BENCHMARKING MOBILE DEVICE CONTROL AGENTS ACROSS DIVERSE CONFIGURATIONS

Juyong Lee[1] **Taywon Min**[2] **Minyong An**[3] **Changyeon Kim**[1] **Kimin Lee**[1]
[1]KAIST [2]Seoul National University [3]Yonsei University

ABSTRACT

Developing autonomous agents for mobile devices can significantly enhance user interactions by offering increased efficiency and accessibility. However, despite the growing interest in mobile device control agents, the absence of a commonly adopted benchmark makes it challenging to quantify scientific progress in this area. In this work, we introduce B-MoCA: a novel benchmark designed specifically for evaluating mobile device control agents. To create a realistic benchmark, we develop B-MoCA based on the Android operating system and define 60 common daily tasks. Importantly, we incorporate a randomization feature that changes various aspects of mobile devices, including user interface layouts and language settings, to assess generalization performance. We benchmark diverse agents, including agents employing large language models (LLMs) or multi-modal LLMs as well as agents trained from scratch using human expert demonstrations. While these agents demonstrate proficiency in executing straightforward tasks, their poor performance on complex tasks highlights significant opportunities for future research to enhance their effectiveness. Our source code is publicly available at https://b-moca.github.io.

1 INTRODUCTION

Autonomous agents controlling digital devices have great potential benefits. For example, these agents can improve the accessibility of user interactions, especially for users with physical disabilities or those facing challenges in operating devices, or boost productivity by automating tedious jobs. This leads to increased interest in developing agents for *mobile* device control, and diverse approaches have been introduced, including agents based on large language models (LLMs; Wen et al. 2023; Yan et al. 2023) and agents trained with human demonstrations (Sun et al., 2022; Li et al., 2023), toward assistive agents that can understand the screen layout of the devices and manipulate the user interface (UI) to follow human instructions.

Despite recent progress in developing mobile device control agents based on real systems, such as Android emulators (Toyama et al., 2021; Shvo et al., 2021; Zhang et al., 2023), prior works often overlook several important properties. One is testing generalization ability across diverse device configurations, which is crucial in deploying agents in real devices. Moreover, practical tasks essential for life (such as creating an alarm or making emergency calls) are often neglected because of the challenges in defining a wide range of practical tasks with robust success criteria in various device settings. The lack of a unified benchmark encompassing these important properties has impeded scientific progress in this field.

In this work, we introduce B-MoCA: a **B**enchmark designed for evaluating **Mo**bile device **C**ontrol **A**gents across diverse configurations, based on Android emulators (see Figure 1). A key feature of B-MoCA is supporting numerous customization to mirror diverse device configurations,including variations in icon placements, sizes, wallpapers, languages, and device types. Utilizing this feature, users can easily create diverse environments with various configurations to evaluate generalization ability. Additionally, we define 60 practical tasks grounded in realistic scenarios, such as opening specific applications, initializing searches over the web, and adjusting device settings. To ensure reliable evaluation, B-MoCA provides rule-based success detectors, which are based on pre-defined task completion criteria.

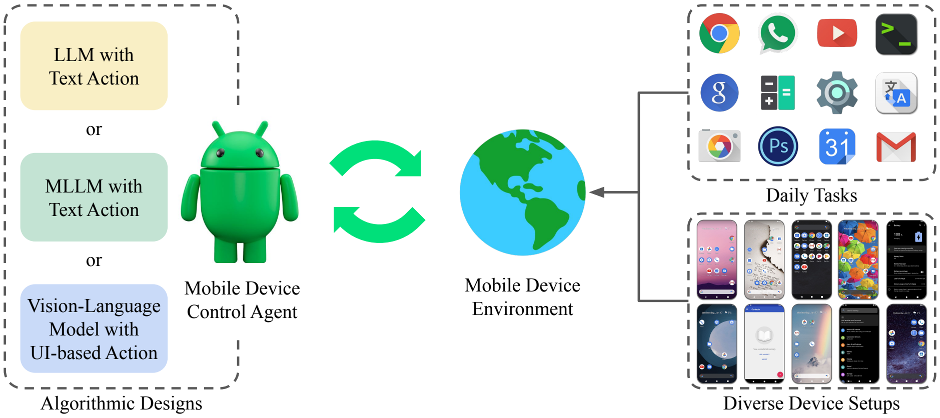


Figure 1: Illustration of B-MoCA. We present a realistic benchmark for assessing the performances of mobile device control agents in executing everyday tasks. To analyze generalization ability, we introduce a randomization feature that changes various device attributes. We benchmark agents leveraging LLMs or MLLMs as well as agents with vision-language models trained from scratch.

We benchmark various methods for building mobile device control agents in B-MoCA. The baselines include agents employing text-only large language models (LLMs) or multi-modal LLMs (MLLMs), which benefit from extensive knowledge obtained through pre-training. We consider both closed-source models, such as GPT-4 (Achiam et al., 2023) and Gemini (Gemini et al., 2023), and open-source models, such as Llama 2 (Touvron et al., 2023) and Llama 3. Additionally, we train agents from scratch that directly interact with device UIs using behavior cloning (BC; Pomerleau 1988).

In our experiments, we find that the agents exhibit fundamental skills in mobile device control, such as solving straightforward tasks or completing tasks in training environments. However, they struggle in more challenging scenarios, such as handling more difficult tasks or generalizing to unseen device configurations. Specifically, the agents employing LLMs or MLLMs show high robustness across diverse device configurations, while they fall short on multiple sequential decision-making. Agents trained with BC, on the other hand, successfully mimic expert behaviors but lack generalization ability in test environments with unseen device configurations. We study the effect of different design choices on leveraging foundation models, including few-shot learning and the visual prompting method. We also analyze the effect of using pre-trained representation models or utilizing different numbers of training device environments while training agents from scratch. Our extensive analyses reveal the limitations of existing methods in mobile device control, calling for future research.

We open-source all the source codes and relevant materials for easy reproduction of our environments and experiments. We hope B-MoCA helps future researchers identify challenges in building assistive agents and easily compare the efficacy of their methods over the prior work.

2 B-MoCA

In this section, we introduce B-MoCA: a benchmark designed to evaluate the performance of mobile device control agents on diverse device configurations in executing common daily tasks.

2.1 Design Factors

To create a realistic benchmark for mobile device control agents, we build our benchmark based on Android, a widely used open-source operating system. In this benchmark, we frame device control as a sequential decision-making problem, reflecting the multi-step nature of the real interactions (Section 2.2). Designing a meaningful benchmark for mobile device control poses a significant challenge, particularly in defining practical tasks like opening applications or adjusting device settings. To address this, we consider 60 basic tasks that involve commonly used applications like Chrome and Calendar, ensuring relevance to everyday life. Each task is equipped with a success detector to evaluate the agent's performance in accurately completing the task (Section 2.3).

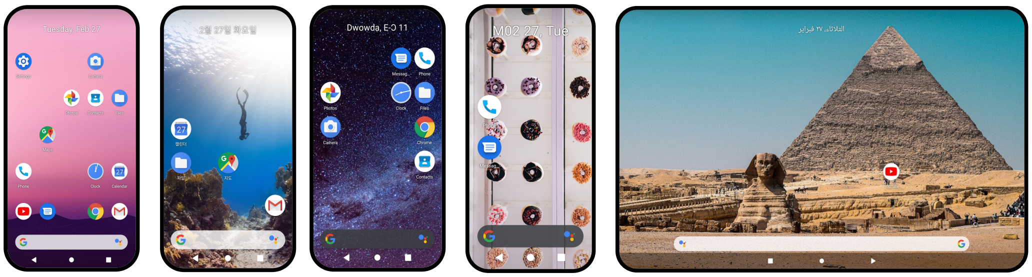


Figure 2: Examples of the home screen images from environments in B-MoCA. The randomized features span icon location, font size, wallpaper, language, and device type and challenge the generalization ability of agents.

