93	5.5

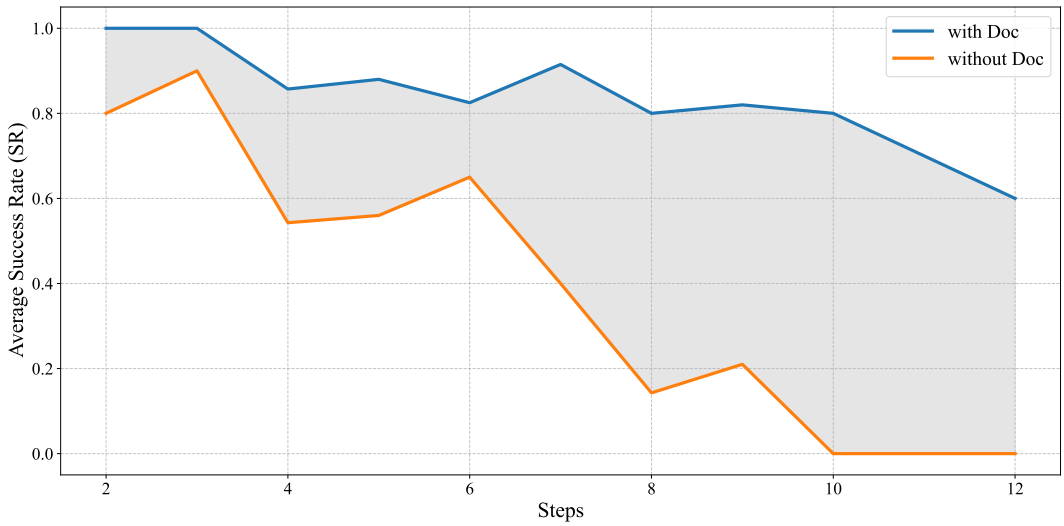

Figure 4: textbfAnalysis of Task Complexity. This figure illustrates the change in performance across tasks of varying complexity, measured by the ground-truth steps. The experimental results demonstrate that as task complexity increases, the advantage of incorporating documents becomes more significant.

documents in this experiment for a fair comparison, and therefore no exploration is conducted.

We reused the metrics defined earlier for this experiment. Additionally, app-specific results under these metrics are presented to provide a more fine-grained analysis of task execution capabilities.

For this comparison, we selected five representative apps, each offering different functionalities, to serve as case studies. As the result shows in Table 2, AppAgent consistently outperforms MobileAgent across all key metrics. One of the main limitations of MobileAgent is its reliance on vision-based models to detect icons and textual information, which causes difficulties when dealing with apps unfamiliar to LLMs. This is evident in cases where MobileAgent struggles to correctly identify key elements, leading to incomplete task executions. In contrast, AppAgent allows for more precise task localization and execution. This results in higher overall task completion and greater efficiency. Furthermore, it has been shown in Table 1 that AppAgent's in-context learning strategies can further significantly improve the performance over the baseline that is solely based on LLM's prior knowledge. In summary, AppAgent's advanced techniques make it a more versatile and effective agent for task execution across diverse apps, offering a notable advantage over vision-based approaches alone.

4.4 Evaluation on Long Tasks.

To further assess the performance of our agent on more complex tasks, we conducted an evaluation focused on long tasks, where each task is divided into three to four smaller sub-tasks. This setting enables us to track the agent's progress at each step and gain insights into its ability to handle tasks that require multiple stages of interaction. Long tasks are especially important for understanding how well the agent can maintain coherence over extended periods of interaction. For this experiment, we introduced several additional state-of-the-art LLMs as baselines, including Claude 3 Sonnet and Gemini 1.5 Pro. These models were chosen to provide a competitive comparison against our framework, as they are known for their advanced multimodal capabilities. To ensure a fair comparison, we only used documents generated through autonomous exploration