Given the diverse nature of user mobile device setups, such as variations in icon placements, wallpaper choices, languages, and device types, it is important to test the generalization abilities of device-control agents across diverse setups. To assess generalization performance, we incorporate a randomization feature in our benchmark. This feature is designed to simulate various real-world scenarios by changing various aspects of mobile devices, such as user interface layouts and wallpapers (Section 2.4).

2.2 Problem Formulation

In B-MoCA, we formulate the device management task as a sequential decision-making problem, where an agent interacts with an environment. Formally, given a task instruction c, an agent receives an observation o_t and takes an action a_t based on its policy $a_t \sim \pi(\cdot|o_t, c)$ at each timestep t. The environment (i.e., Android emulator) returns a success signal r_t and the environment transitions to the next observation o_{t+1}.

Observations, which capture the UI elements, can be represented as either screen pixels, screen descriptions derived from the Android view hierarchy, or a combination of both. The action space comprises a dual-gesture, similar to Rawles et al. (2023), which consists of a pair of (x, y) screen locations for `touch` and `lift`. The dual-gesture action is identified as `tapping` the screen when the two locations are identical within a specified threshold or `swiping` the screen when the distance between the two locations exceeds this threshold. Additionally, the agent can press navigation buttons (i.e., back, home, and overview) by touching the corresponding button locations on the screen. We note that our benchmark supports text-based actions, enabling the utilization of the LLMs or MLLMs (see Section 3.1 for details).

We refer the readers for further details on the environment implementation to Appendix A.1.

2.3 Daily Tasks

Our B-MoCA includes 60 tasks essential for managing digital devices, providing functionalities useful in daily routines. Each task is designed to be grounded in realistic situations, such as setting the alarm or enabling airplane mode. The tasks span various applications and require agents to interact with diverse UI elements, such as application icons, checkboxes, toggle switches, input fields, and sliders. For a comprehensive list of tasks, we refer readers to Appendix B.1.

Task completion is determined by a rule-based success detector implemented using Android Debug Bridge (ADB). This success detector monitors logs from ADB and identifies the successful completion based on pre-defined criteria. These criteria are established by examining ADB logs from human demonstrations for each task and selecting the log produced when the target task is completed. With the pre-defined criteria, then, the success detector automatically finds the matching regular expression in the ADB logs to signal the task completion. The success signal is with the value of $+1$ when the task is completed, and 0 otherwise. An episode terminates as a success if the success detector signals completion, or as a failure if the agent exceeds a maximum step limit without meeting the criteria.

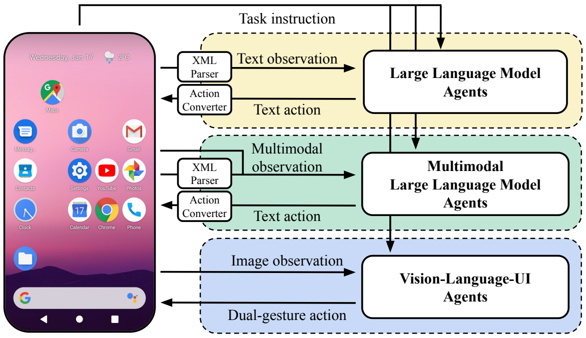


Figure 3: Illustration of baseline agents. LLM agents and MLLM agents interact with environments through additional XML parser and action converter, to obtain text descriptions and manipulate UIs with text actions. VLUI agents directly leverage the UIs with screen images and dual-gesture actions.

2.4 Environment Randomization

In mobile device control, developing agents that can generalize across various device setups is crucial. To evaluate their generalization ability, B-MoCA incorporates a randomization feature that changes icon placements and sizes, wallpapers, languages, and device types. Users can select the device type from a device list that includes popular models like Pixel 3, Pixel 4, Pixel 6, and WGXA Tablet. They can also specify the locales to set the language and region, choose wallpapers from a selection of custom images, and activate dark mode for further environmental variation. Moreover, the sizes of icons and text can vary between small, medium, and large. Lastly, applications can be randomly placed on the home screen to simulate real-world usage patterns.

Using randomization features, we create 45 unique environments in B-MoCA, with examples shown in Figure 2. To assess the generalization ability, we divide the 45 distinct environments into two sets: 35 for training and 10 for testing. We employ domain randomization (Tobin et al., 2017) to train agents, enabling them to perform tasks robustly across diverse device configurations. We then evaluate the performance on test environments, which include unseen device setups. A detailed list of environment device configurations we prepare is available in Appendix A.2.

3 Baselines

In this work, we benchmark various approaches for building mobile device control agents: LLM agents, MLLM agents, and Vision-Language-UI (VLUI) agents (see Figure 3). LLM agents and MLLM agents are developed using foundation models like LLMs and MLLMs, respectively (Section 3.1). VLUI agents, which consist of vision-language encoders, are trained from scratch using human expert demonstrations (Section 3.2).

3.1 LLM Agents and MLLM Agents

Utilizing foundation models such as LLMs and MLLMs, which contain extensive knowledge and have emergent capabilities, becomes a major direction in developing mobile device control agents (Wen et al., 2023; Yan et al., 2023). In this work, we benchmark two types of agents that employ different foundation models: LLMs (e.g., GPT-4) and MLLMs (e.g., GPT-4V). LLM agents utilize only the text descriptions of the screen layout to generate text actions, while MLLM agents process both text and visual inputs.

To facilitate the interactions of LLM and MLLM agents with an Android emulator, we define an XML parser (Zhang et al., 2023; Yang et al., 2023b). This XML parser converts the UI elements, from the Android view hierarchy of the screen presented in XML format, into a list of text descriptions. The description includes the location of the bounding box, if necessary. Additionally, we define a set of possible action options, as detailed in Table 1, that can be converted into a corresponding dual-gesture

Action option	Description
`dual-gesture(*)`	Operate a dual-gesture action with arguments `(*)`.
`tap(numeric tag)`	Tap UI element labeled with `numeric tag`.
`swipe(direction)`	Swipe to `direction`.
`press("HOME")`	Press home button.
`press("BACK")`	Press back button.
`press("OVERVIEW")`	Press overview button.

Table 1: A set of action options for text-based agents. Additional options are converted into corresponding dual-gesture actions.

Role: You are an agent that is trained to perform daily tasks on digital devices, such as smartphones [...]

Action space: You need to select an action option [...]

Goal: [...]

(Optional) **Few-shot examples**: [...]

Output format: Your output should follow the given format

- Description: Describe what you observe in the input
- Thought: To complete the given task, what is the next step
- Action: The function call with the correct parameters

Observation: [...]

Figure 4: An overview of prompt for the text-based agents, with abbreviated relevant information as [...]. The complete prompt is at Appendix C.1.

action.[1] These action options include tapping the UI element by choosing the numeric tags, swiping the screen in pre-defined directions (up, down, left, right), and pressing the button with the names.

With these text-based observations and actions, we prompt the foundation models to explain the agents' role, action space definition, goal, (optional) few-shot examples, and current observation. Our prompts, outlined in Figure 4, also incorporate the Chain-of-Thought technique (Wei et al., 2022) to enhance the reasoning ability of the agents by enforcing a certain output format.

3.2 VLUI Agents

Despite the promising results of LLMs, leveraging these foundation models presents several challenges such as the necessity of auxiliary interfaces or difficulties in fine-tuning. Thus, we also investigate another type of agent that can be trained from scratch: VLUI agents, named after the vision-language model with UI actions. Characterized by their direct interaction with device UIs in a human-like manner, these agents can significantly benefit from the easy incorporation of human demonstrations for training, potentially improving learning efficiency.

To be detailed, VLUI agents take a task instruction and screen images as the input and produce a dual-gesture action as the output. Input embeddings are extracted using vision and language encoders and a transformer (Vaswani et al., 2017) module is utilized to process these embeddings and generate the dual-gesture actions. Specifically, we train a deterministic multi-task policy $\pi_\theta(a_t|o_t, c)$ using BC (Pomerleau 1988; Schaal 1996). The parameters θ of the policies are optimized to imitate the human expert demonstrations $\mathcal{D} = \{(o_t, a_t^*, c)\}$ by minimizing the following objective with mean squared error function $L(\cdot)$:

$$\sum_{(o_t, a_t^*, c)\sim\mathcal{D}} L(\pi_\theta(a_t|o_t, c), a_t^*).$$

We refer readers to Appendix C.2 for more details on the architecture of VLUI agents.