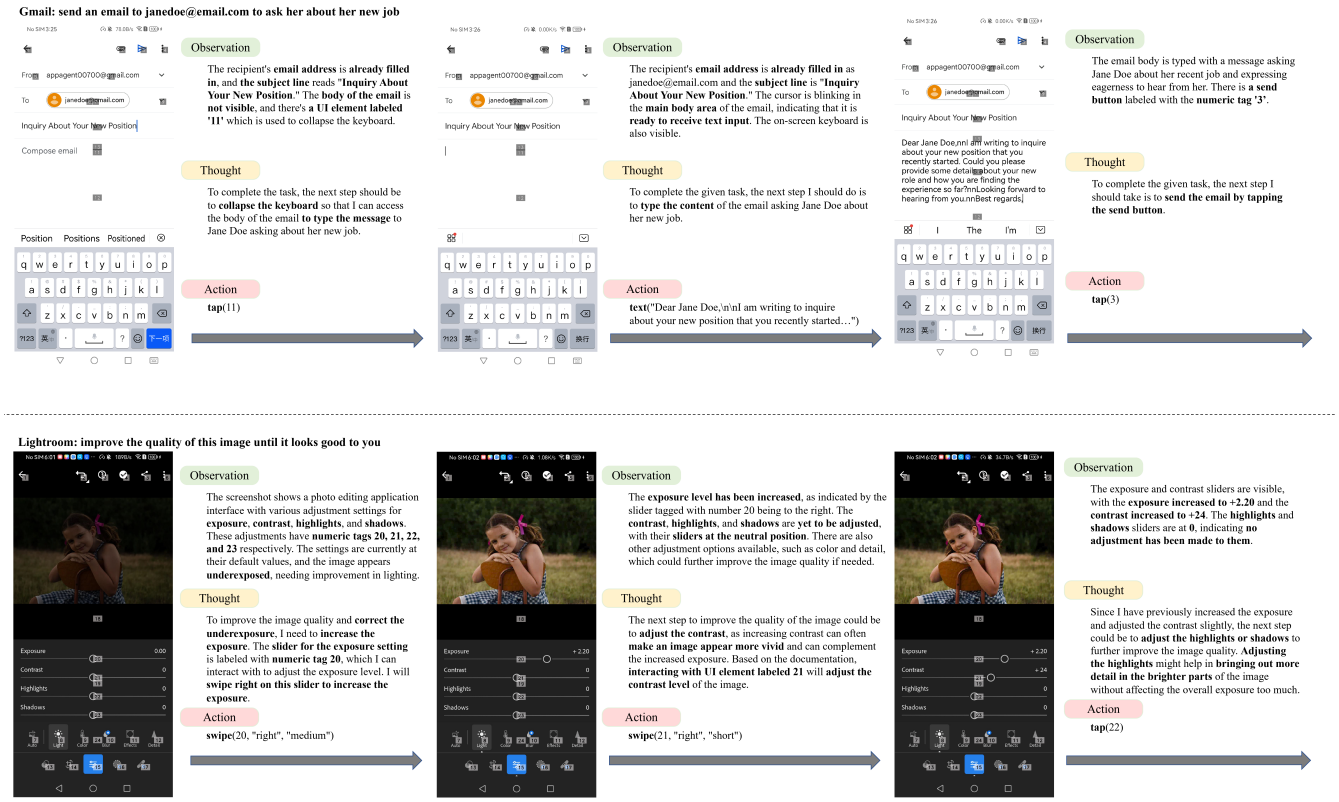

Figure 5: Qualitative Task Evaluation. This figure presents qualitative results for two distinct tasks conducted on Gmail and Lightroom. It showcases AppAgent's ability to accurately perceive, reason, and execute tasks, demonstrating its competence in various application contexts. Due to space constraints, some less critical details have been omitted from the description.

Table 2: Comparision of MobileAgent and AppAgent. AppAgent consistently outperforms MobileAgent across all metrics.

Task	X			Temu			YouTube			Yelp			Clock		
	Reward	NR	SR	Reward	NR	SR	Reward	NR	SR	Reward	NR	SR	Reward	NR	SR
MobileAgent															
Task 1	2/4	0.50	0	5/5	1.00	1	1/8	0.13	0	8/8	1.00	1	6/12	0.50	0
Task 2	2/7	0.29	0	3/8	0.38	0	6/6	1.00	1	7/7	1.00	1	4/4	1.00	1
Task 3	2/5	0.40	0	1/3	0.33	0	1/5	0.20	0	6/6	1.00	1	1/5	0.20	0
Task 4	1/6	0.17	0	1/5	0.20	0	1/4	0.25	0	4/10	0.40	0	3/6	0.50	0
Task 5	1/4	0.25	0	3/7	0.43	0	1/5	0.20	0	2/4	0.50	0	8/8	1.00	1
Avg	1.6/5.2	0.32	0	2.6/5.6	0.47	0.2	2.0/5.6	0.36	0.2	5.4/7.0	0.78	0.6	4.4/7.0	0.64	0.4
AppAgent (Ours)															
Task 1	4/4	1.00	1	5/5	1.00	1	8/8	1.00	1	8/8	1.00	1	5/12	0.42	0
Task 2	7/7	1.00	1	8/8	1.00	1	6/6	1.00	1	7/7	1.00	1	4/4	1.00	1
Task 3	5/5	1.00	1	1/3	0.33	0	5/5	1.00	1	6/6	1.00	1	5/5	1.00	1
Task 4	2/6	0.33	0	5/5	1.00	1	4/4	1.00	1	10/10	1.00	1	2/6	0.33	0
Task 5	3/4	0.75	0	5/7	0.71	0	3/5	0.60	0	4/4	1.00	1	8/8	1.00	1
Avg	**4.2/5.2**	**0.82**	**0.6**	**4.8/5.6**	**0.81**	**0.6**	**5.2/5.6**	**0.92**	**0.8**	**7.0/7.0**	**1.00**	**1.0**	**4.8/7.0**	**0.75**	**0.6**

for our agent's actions, maintaining consistency with our previous experiments. Additionally, we evaluated a variant of our AppAgent model, where the agent's exploration phase was based on similar sub-tasks to the ones in the long tasks. During deployment, the agent relied on a consolidated document, summarizing UI knowledge gained from these sub-tasks, to complete the full task. This

Table 3: Evaluation of Agent Performance on Long Tasks. This table demonstrates the effectiveness of our approach in enhancing performance on long tasks. We show that incorporating our exploration mechanism significantly improves the performance of various state-of-the-art LLMs. Additionally, exploring across different sub-tasks helps the agent better manage complex tasks, leading to improved overall performance.

Method	Document	SSR (%) ↑	SR (%) ↑
Gemini 1.5 Pro	None	45.6	4
	Sub-task Exploration	**67.0**	**32**
	Full-task Exploration	63.7	**32**
Claude 3 Sonnet	None	55.0	16
	Sub-task Exploration	**92.3**	**72**
	Full-task Exploration	70.0	36
GPT-4o	None	44.7	8
	Sub-task Exploration	**87.3**	**56**
	Full-task Exploration	75.3	40

Table 4: Case study on image editing tasks with Lightroom App. We conduct a user study to rank the image editing results of different methods. Our agents produce better results than the GPT-4 baseline.

Method	Document	Action Space	Avg. Rank ↓	Num. Tools
GPT4 (Baseline)	None	**Ours**	2.30	2.4
AppAgent	**Watching Demos**	**Ours**	1.95	5.8
	Manually Crafted	**Ours**	1.75	4.0

variant was introduced to test whether the agent's learning from multiple individual sub-tasks could enhance its performance on unseen, more complex tasks. This experiment involved five long tasks from five different apps, with up to 21 steps required to finish, as demonstrated by experts. During both exploration and deployment phases, we capped the maximum number of steps at 50. To minimize the effects of randomness in long task performance, each experiment was repeated five times to account for variability and ensure robustness in the results. The specific tasks and their details can be found in Appendix A. In addition to the previously used metrics, Successful Rate (SR), we introduced a new metric: Averaged Sub-task Successful Rate (SSR) that calculates the averaged successful rate of all sub-tasks in the long tasks. The SSR was designed to observe the model's performance at each stage of the long task, offering insight into how well the agent performs on intermediate steps, even if the final task completion was unsuccessful.

Results. As shown in Table 3, relying on LLMs alone led to poor overall results. Specifically, for the Successful Rate (SR) metric, all LLMs performed poorly. Despite this, they achieved close to 50% success on the individual sub-tasks. This disparity highlights the challenge LLMs face when tasked with completing complex tasks that involve multiple stages of interaction.

The introduction of documents significantly improved the performance of the LLMs across all metrics. Notably, documents generated through sub-task exploration learning led to even better performance. A potential explanation for this is that directly exploring the full-length tasks may make it difficult for the agent to successfully complete the task, as subsequent sub-tasks may depend on UI pages that were not encountered during the exploration phase. In contrast, the agent's performance is improved when it first explores smaller, more manageable sub-tasks, allowing it to encounter relevant UI pages and accumulate knowledge that can be leveraged during task deployment. The effectiveness of sub-task exploration in enhancing performance on long tasks further demonstrates the generalizability of our method. It illustrates that experiences gained through exploration of simpler, independent tasks can be transferred and applied effectively to more complex, multi-stage tasks.

4.5 Case Study

To gain deeper insights into the vision capabilities of our agent, we conducted an extensive case study using Adobe Lightroom, an image-editing application. This specific case study allowed us to evaluate the agent's proficiency in handling visual tasks, which was previously impossible for text-only agent models. Lightroom, as an image-editing app with various editing tools, demands a wide range of operations, such as selecting appropriate tools and manipulating image parameters. This case study provides a robust evaluation of the agent's overall capabilities. Additionally, the open-ended nature of image editing tasks allows us to assess the agent's problem-solving abilities. We prepared five images with common visual issues, such as low contrast and overexposure. Various variants of models, as previously illustrated, were used to edit these images. A user study with 10 participants, all researchers in the AI field with expertise in image processing, was conducted to rank the editing results produced by different methods. Participants were asked to rank the images according to their understanding of image issues and aesthetic quality of the edited images. The average ranks from all participants were then computed to provide a comprehensive

evaluation of the editing quality. We also reported the average number of tools used for image editing, providing an additional reference to the editing process's complexity. The images used for the evaluation can be found in Appendix A. All models were assigned the task of "fix this image until it looks good to you" without specifying the image's problems. The comparison of the results is presented in Table 4. As we can see, our agent model with documents yields consistently better results than the GPT-4 baseline, which emphasizes the influence of documents in our design. The generated documents by watching the demonstration produced comparable results with the results of manually crafted documents, which suggests the effectiveness of the exploration phase. We also find that with a document, the agent tends to use various tools to improve the image quality, while the GPT-4 baseline uses fewer tools.