4 Experiments

We design our experiments to investigate the following research questions:

- Can baseline agents perform daily tasks on mobile devices? (Section 4.2)
- What are the distinctive characteristics of each agent? (Section 4.2)
- What are the effects of different design choices for LLM or MLLM agents? (Section 4.3)
- How crucial is the pre-training or training data diversity for VLUI agents? (Section 4.4)

[1] To convert text actions to dual-gesture actions, we define the action converter. We analyze the efficacy of the action options in Appendix E.1.

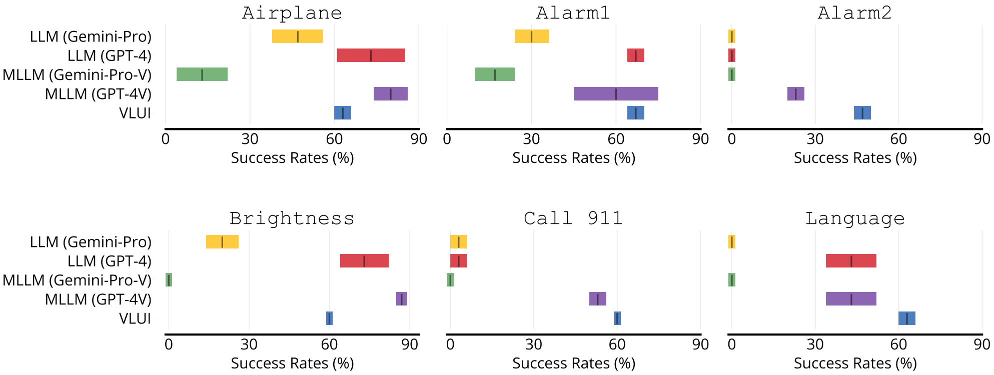


Figure 5: Average success rates of the baseline agents in the test environments. We report the mean and standard error across three runs. LLM agents are in three-shot learning, and MLLM agents are *without* SoM prompting and in one-shot learning for `Gemini-Pro-V` (due to maximum context length) or three-shot learning for GPT-4V. The text-based agents with GPT-4 or GPT-4V show the best performances on `Airplane`, `Alarm1`, and `Birhgtness`, while VLUI agents show better performances on `Alarm2`, `Call 911`, and `Language`.

4.1 Experimental Setup

In our experiments, we evaluate LLM agents, MLLM agents, and VLUI agents using six representative tasks: named, `Airplane`, `Alarm1`, `Alarm2`, `Brightness`, `Call 911`, and `Language`. These tasks are selected to cover navigating multiple pages in target applications and manipulating diverse UI elements which vary in configuration across different device settings. For example, on `Alarm2`, the agents need to reach the alarm tab in the clock application and adapt to varying shapes of clock UI in a shape of either rectangle or circle with different size options. We display exemplary expert demonstrations on these tasks in Appendix B.2. For each task, the task instruction is as follows:

- `Airplane`: "turn on airplane mode"
- `Alarm1`: "turn on alarm at 9 am"
- `Alarm2`: "create an alarm at 10:30 am"
- `Brightness`: "decrease the screen brightness in setting"
- `Call 911`: "call 911"
- `Language`: "go to the 'add a language' page in setting"

For LLM agents, we employ the closed-source models Gemini-Pro (Gemini et al., 2023) and GPT-4 (`GPT-4-0125-preview`; Achiam et al. 2023).[2] We study LLM agents with both zero-shot and few-shot learning cases. For few-shot learning, we sample examples from 210 human expert demonstrations (see Appendix D.1 for dataset collection). For MLLM agents, we leverage Gemini-Pro-V and GPT-4V (`GPT-4-vision-preview`). We report MLLM agents in only few-shot learning and investigate visually grounding the agents with Set-of-Mark (SoM) prompting (Yang et al., 2023a). We provide more details on the configurations for LLM and MLLM Agents in Appendix C.3. For VLUI agents, we train multi-task policies where each policy performs all six tasks. The policies are trained with BC using the 210 human expert demonstrations.[3] We refer the readers to Appendix C.4 for more details on the training procedures of VLUI agents.

For each evaluation, we measure the success rates of the agents in the 10 test environments and compute the average success rates. These success rates are automatically computed by the rule-based success detector. We report the mean and standard error across three different runs.

4.2 Main Results

Figure 5 shows the success rates of LLM agents, MLLM agents, and VLUI agents in test environments. LLM agents and MLLM agents utilize their pre-trained base knowledge and few-shot

[2] We include experiments with open-source models of Llama 2 (Touvron et al., 2023), Llama 3, and AgentLM (Zeng et al., 2023) in Appendix E.2.

[3] We also include experimental results of VLUI agents trained with offline reinforcement learning by employing the success signals as rewards in Appendix E.3.

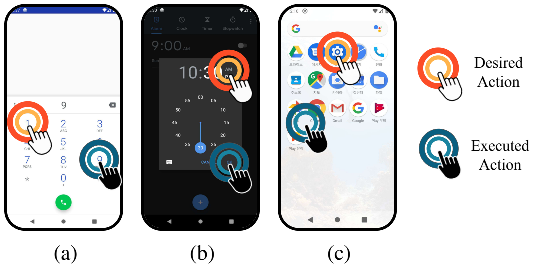


Figure 6: The common failure modes of the agents. (a) LLM agents fail to complete sequential steps, (b) MLLM agents miss details in the images, and (c) VLUI agents tap the wrong icon locations.

	LLM (zero-shot)	LLM (few-shot)	MLLM (w/o SoM)	MLLM (w/ SoM)
Airplane	53 ± 03	73 ± 12	80 ± 06	83 ± 03
Alarm1	42 ± 13	67 ± 03	60 ± 15	62 ± 09
Alarm2	00 ± 00	00 ± 00	23 ± 03	17 ± 03
Brightness	73 ± 12	73 ± 09	87 ± 03	83 ± 03
Call 911	00 ± 00	03 ± 03	53 ± 03	33 ± 09
Language	27 ± 06	43 ± 09	43 ± 09	47 ± 17

Table 2: Success rates of text-based agents with different prompting methods. While few-shot examples help LLM agents with GPT-4, we observe no significant gain from SoM prompting for MLLM agents with GPT-4V.

examples to complete simple tasks with high performances (e.g., more than 70% on Airplane and Brightness with GPT-4 or GPT-4V), but their success rates significantly drop as the tasks become complex (e.g., less than 30% on Alarm2 even with GPT-4 or GPT-4V). VLUI agents, on the other hand, imitate the behaviors of experts and exhibit average success rates of higher than 50% on all tasks, except 47% on Alarm2. However, all methods still show low performances (less than 60%) on complex tasks (i.e., Alarm2 and Call 911), which calls for new algorithms.

We provide more remarks on each agent type below.

Robustness of LLM agents and MLLM agents Both types of agents employing foundation models have shown robust performances in diverse device configurations. It is straightforward that these agents are robust to the randomization over the visual appearances, such as icon locations or font size, as the locations of the UI elements are described in the Android view hierarchy. In addition, LLM agents with both Gemini-Pro and GPT-4 are robust to language changes, with descriptions of UI elements in different languages. Particularly, these agents generalize to languages in test environments, e.g., Korean and Egyptian Arabic, which are not included in the few-shot examples.

Remaining challenges for LLM agents While exhibiting robust performances across diverse device settings, several limitations of LLM agents are observed. First, the agents face difficulties with long-horizon tasks, which require completing a precise sequence of multiple actions. For example, on Call 911, the agents often make mistakes while typing the sequence of 9-1-1, as shown in Figure 6(a). Second, the agents struggle to leverage few-shot examples adaptively. For instance, on Brightness, we observe LLM agents naively copying the few-shot examples from different device configurations without adjusting them to the current environment.

Efficacy of multi-modal input for MLLM agents We confirm the effectiveness of image input with MLLM agents employing GPT-4V, as large increases in success rates are observed on Alarm2 and Call 911 compared to LLM agents with GPT-4. However, MLLM agents share the challenges of LLM agents in accurately executing complex tasks. Moreover, they still fall short in understanding details of visual input, such as the small interface for setting AM/PM on Alarm2 as shown in Figure 6(b). MLLM agents with Gemini-Pro-V show significantly lower performances than LLM agents with Gemini-Pro, assumably due to the longer context length of multi-modal inputs. These results indicate the remaining headroom in leveraging multi-modal inputs more efficiently.

Generalization ability of VLUI agents We observe training VLUI agents with BC can lead to high performances on many complex tasks where MLLM agents fail. These agents perform robustly to unseen wallpapers, as being trained with multiple different background images. Also, they can generalize their actions to unseen devices, e.g., Pixel 4, even though they are trained only on a single device type, i.e., Pixel 3. However, VLUI agents begin failing to complete the tasks with severe visual changes induced by unseen device configurations. While they exhibit higher than 90% success rates in training environments, the performance degrades to less than 70% in test environments (see Appendix D.2 for more details). Specifically, they suffer from handling unseen locations of UI elements, as shown in Figure 6(c). We believe these findings reveal the importance of diversity in training data from randomized environments (see Section 4.4 for more discussions).

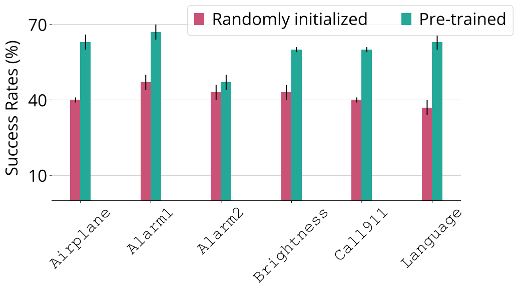


Figure 7: Success rates of VLUI agents with visual encoders randomly initialized or pre-trained. Pre-training helps the performances of the agents.

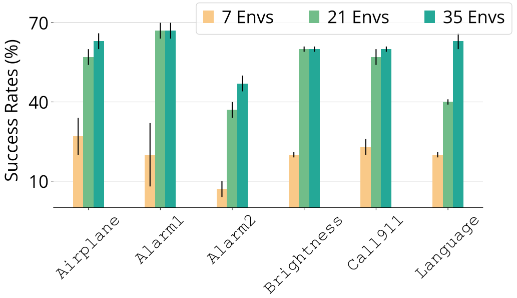


Figure 8: Success rates of VLUI agents with varying numbers of training environments. The success rates escalate with more environments.

4.3 Investigations on Design Choices for LLM Agents and MLLM Agents

The performance of LLM agents heavily relies on how the input prompts are tailored. Considering only the leaf UI elements of Android view hierarchy to describe the screen layout, similar to prior work (Li et al., 2020; Yang et al., 2023b) for example, might result in meaningless descriptions in certain applications (e.g., the setting application on `Airplane` and `Language`). In this work, we have leveraged the text attributes of all the available nodes to avoid such collapse, while we believe there can be more simple yet expressive representation methods.

In addition, we observe that few-shot examples can significantly improve the performance of LLM agents with GPT-4 compared to zero-shot cases. As shown in Table 2, equipping prompt with few-shot examples improves the performance from 42% to 67% on `Alarm1` and from 27% to 43% on `Language`. However, employing few-shot examples does not always help agents, as shown on `Alarm2` or `Brightness`. We note that naive exploitation of expert demonstrations might lead to excessive increases in computational cost and highlight the necessity of efficient few-shot prompting.

Moreover, we investigate the effect of common visual prompting methods for MLLM agents with GPT-4V. To enhance the visual grounding ability of MLLMs, prior studies (Yan et al., 2023; Yang et al., 2023b) have actively adopted SoM prompting, where each UI element in the input image is marked with numeric tags. However, we find that SoM prompting can often significantly degrade the performance of MLLM agents on `Alarm2` and `Call 911` as shown in Table 2. We hypothesize that the numeric tags may cause confusion when overlaid on UI elements with numbers, such as dial buttons or clock interfaces. For examples of the inputs used in SoM prompting, see Appendix D.3.

4.4 Effects of Pre-trained Encoders and Data Diversity for VLUI Agents

The main challenge of VLUI agents is the lack of generalization ability as mentioned in Section 4.2. Hence, we examine the different algorithmic designs for the representation model of VLUI agents and the effects of training diversity on performance robustness. We also include an additional experiment with varying model sizes of visual encoders in Appendix E.4.

First, we compare VLUI agents in two different designs: visual encoders with parameters randomly initialized and visual encoders pre-trained with ImageNet (Krizhevsky et al., 2017). As shown in Figure 7, we observe significant improvements in success rates with pre-training, e.g., from 37% to 63% on `Language`. These results demonstrate the benefit of employing pre-trained representation models, and we expect further improvements can be induced by leveraging more Android-specific images for pre-training (Sun et al., 2022; Rawles et al., 2023).

Furthermore, we train VLUI agents by progressively increasing the number of training environments (see Appendix D.1 for more details of the experiment setting). As shown in Figure 8, as the number of training environments increases, the performance of VLUI agents escalates. Specifically, the agents exhibit success rates of 20%, 40%, and 63% on `Language` with the number of training environments 7, 21, and 35, respectively. We believe this verifies the efficacy of the environment randomization feature incorporated in our benchmark toward more practical agents.

5 Related Work

Foundation models for decision-making system Inspired by the strong emergent properties of foundation models (Brown et al., 2020; Wei et al., 2022), many researches have adopted LLMs to develop decision-making system (Yao et al., 2023; Shinn et al., 2023). In robot learning, for example, LLMs have been widely equipped for reasoning, planning, manipulation, and navigation (Driess et al., 2023; Liang et al., 2023; Huang et al., 2023). Furthermore, agents with LLMs have shown capabilities of performing interesting tasks in numerous simulated worlds, including game environments (Wang et al., 2023; Tan et al., 2024) and virtual reality (Qian et al., 2023; Yang et al., 2024). In recent days, focusing on practicalness, solving computer tasks with foundation models has also been actively explored (Nakano et al., 2021; Furuta et al., 2023). We further study the abilities of foundation models to control mobile devices toward assistive agents in real life.

Developing assistive agent for device control For agents that effectively understand and manipulate the UI elements, a large body of work has leveraged the structural information, such as document object model in HTML or Android view hierarchy (Branavan et al., 2010; Gur et al., 2019). In addition, methods for equipping agents with the ability to understand information-rich screen images have been widely investigated, mainly with vision-based reinforcement learning (Liu et al., 2018; Humphreys et al., 2022; Shaw et al., 2023). Recently, diverse strategies to build device control agents with foundation models are introduced, including prompting methods (Wen et al., 2023; Kim et al., 2023), instruction-tuning (Furuta et al., 2023), fine-tuning with images (Zhan & Zhang, 2023; Hong et al., 2023), and visual prompting (Yan et al., 2023; Yang et al., 2023b). Here, we present an elaborate analysis of the main methods for building mobile device control agents.

Benchmark for decision-making agents There have been continuous efforts to build reliable benchmarks for sequential decision-making in video games (Bellemare et al., 2013), locomotion (Brockman et al., 2016), and robotic manipulation (James et al., 2020). Lately, researchers have proposed benchmarks for solving device control tasks, viewing it as another decision-making problem. For example, Yao et al. (2022) and Zhou et al. (2024) have presented benchmark simulating web platforms, while Toyama et al. (2021), Shvo et al. (2021), and Zhang et al. (2023) have suggested RL environments adopting Android emulators. In this work, inspired by special-purpose benchmarks quantifying the robustness of the agents (Cobbe et al., 2020; Stone et al., 2021), we newly propose a benchmark with the randomization feature.