5 Limitation and Future Work.

The current design of AppAgent supports a simplified action space, which excludes advanced controls like multi-touch and irregular gestures. This limitation could restrict the agent's effectiveness in more complex app interaction scenarios, especially when handling applications that require intricate and nuanced gestures. For example, professional applications such as CAD modeling tools, which rely heavily on specialized multi-touch gestures and offer highly specialized interfaces, present challenges for our current design. The absence of multi-touch support and the reliance on basic gesture controls may render AppAgent less effective in these contexts, limiting its ability to provide seamless interaction.

Additionally, AppAgent's reliance on XML files for capturing app interfaces sometimes leads to the omission of elements, especially those embedded within web pages. This constraint highlights the need for more advanced parsing and documentation methods to ensure the accurate representation of app interfaces. Furthermore, highly dynamic apps, such as games, where the UI elements are constantly changing, are likely to pose significant challenges for AppAgent. The rapidly evolving nature of these apps, coupled with their often unpredictable interface modifications, makes it difficult for the agent to adapt in real time without retraining or manual updates to its documentation.

To address these limitations, we propose two main directions for future work. First, enhancing the action space to support advanced gestures, such as multi-touch and irregular interactions, will significantly improve AppAgent's ability to handle more sophisticated applications. Second, developing more refined parsing techniques that capture the full range of dynamic elements, particularly in web-based and game-like environments, is crucial. These improvements will help AppAgent better navigate and operate applications with complex and highly dynamic UIs, extending its usability to a wider array of real-world tasks.

6 Conclusion

In this paper, we have introduced a multimodal agent. The proposed approach is unique in that it completely bypasses the need for system back-end access, a feature that brings with it a multitude of advantages, including enhanced security, adaptability, and flexibility. Our exploration-based learning strategy allows the agent to quickly adapt to new applications with unfamiliar user interfaces, making it a versatile tool for various tasks. Our extensive experiments across various apps highlight our agent's ability to handle diverse high-level tasks and underscore its adaptability and learning efficiency.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do As I Can and Not As I Say: Grounding Language in Robotic Affordances. In *arXiv preprint arXiv:2204.01691*.

[2] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. Survey: Robot programming by demonstration. *Springer handbook of robotics* (2008), 1371–1394.

[3] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818* (2023).

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. 2022. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817* (2022).

[5] Sylvain Calinon. 2009. *Robot programming by demonstration.* EPFL Press.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[7] Allen Cypher and Daniel Conrad Halbert. 1993. *Watch what I do: programming by demonstration.* MIT press.

[8] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems* 36 (2024).

[9] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. 2024. WorkArena: How Capable are Web Agents at Solving Common Knowledge Work Tasks? *arXiv preprint arXiv:2403.07718* (2024).

[10] Meta FAIR, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. 2022. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science* 378, 6624 (2022), 1067–1074.

[11] Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. 2023. Multimodal Web Navigation with Instruction-Finetuned Foundation Models. arXiv:2305.11854 [cs.LG]

[12] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. arXiv:2307.12856 [cs.LG]

[13] Daniel Conrad Halbert. 1984. *Programming by example.* University of California, Berkeley.

[14] Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. 2023. ChartLlama: A Multimodal LLM for Chart Understanding and Generation. arXiv:2311.16483 [cs.CV]

[15] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI]

[16] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. CogAgent: A Visual Language Model for GUI Agents. *arXiv preprint arXiv: 2312.08914* (2023).

[17] Zhiting Hu and Tianmin Shu. 2023. Language Models, Agent Models, and World Models: The LAW for Machine Reasoning and Planning. *arXiv preprint arXiv:2312.05230* (2023).

[18] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649* (2024).

[19] Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024. Tree search for language model agents. *arXiv preprint arXiv:2407.01476* (2024).

[20] Yanda Li, Chi Zhang, Gang Yu, Zhibin Wang, Bin Fu, Guosheng Lin, Chunhua Shen, Ling Chen, and Yunchao Wei. 2023. StableLLaVA: Enhanced Visual Instruction Tuning with Synthesized Image-Dialogue Data. arXiv:2308.10253 [cs.CV]

[21] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023. Improved Baselines with Visual Instruction Tuning.

[22] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning.

[23] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv: 2308.03688* (2023).

[24] OpenAI. 2021. ChatGPT. https://openai.com/research/chatgpt.

[25] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[26] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–22.

[27] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924* (2023).

[28] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the Wild: A Large-Scale Dataset for Android Device Control. *arXiv preprint arXiv: 2307.10088* (2023).