6 Discussion & Conclusion

We present B-MoCA, a new benchmark designed for evaluating mobile device control agents. Our benchmark provides diverse tasks applicable to everyday routines and environments that simulate numerous device configurations. We conduct extensive experiments and demonstrate that B-MoCA can serve as a standardized platform for developing different types of agents in a unified setting. Finally, we mention several limitations and promising future directions of this work:

- *Tasks with text typing* While we define the action spaces with dual-gesture actions, text typing by touching the soft keyboard demands excessively long interactions. In the future, we plan to include tasks requiring text typing, such as web search or e-mail sending, with advanced interfaces.
- *Open-ended tasks and reward modeling* Since the ADB-based success detector does not capture the semantics of agent behaviors, tasks with ambiguous success criteria are hard to evaluate. Alternatively, we believe employing the reward model learned from demonstrations (Fan et al., 2022) can be used for integrating open-ended tasks.
- *More on LLM agents* Foundation models can be employed in different ways, such as using them as a high-level planner to operate a set of pre-defined APIs (Chen & Li, 2024) or neural network policies (Ahn et al., 2022) as low-level actors. Also, as training VLUI agents with demonstrations results in high performances, fine-tuning LLMs is highly promising.

Toward practical mobile device control agents, we hope that B-MoCA stands as a valuable platform with helpful resources for future innovations.

IMPACT STATEMENT

This study proposes a benchmark designed to assess interactive mobile device management agents, with social opportunities to enhance user accessibility and aid those facing disabilities. We caution users about privacy concerns while we try to eliminate such potentials during task designs. Noting the importance of research for preventing malicious usages of device control agents, we emphasize B-MoCA as a useful test bed.

ACKNOWLEDGMENTS

We thank Dongjun Lee, Kyuyoung Kim, and Ahjeong Seo for providing sincere suggestions for improving our work. This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00075 Artificial Intelligence Graduate School Program (KAIST)).

REFERENCES

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

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *The Conference on Robot Learning*, 2022.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

SRK Branavan, Luke Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics*, 2010.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Conference on Neural Information Processing Systems*, 2020.

Wei Chen and Zhiyuan Li. Octopus v2: On-device language model for super agent. *arXiv preprint arXiv:2404.01744*, 2024.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, 2020.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *International Conference on Machine Learning*, 2023.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Conference on Neural Information Processing Systems*, 2022.

Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. Instruction-finetuned foundation models for multimodal web navigation. In *International Conference on Learning Representations 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023.

Team Gemini, 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.

Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to navigate the web. In *International Conference on Learning Representations*, 2019.

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. *arXiv preprint arXiv:2312.08914*, 2023.

Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. In *International Conference on Robotics and Automation*, 2023.

Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*, 2022.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Conference on Neural Information Processing Systems*, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2017.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning: State-of-the-art*, pp. 45–73. Springer, 2012.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Wei Li, Fu-Lin Hsu, Will Bishop, Folawiyo Campbell-Ajala, Oriana Riva, and Max Lin. Uinav: A maker of ui automation agents. *arXiv preprint arXiv:2312.10170*, 2023.

Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. Widget captioning: Generating natural language description for mobile user interface elements. In *Conference on Empirical Methods in Natural Language Processing*, 2020.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *International Conference on Robotics and Automation*, 2023.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Conference on Neural Information Processing Systems*, 2023.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *International Conference on Machine learning*, 2007.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Conference on Neural Information Processing Systems*, 1988.

Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P Lillicrap. Androidinthewild: A large-scale dataset for android device control. In *Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Stefan Schaal. Learning from demonstration. *Conference on Neural Information Processing Systems*, 1996.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. In *Conference on Neural Information Processing Systems*, 2023.

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

Maayan Shvo, Zhiming Hu, Rodrigo Toro Icarte, Iqbal Mohomed, Allan D. Jepson, and Sheila A. McIlraith. Appbuddy: Learning to accomplish tasks in mobile apps via reinforcement learning. In *Canadian Conference on Artificial Intelligence*, 2021.

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite–a challenging benchmark for reinforcement learning from pixels. *arXiv preprint arXiv:2101.02722*, 2021.

Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. Meta-gui: Towards multi-modal conversational agents on mobile gui. *Conference on Empirical Methods in Natural Language Processing*, 2022.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 2019.

Weihao Tan, Ziluo Ding, Wentao Zhang, Boyu Li, Bohan Zhou, Junpeng Yue, Haochong Xia, Jiechuan Jiang, Longtao Zheng, Xinrun Xu, et al. Towards general computer control: A multimodal agent for red dead redemption ii as a case study. *arXiv preprint arXiv:2403.03186*, 2024.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems*, 2017.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Conference on Neural Information Processing Systems*, 2017.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. In *Conference on Neural Information Processing Systems*, 2023.

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. In *Conference on Neural Information Processing Systems*, 2022.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Empowering llm to use smartphone for intelligent task automation. *arXiv preprint arXiv:2308.15272*, 2023.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.

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*, 2023a.

Jihan Yang, Runyu Ding, Ellis Brown, Xiaojuan Qi, and Saining Xie. V-irl: Grounding virtual intelligence in real life. *arXiv preprint arXiv:2402.03310*, 2024.

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*, 2023b.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Conference on Neural Information Processing Systems*, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.

Zhuosheng Zhan and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*, 2023.

Danyang Zhang, Lu Chen, and Kai Yu. Mobile-env: A universal platform for training and evaluation of mobile interaction. *arXiv preprint arXiv:2305.08144*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

Appendix:

Benchmarking Mobile Device Control Agent across Diverse Configurations

A Environment Details

A.1 Environment Implementation and interface

Environment B-MoCA is based on Android OS for real-system interactive evaluation. The environment is simulated with Android virtual devices, containing the device hardware profile, system image, storage area, and other relevant properties. The dynamics of the environments, such as the transition rules, are governed by Android OS and applications.

Each environment is represented as an Android device, running on top of the Android emulator. To be more specific, we define each environment as a snapshot, a stored image of the Android virtual device. Each snapshot is built by saving an image of the target device after the configurations. These configurations include randomizing the features of environments by placing icons in random locations, setting dots per inch (DPI), modifying wallpapers, and changing the language. In addition, our configuration process includes adjusting several device settings for accurate evaluation, such as changing the database of applications.

To facilitate interactions between the environment and agents, we develop a set of interfaces. These interfaces encompass various functionalities: to provide the task descriptions in text to the agents, to capture screenshots of the virtual device, to provide the Android view hierarchy in XML format and parse the text description of the screen, to extract dual-gesture actions from text-based actions, and to deliver the dual-gesture action to the Android emulator.

Interaction Frequency The Android emulators run asynchronously independent of the agent that is interacting with the environments. However, this asynchronicity between the agent and the environment may cause several issues such as incomplete transition of the environments or delayed success signals. To alleviate the issue, we adjust the interaction frequency between agents and environments. Specifically, this adjustment is operated by forcing the agent to wait a pre-defined time before fetching the screen information from the environment. In our experiments, we fix the interaction frequency during evaluation to be $1/3$Hz across all types of agents.

Observation space The observation space is comprised of either a screen image, a text description of the screen in XML formats based on the Android view hierarchy, or both.

The screen images are used for multi-modal large language model (MLLM) agents and vision-language-UI (VLUI) agents. Each image is resized into a resolution of 1024×2048 for MLLM agents and 128×256 for VLUI agents.

The text descriptions are used for agents with LLMs and MLLMs. To build the text description, the Android debug bridge (ADB) UI Automator is employed for acquiring the Android view hierarchy in XML format. A pre-defined parser, then, converts the list of UI elements with attributes in an XML file into a set of text descriptions of UI elements. The description includes the numeric tag of UI elements, a short description of the elements including class name or content descriptions (e.g., "com.google.android.apps.nexuslauncher.id_title_weather_text_Sunny,1°C" for a view in the home screen describing the weather). Also, we optionally provide the bounding box x-y coordinates specifying the location of the elements, such as the slider interface. Moreover, we define the parser to capture the descriptions of all the nodes in the Android view hierarchy. This is because we observe that many UI elements are omitted if we only parse leaf nodes, resulting in meaningless descriptions as discussed in Section 4.3.

Action space The action space of the agents is defined as a set of dual-gesture actions $\{a | \ a = (y_{\text{touch}}, x_{\text{touch}}, y_{\text{lift}}, x_{\text{lift}}) \in \mathbb{R}^4\}$, similar to Rawles et al. (2023). Each value of dual-gesture action a is normalized to be in between $[-1, 1]$ with respect to the screen resolutions. The former two values specify the location of the screen to touch, while the latter two values determine the location of the screen to lift. This definition enables interpreting useful actions in digital device control, i.e., tapping or swiping the screens, in a precise and compressive manner. Also, our interface allows pressing the navigator buttons available by touching the screen to support the essential actions for manipulating Android devices.

Specifically, we implement an interface that determines whether the action is a tap, swipe, or pressing of navigation buttons i.e., back, home, and overview. The action parsing interface converts the action into taps, swipes, or pressing buttons following the rule as follows:

- The action is `tapping`, if $d((x_{\text{touch}}, y_{\text{touch}}), (x_{\text{lift}}, y_{\text{lift}})) <$ threshold
  - The `tapping` is to press BACK button, if $(x_{\text{touch}}, y_{\text{touch}}) = (0.95, 0.22)$
  - The `tapping` is to press HOME button, if $(x_{\text{touch}}, y_{\text{touch}}) = (0.95, 0.50)$
  - The `tapping` is to press OVERVIEW button, if $(x_{\text{touch}}, y_{\text{touch}}) = (0.95, 0.78)$
- The action is `swiping`, if $d((x_{\text{touch}}, y_{\text{touch}}), (x_{\text{lift}}, y_{\text{lift}})) \geq$ threshold,

where the threshold value is defined as 0.14. This value is adjustable by users, while we find that the value of 0.14 ensures proper interactions over UI elements, e.g., `tapping` the target application icon, in all of our experiments. These specific values are tested to be consistent across different device types, ensuring that the positions correspond to the correct buttons in all B-MoCA environments.

For LLM agents and MLLM agents, we further define action options. Following the action space definition, the action options are designed to be compatible with a dual-gesture action. We prompt the LLM agents to output actions among six possible options: raw dual-gesture action, tap, swipe, press("HOME"), press("BACK"), and press("OVERVIEW"). These action options are converted into a corresponding dual-gesture action by an additional action extractor we define as below:

- For the dual-gesture action, we convert the text action into the four floating points by rounding each value into the second decimal point.
- For tap actions, the LLM agent outputs an integer value specifying the numeric tag assigned to the UI element. Given the tapping action with a numeric tag, the parser converts the action into a tapping dual-gesture action with the bounding box information of the chosen UI element.
- For swipe actions, a direction 'up', 'down', 'left' and 'right' is converted into $(0.8, 0.5, 0.2, 0.5)$, $(0.2, 0.5, 0.8, 0.5)$, $(0.5, 0.2, 0.5, 0.8)$, and $(0.5, 0.8, 0.5, 0.2)$, respectively.
- For the action press("HOME"), press("BACK"), and press("OVERVIEW"), we convert the outputs to dual-gesture actions in the same way as VLUI agents.

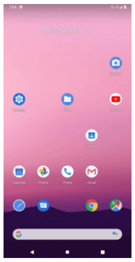
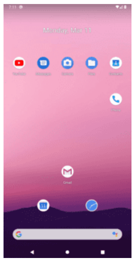
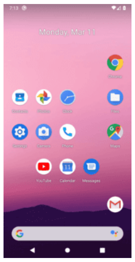
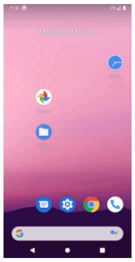
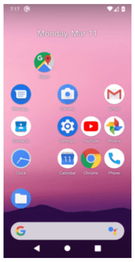
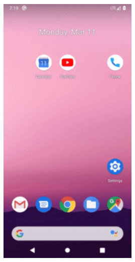
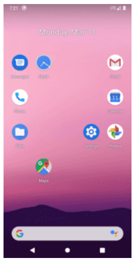
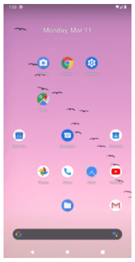
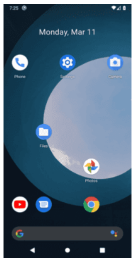
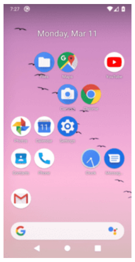
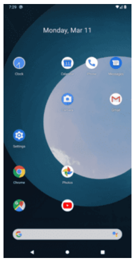
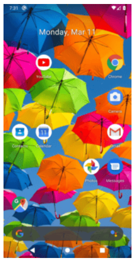
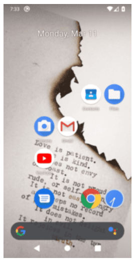
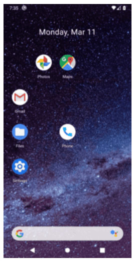
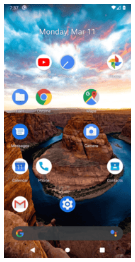
During the evaluation, we ignore the action in the wrong format by skipping the transition of the environments but penalizing the agents by incrementing the steps taken, yet we observe both `Gemini` and `GPT` models (as well as vision version of them) rarely make mistakes on the format.

A.2 Training and Test Environments Configurations

We construct 45 unique environments in B-MoCA, where 35 environments are for training and 10 environments are for testing. Each environment is provided with a unique identification (ID) number, to distinguish the environments easily. Table 3 shows the list of the device configurations and the home screen images of exemplary environments.

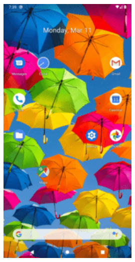
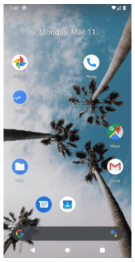
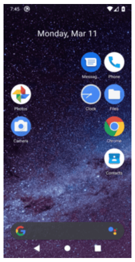
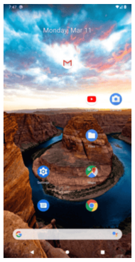
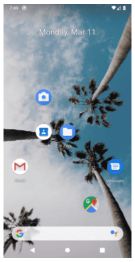
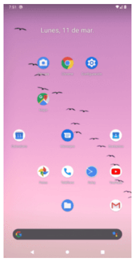
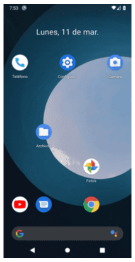
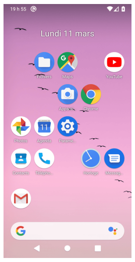
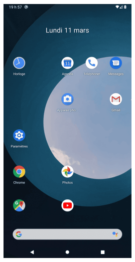
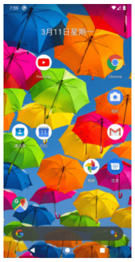
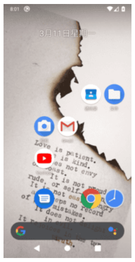
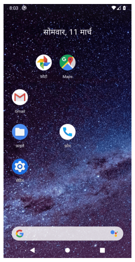
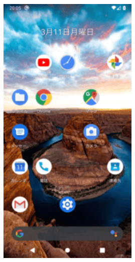
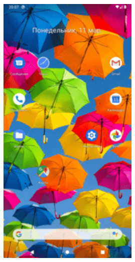
To construct environments, we use popular device types: Pixel 3, Pixel 4, Pixel 4 XL, Pixel 6, and WGXA Tablet. For training environments, only Pixel 3 is employed. For evaluation environments, we use all device types Pixel 3, Pixel 4, Pixel 4 XL, Pixel 6, and WGXA Tablet. In these models, we alter the icon and font sizes by changing the dots per inch (DPI) values of the devices. For each device type, we prepare three different sizes that users can select. We, then, change the wallpaper with 13 images collected from a free license image website. These wallpaper image files are shared in the open-source repository. We also customize the background images with the dark theme mode. If the dark theme mode is activated, the device provides screen images with light-dark color reversed. For instance, the wallpaper of the application list page is white in the default setting, while it becomes black with dark theme mode activated. Furthermore, we incorporate changes in locale, specifying the language and location of the devices. 12 different locales are used for 35 training environments, while we include three more locales for the test environments.