[29] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. 2022. A generalist agent. *arXiv preprint arXiv:2205.06175* (2022).

[30] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace. In *Advances in Neural Information Processing Systems*.

[31] Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 2023. 3D-GPT: Procedural 3D Modeling with Large Language Models. *arXiv preprint arXiv:2310.12945* (2023).

[32] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[33] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[34] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[35] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158* (2024).

[36] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. 2023. JARVIS-1: Open-World Multi-task Agents with Memory-Augmented Multimodal Language Models. arXiv:2311.05997 [cs.AI]

[37] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *arXiv preprint arXiv:2409.07429* (2024).

[38] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2023. Empowering LLM to use Smartphone for Intelligent Task Automation. *arXiv preprint arXiv: 2308.15272* (2023).

[39] Jason Wu, Rebecca Krosnick, Eldon Schoop, Amanda Swearngin, Jeffrey P Bigham, and Jeffrey Nichols. 2023. Never-ending learning of user interfaces. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[40] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).

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

[42] 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. 2023. OpenAgents: An Open Platform for Language Agents in the Wild. arXiv:2310.10634 [cs.CL]

[43] Yuzhuang Xu, Shuo Wang, Peng Li, Fuwen Luo, Xiaolong Wang, Weidong Liu, and Yang Liu. 2023. Exploring large language models for communication games: An empirical study on werewolf. *arXiv preprint arXiv:2309.04658* (2023).

[44] An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, Zicheng Liu, and Lijuan Wang. 2023. GPT-4V in Wonderland: Large Multimodal Models for Zero-Shot Smartphone GUI Navigation. *arXiv preprint arXiv: 2311.07562* (2023).

[45] Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions. arXiv:2306.02224 [cs.AI]

[46] Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. 2023. The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421* (2023).

[47] Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. 2023. The Dawn of LMMs: Preliminary Explorations with GPT-4V(ision). *arXiv preprint arXiv: 2309.17421* (2023).

[48] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*.

[49] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414* (2022).

[50] Zhuosheng Zhan and Aston Zhang. 2023. You Only Look at Screens: Multimodal Chain-of-Action Agents. *arXiv preprint arXiv:2309.11436* (2023).

[51] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL]

[52] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406* (2023).

[53] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854* (2023).

A Appendix

A.1 Prompt Details

We provide detailed prompts that are integral to our agent framework in Fig 6, 7, 8, and 9. It is important to note that these prompts do not encompass all the prompts required by our agent framework. Here, we only present the most critical prompts related to the exploration phase and deployment. For comprehensive details and usage instructions, please refer to the source code in our supplementary material.

A.2 Evaluation Details

Table 5 provides the detailed 45 tasks we used to measure the capability of AppAgent on 9 different Android apps. Table 6 provides the details of the long tasks used in long-task evaluation. Fig 10 shows the image samples presented to participants in our user study, conducted using the Adobe Lightroom app.

Prompt for **Deployment**

You are an agent that is trained to perform some basic tasks on a smartphone. When given a smartphone screenshot along with reference documents, your primary directive is to derive actionable insights from the documentation provided. These documents are essential for understanding the functionalities of UI elements that may not be immediately apparent from the screenshot alone. Your actions should be primarily informed by these documents, with the current UI interface analysis serving as a secondary reference.

#Requirements for detailed actions
[Mark the specific operations and functions behind each action.]
1.tap(element: int)
2.text(text_input: str)
3.long_press(element: int)
4.swipe(element: int, direction: str, dist: str)
5.grid()

You also have access to the following documentations that describes the functionalities of UI elements you can interact on the screen. These docs are crucial for you to determine the target of your next action. You should always prioritize these documented elements for interaction:
[UI document]

The task you need to complete is to <task_description>. Your past actions to proceed with this task are summarized as follows: <last_act>
Now, given the documentation and the following labeled screenshot, you need to think and call the function needed to proceed with the task.
Your output should include three parts in the given format:

Observation: <Describe what you observe in the image>
Thought: <To complete the given task, what is the next step I should do>
Action: <The function call with the correct parameters to proceed with the task.
If you believe the task is completed or there is nothing to be done, you should output FINISH. You cannot output anything else except a function call or FINISH in this field.>
Summary: <Summarize your past actions along with your latest action in one or two sentences. Do not include the numeric tag in your summary>
You can only take one action at a time, so please directly call the function.

Figure 6: Prompt for Deployment. During deployment, detailed information and choices of actions are provided. This knowledge could be obtained by either understanding demonstration or exploring. Following the instructions and documentation, our agent could act through a thought-action chain, which breaks a complex problem into a series of simpler ones.

Prompt for **Self-exploration** (Part I)

You are an agent that is trained to complete certain tasks on a smartphone. You will be given a screenshot of a smartphone app. The interactive UI elements on the screenshot are labeled with numeric tags starting from 1.