Table 3: The device configuration of each environment with the home screen image

					
ID	000	001	002	003	004
Device type	Pixel 3	Pixel 3	Pixel 3	Pixel 3	Pixel 3
DPI	330	330	440	440	550
Locale	en-US	en-US	en-US	en-US	en-US
Wallpaper	000.jpg	000.jpg	000.jpg	000.jpg	000.jpg
Dark theme	-	-	-	-	-
					
ID	005	006	007	008	009
Device type	Pixel 3	Pixel 3	Pixel 3	Pixel 3	Pixel 3
DPI	440	440	330	440	550
Locale	en-US	en-US	en-US	en-US	en-US
Wallpaper	000.jpg	000.jpg	001.jpg	002.jpg	001.jpg
Dark theme	-	-	✓	✓	-
					
ID	010	011	012	013	014
Device type	Pixel 3	Pixel 3	Pixel 3	Pixel 3	Pixel 3
DPI	330	440	550	440	440
Locale	en-US	en-US	en-US	en-US	en-US
Wallpaper	002.jpg	008.jpg	003.jpg	010.jpg	013.jpg
Dark theme	-	✓	✓	-	✓

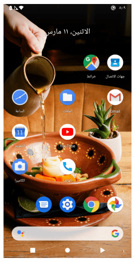
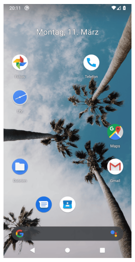
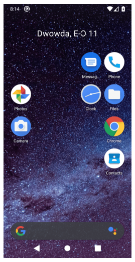
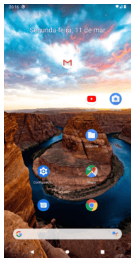
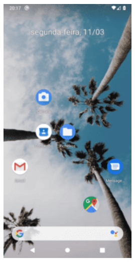
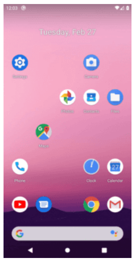
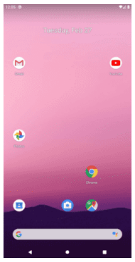
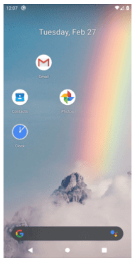
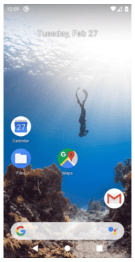
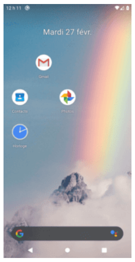
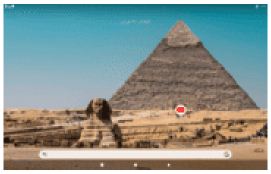
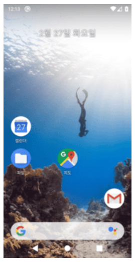
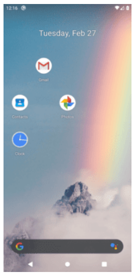
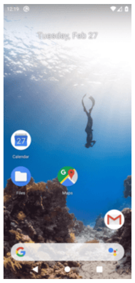
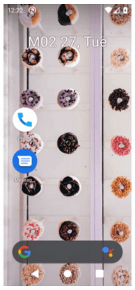
Continued on next page

Table 3: The device configuration of each environment with the home screen image (Continued)

					
ID	015	016	017	018	019
Device type	Pixel 3	Pixel 3	Pixel 3	Pixel 3	Pixel 3
DPI	330	440	440	550	330
Locale	en-US	en-US	en-US	en-US	en-US
Wallpaper	008.jpg	007.jpg	004.jpg	010.jpg	013.jpg
Dark theme	-	-	✓	✓	-
					
ID	020	021	022	023	024
Device type	Pixel 3	Pixel 3	Pixel 3	Pixel 3	Pixel 3
DPI	440	330	440	550	330
Locale	en-US	es-US	es-US	fr-CA	fr-CA
Wallpaper	004.jpg	001.jpg	002.jpg	001.jpg	002.jpg
Dark theme	-	✓	✓	-	-
					
ID	025	026	027	028	029
Device type	Pixel 3	Pixel 4	Pixel 4	Pixel 4	Pixel 5
DPI	440	550	440	440	330
Locale	zh-hans-CN	zh-hans-CN	hi-IN	ja-JP	ru-MD
Wallpaper	008.jpg	003.jpg	010.jpg	013.jpg	008.jpg
Dark theme	✓	✓	-	✓	-

Continued on next page

Table 3: The device configuration of each environment with the home screen image (Continued)

					
ID	030	031	032	033	034
Device type	Pixel 3	Pixel 3	Pixel 3	Pixel 3	Pixel 3
DPI	440	440	550	330	440
Locale	ar-AE	de-DE	ak-GH	pt-BR	pt-PT
Wallpaper	007.jpg	004.jpg	010.jpg	013.jpg	004.jpg
Dark theme	-	✓	✓	-	-
					
ID	100	101	102	103	104
Device type	Pixel 3	Pixel 3	Pixel 3	Pixel 3	Pixel 3
DPI	440	330	440	550	440
Locale	en-US	en-US	en-US	en-US	fr-CA
Wallpaper	000.jpg	000.jpg	009.jpg	012.jpg	009.jpg
Dark theme	-	-	✓	-	✓
					
ID	109	105	106	107	108
Device type	WXGA Tablet	Pixel 3	Pixel 4	Pixel 5	Pixel 6
DPI	160	550	440	550	700
Locale	ar-EG	ko-KR	en-US	en-US	ur-PK
Wallpaper	012.jpg	009.jpg	012.jpg	005.jpg	011.jpg
Dark theme	-	✓	-	✓	-

B Task Details

B.1 List of daily tasks

B-MoCA presents 60 daily tasks that are common in everyday life. The tasks are designed to operate in diverse environments seamlessly and cover commonly used applications. Daily tasks effectively simulate a wide range of essential skills for mobile device control problems, such as manipulating UI elements (including application icons, checkboxes, and sliders), and can be employed for evaluating mobile device control agents' capabilities in performing tasks that mirror our daily activities.

In Table 4, we include the detailed list of tasks with the maximum step limit and the success criteria. The success criteria are defined in the form of regular expression and are employed by the rule-based success detector. The success criteria (filter) specifies the target application or activity, and the success criteria (regex) refers to the regular expression we use. We also define the maximum step limits, which are set for the rigorous evaluation of the agents' proficiency on each task.

B.2 Example of Demonstration on Representative Tasks

In our experiments, we select six representative tasks. The tasks are selected to cover a wide range of functionalities, such as navigating pages (e.g., tab in the clock application or different setting pages in the setting application) and manipulating various UI elements (e.g., checkbox, slider, time pickers, etc.). On each task, we display the successful demonstration in Figure 9.