#Requirements for detailed actions
[Mark the specific operations and functions behind each action.]
1.tap(element: int)
2.text(text_input: str)
3.long_press(element: int)
4.swipe(element: int, direction: str, dist: str)
5.grid()

The task you need to complete is to <task_description>. Your past actions to proceed with this task are summarized as follows: <last_act>
Now, given the documentation and the following labeled screenshot, you need to think and call the function needed to proceed with the task.
Your output should include three parts in the given format:

Observation: <Describe what you observe in the image>
Thought: <To complete the given task, what is the next step I should do>
Action: <The function call with the correct parameters to proceed with the task.
If you believe the task is completed or there is nothing to be done, you should output FINISH. You cannot output anything else except a function call or FINISH in this field.>
Summary: <Summarize your past actions along with your latest action in one or two sentences. Do not include the numeric tag in your summary>
You can only take one action at a time, so please directly call the function.

Figure 7: Prompt for Self-Exploration (Part I). In Part I of the self-exploration, the agent attempts to complete a task independently without referencing external information. Similar to the deployment phase, the agent is required to output observations, thoughts, actions, and summaries.

Prompt for **Self-exploration** (Part II)

I will give you screenshots of a mobile app before and after <action> the UI element labeled with the number <ui_element> on the first screenshot. The numeric tag of each element is located at the center of the element. The action of <action> this UI element was described as follows: <last_act> The action was also an attempt to proceed with a larger task, which is to <task_desc>. Your job is to carefully analyze the difference between the two screenshots to determine if the action is in accord with the description above and at the same time effectively moved the task forward.

#Requirements for detailed decisions
[Mark the specific requirements behind each decision]

1. BACK
If you think the action navigated you to a page where you cannot proceed with the given task, you should go back to the previous interface.
2. INEFFECTIVE
If you find the action changed nothing on the screen (screenshots before and after the action are identical).
3. CONTINUE
If you find the action changed something on the screen but does not reflect the action description above and did not move the given task forward.
4. SUCCESS
If you think the action successfully moved the task forward (even though it did not complete the task.

[General requirements for each decision]
Never include the numeric tag of the UI element in your description. You can use pronouns such as "the UI element" to refer to the element. Your output should be in the following format:

Decision: <Decision>
Thought: <explain why you think the last action is wrong and you should go back to the previous interface>
Documentation: <describe the function of the UI element>

Figure 8: Prompt for Self-Exploration (Part II). In Part II of the self-exploration, the agent aims to understand the functions of a UI element by analyzing screenshots taken before and after an action is performed. Additionally, the agent must determine whether the action is relevant to the given task.

Prompt for **Document Generation**
based on Human-Demonstration

Prompts for detailed action (take tap button as an example)
I will give you the screenshot of a mobile app before and after tapping the UI element labeled with the button <ui_element> on the screen. Tapping this UI element is a necessary part of proceeding with a larger task, which is to <task_desc>.
Your task is to describe the functionality of the UI element concisely in one or two sentences. Notice that your description of the UI element should focus on the general function.

Special requirements for action function generation
For example, if the UI element is used to navigate to the chat window with John, your description should not include the name of the specific person. Just say: "Tapping this area will navigate the user to the chat window". Never include the numeric tag of the UI element in your description. You can use pronouns such as "the UI element" to refer to the element.

Figure 9: Prompt for Document Generation based on Human-Demonstration. According to the given screenshots, the GPT-4V could understand the functionality of specific UI elements and generate documents. The generated documents could help with following steps when the model needs to decide what action should be taken to achieve specific targets.

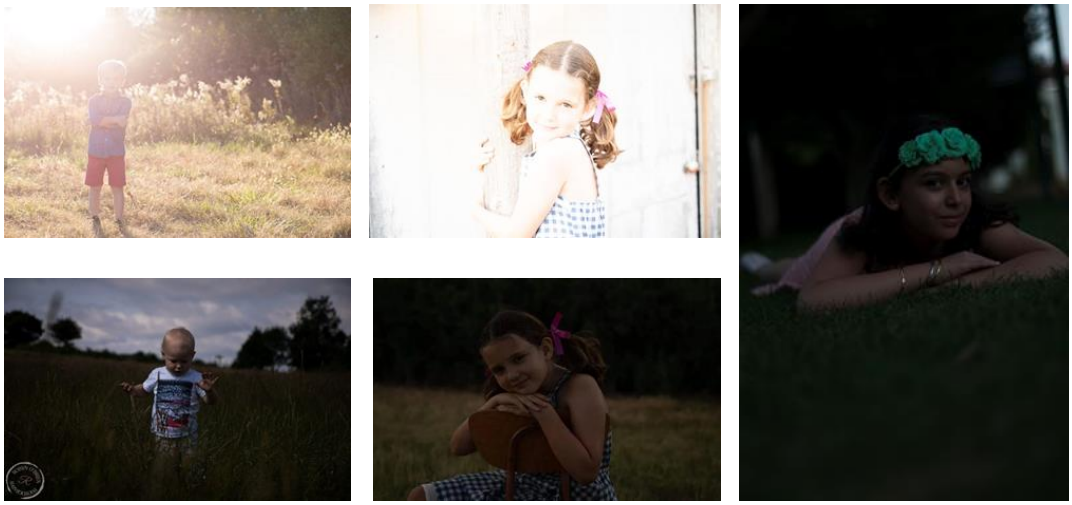

Figure 10: Image Samples Used in the User Study with Adobe Lightroom.

Table 5: 45 evaluation tasks we used to measure the capability of AppAgent on 9 different Android apps. The "Steps" column represents the ground-truth number of steps for each task, as demonstrated by experts.

App	Evaluation Task	Exploration Task	Steps
Google Maps	search for Tencent Shanghai Branch, get directions for driving and start navigation	search for Alibaba Hangzhou , get driving directions, and begin the route	4
	search for nearby restaurants, sort them by distance and get directions to the first one in the list	find nearby cafes, arrange them by distance, and navigate to the nearest one	5
	go to saved locations and get directions to home	access saved locations and route to my apartment	3
	search for Oriental Pearl Tower, get directions for walking and start navigation	search for Burj Khalifa, find walking directions, and start the walk	6
	search for nearby hotel rooms which can host 4 adults and check availability of one hotel	find nearby hotels that accommodate 3 adults and check availability	8
X	send a tweet post with the content "One Tencent, One Goal"	post a tweet saying "Hello World"	4
	send a tweet post with a random photo and text "This is for Elon"	tweet with a random picture and caption "This is it"	7
	search for the user Lebron James and follow him	search for the user Serena Williams and follow her	5
	go to my profile page and edit my profile name	visit my profile and update my name to "Cool"	6
	enter a community page and ask to join	visit a community page and request to join the group	4
Telegram	chat with the user AppAgent and answer his question	talk to user AppAgent and answer his inquiries	3
	go to my profile page, enter chat settings, and change a color theme	open profile settings, navigate to chat preferences, and choose a color theme	7
	change the app to dark mode	switch to night mode in the app	2
	go to my profile and set a profile picture	set a new avatar on my profile	5
	create a new group and add AppAgent to the group	form a new group and invite AppAgent to join	9
Temu	search for a gaming headset and add it to my shopping cart	search for a Bluetooth speaker and add it to my cart	5
	find my coupons and offers and apply a new one by entering coupon code "COUPON123"	find my coupons and apply the coupon code "COUPONTEST"	8
	remove all items from my shopping cart	clear all products from my shopping cart	3
	find the settings page and change my country from the UK to Australia	access the settings and change my country to Japan	5
	add a new credit card to my payment methods in the settings	add a new credit card to my payment methods	7
YouTube	search for the music video of the song "Wonderful Tonight" and leave a praising comment there	search for "Bohemian Rhapsody" video and leave a positive comment	8
	search for the channel "tesla" and subscribe to the channel	search for the channel "NASA" and subscribe	6
	go to the app setting and change the app language from English to Albanian	change the language setting to Finnish in the app	5
	find a video from my subscriptions and share it with my friend on Telegram	locate a video from my subscriptions and send it to a friend on Telegram	4
	find a video from my subscriptions and set the playback speed to 2x	pick a video from my subscriptions and adjust the playback speed to 0.5x	5
Spotify	search for the artist Eric Clapton, enter his artist page, and shuffle play his playlist	search for the artist Bob Dylan, enter his profile, and shuffle play his top tracks	8
	search the song "Smooth Criminal" and add it to my Liked Songs playlist	find "Billie Jean" and add it to my Liked Songs playlist	4
	change my profile name to AppAgent	update my profile name to "AI Enthusiast"	6
	go to the app settings and switch to save data mode	turn off data saver mode in settings	5
	add a new playlist to my library and name it "AI generation"	create a playlist titled "Future Beats" in my library	5
Yelp	search for restaurants nearby, filter to leave only the low-priced Chinese restaurants and sort them by distance	search for nearby vegan-friendly restaurants, filter by budget, and sort them by proximity	8
	search for restaurants nearby, filter to leave only the high-priced Pizza restaurants and sort them by rating	look for upscale Italian restaurants nearby, filter by price range, and sort by ratings	7
	search for restaurants, filter to leave only Mexican restaurants, view the first sponsored result, and find its open hours	search for fast food restaurants, filter to only show sponsored results, and check operating hours	6
	search for the restaurant "sakanaya" in Champaign and leave a 3 star review	look up "Sushi Spot" in Boston and leave a 3-star review	10
	find the list of restaurants that I recently viewed	check my recently viewed restaurants list in the app	4
Gmail	send an email to janedoe@email.com to ask her about her new job	email janedoe@email.com asking about her new career	6
	send an email to janedoe@email.com and cc johndoe@email.com. The email is about asking jane about her recent job interview	send an email to janedoe@email.com with a cc to johndoe@email.com, inquiring about his interview	8
	send an email that contains a random image file to janedoe@email.com	attach an image file and email it to janedoe1234@email.com	9
	send an email to janedoe@email.com to ask her about her new job. Schedule the send to tomorrow morning	schedule an email to janedoe@email.com about her job for tomorrow morning	6
	check my email drafts	review the saved drafts in my Gmail	3
Clock	set an alarm at 12:30 pm every Friday and Sunday, disable the vibration	set an alarm at 9:00 am every Monday and Thursday without vibration	12
	add a clock which shows the New York time	add a clock for Sydney, Australia	4
	open the settings to change clock style from digital to analog	access the settings to change the clock format from digital to traditional	5
	set an alarm at 6:15 am every Tuesday	set a wake-up alarm for 7:00 am every Wednesday	6
	go to bedtime to change my sleep sounds to Deep Space	go to bedtime mode and set my sleep sound to "Deep Space"	8

Table 6: Long tasks used for evaluation. The "Steps" column represents the ground-truth number of steps for each task, as demonstrated by experts.

Long Task	Sub Task	Exploration Sub Task	Exploration Long Task	Step
Add a new playlist called "llm" then search for the video author "3blue1brown". Follow him on his homepage and add his video list about putting llm related videos to the created video playlist.	1. Create a new playlist called "llm".	Generate a new video list named "AI Compilation."	Create a new playlist named "AI Compilation," search for the creator "Numberphile," subscribe to their channel, find videos related to generative AI from their content, and add these videos, as well as the currently playing video, to the "AI Compilation" playlist.	3
	2. Search for the author "3blue1brown" and follow.	Look up the creator "Numberphile" and subscribe to their channel.		4
	3. Find videos about llm on the author page and open it.	Locate videos about generative AI in the creator's content and view them.		3
	4. Add current video to llm playlist	Save the currently playing video to the "AI Compilation" library.		6
Set a route in the map, drive from UCLA to UCSB and save that path, and finally start this navigation.	1. Setting up a route in the map	Plan a walking route to a nearby park.	Plan a route to a nearby park, starting from a specified location and ending at the final destination, selecting public transportation as the preferred method. Save the itinerary, enable live traffic updates, and initiate navigation for the journey.	1
	2. Setting the Start and Finish Points	Specify the starting location and the final destination.		6
	3. Setting up automobile travel modes and bookmarking routes	Choose public transportation as the preferred travel method and save the itinerary.		9
	4. Start navigation with the current route	Initiate the route guidance and enable live traffic updates.		2
Open contacts in telegram, find the user AgentTester, send a test message to that user and set the chat wallpaper to a theme with a blue background.	1. Open contacts in telegram	Access the contact list in app.	Open the contact list in the app, locate the user "TesterBot," send them a greeting message, and change the conversation theme to a dark mode setting.	2
	2. Find the user AgentTester, send a test message.	Search for the user "TesterBot" and send a greeting message.		3
	3. Set the chat wallpaper	Change the conversation theme to a dark mode setting.		4
Send an email to janedoe@email.com, cc'd to same person. The email asks about Jane's most recent job interview and includes a photo from the album. Set this email in outbox as important.	1. Creating a new message and composing the message.	Draft a new email and include hello world in the text.	Send a new email that starts with "Test" to test.to@example.com, making sure to BCC test.cc@example.com. Then, move the email to the "Follow-Up" folder and mark it as high priority.	5
	2. Setting up senders and cc's	Add recipients in the "To" field and include a few in the "BCC" field.		4
	3. Set this email in your outbox as important it.	Flag the email as high-priority and move it to a "Follow-Up" folder.		4
Add a clock displaying New York time, then open the settings to change the clock style from digital to analog. Finally,Set an alarm for 12:30 PM every Friday and Sunday, ensuring the vibration is disabled.	1. Add a clock displaying New York time	Add a clock showing Tokyo's current time to the dashboard.	Add a clock displaying Tokyo's current time to the dashboard, open the clock preferences to switch the display format to 24-hour notation, schedule a reminder for 8:00 AM on Mondays and Thursdays, and configure notifications to disable sound while enabling only pop-ups.	3
	2. open the settings to change the clock style from digital to analog	Open clock preferences and change the display format to 24-hour notation.		5
	3. Set an alarm for 12:30 PM every Friday and Sunday	Schedule a reminder for 8:00 AM on Mondays and Thursdays.		9
	4. Ensuring the vibration is disabled	Disable sound notifications and ensure only pop-ups are enabled.		4