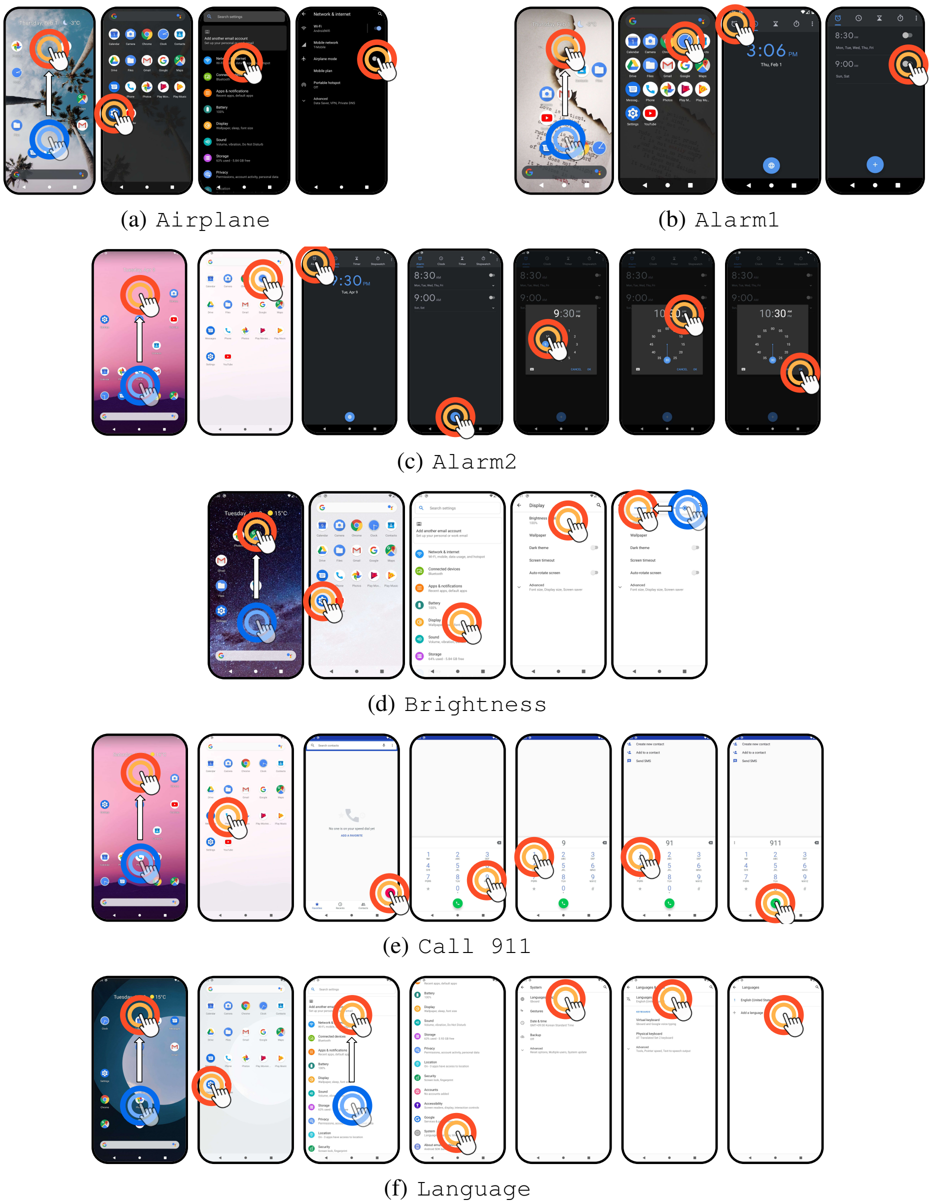
Table 4: Comprehensive list of tasks.

Step limit	Task instruction	Success criteria (filter)	Success criteria (regex)
4	"open the calendar app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.android.calendar
4	"open the camera app"	ActivityTaskManager	ˆ(.*?)Start proc(.*?)com.android.camera
4	"open the chrome app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.google.android.apps.chrome
4	"open the clock app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.android.deskclock
4	"open the contact app"	ActivityTaskManager	ˆ(.*?)Start proc(.*?)com.android.contacts
4	"open the file manager app"	ActivityTaskManager	ˆ(.*?)START(.*?)files.FilesActivity
4	"open the gmail app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.google.android.gm
4	"open the map app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.google.android.maps.MapsActivity
4	"open the message app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.google.android.apps.messaging
4	"open the phone app"	Dialer	ˆ(.*?)MainActivity.onCreate
4	"open the photos app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.google.android.apps.photos
4	"open the play music app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.android.music
4	"open the setting app"	ActivityManager	ˆ(*.?)Start proc(.*?)com.android.settings.Settings
4	"open the youtube app"	ActivityTaskManager	ˆ(.*?)START(.*?)com.google.android.youtube
4	"turn on alarm at 9 am"	AlarmClock	ˆ(.*?)Created new alarm instance
5	"delete alarm at 9 am"	AlarmClock	ˆ(.*?)Removed alarm
5	"go to the alarm page in clock"	AlarmClock	ˆ(.*?)Events: [Alarm] [Show Tab] [Tap]
5	"go to the stopwatch page in clock"	AlarmClock	ˆ(.*?)Events: [Stopwatch] [Show Tab] [Tap]
5	"go to the timer page in clock"	AlarmClock	ˆ(.*?)Events: [Timer] [Show Tab] [Tap]
5	"list audio files in file manager"	DirectoryFragment	ˆ(.*?)Showing directory(.*?)audio(.*?)root
5	"list image files in file manager"	DirectoryFragment	ˆ(.*?)Showing directory(.*?)images
5	"list video files in file manager"	DirectoryFragment	ˆ(.*?)Showing directory(.*?)videos

Continued on next page

Table 4: Comprehensive list of tasks. (Continued)

5	“list download files in file manager”	DirectoryFragment	ˆ(.*?)Showing directory(.*?)download
5	“activate the insert page in contact”	ActivityTaskManager	ˆ(.*?)START(.*?)INSERT(.*?)ContactEditorActivity
5	“activate the edit page in contact”	ActivityTaskManager	ˆ(.*?)START(.*?)EDIT(.*?)ContactEditorActivity
5	“activate the search bar in chrome”	AndroidIME	ˆ(.*?)LatinIme.onActivate(.*?)android.chrome
5	“activate the search bar in map”	AndroidIME	ˆ(.*?)LatinIme.onActivate(.*?)apps.map
5	“activate the search bar in youtube”	AndroidIME	ˆ(.*?)LatinIme.onActivate(.*?)android.youtube
5	“activate the search bar in google”	AndroidIME	ˆ(.*?)LatinIme.onActivate(.*?)android.googlequicksearchbox
5	“activate the search bar in message”	AndroidIME	ˆ(.*?)LatinIme.onActivate(.*?)apps.messaging
5	“start chatting in message”	BugleUsageStatistics	ˆ(.*?)BUGLE CREATE(.*?)DEFAULT
5	“press the call button in dial”	Telecom	ˆ(.*?)LogUtils(.*?)EventRecord added as Call
5	“turn on airplane mode”	PhoneGlobals	ˆ(.*?)Turning radio off(.*?)airplane
5	“turn off airplane mode”	PhoneGlobals	ˆ(.*?)Turning radio on(.*?)airplane
5	“turn on wifi”	WifiService	ˆ(.*?)setWifiEnabled(.*?)com.android.settings(.*?)enable=true
5	“turn off wifi”	WifiService	ˆ(.*?)setWifiEnabled(.*?)com.android.settings(.*?)enable=false
5	“start the stopwatch in clock”	AlarmClock	ˆ(.*?)Start
5	“pause the stopwatch in clock”	AlarmClock	ˆ(.*?)Pause
5	“reset the stopwatch in clock”	AlarmClock	ˆ(.*?)Reset
5	“go to search history in chrome”	ActivityTaskManager	ˆ(.*?)START(.*?)chrome.browser.history.HistoryActivity
5	“go to trash page in photo”	ActivityTaskManager	ˆ(.*?)START(.*?)apps.photos(.*?)TrashPhotosActivity
5	“go to smart pairing page in youtube”	ActivityTaskManager	ˆ(.*?)START(.*?)youtube.mdx.smartpairing.PairWithTvActivity
6	“increase media volume in setting”	vol.Events	ˆ(.*?)MEDIA
6	“increase call volume in setting”	vol.Events	ˆ(.*?)CALL
6	“increase ring volume in setting”	vol.Events	ˆ(.*?)MUSIC
6	“increase alarm volume in setting”	vol.Events	ˆ(.*?)ALARM
6	“decrease screen brightness in setting”	DisplayPowerController	ˆ(.*?)Brightness(.*?)changing(.*?)manual
6	“toggle dark theme in setting”	SettingsProvider	ˆ(.*?)content(.*?)settings(.*?)dark(.*?)mode
6	“toggle vibrate for calls in setting”	SettingsProvider	ˆ(.*?)vibrate(.*?)when(.*?)ringing
6	“go to app info list in setting”	SettingsActivity	ˆ(.*?)Switching(.*?)android.settings(.*?)ManageApplications
6	“go to bluetooth setting”	PrefCtrlListHelper	ˆ(.*?)android.settings.bluetooth.BluetoothDevice
7	“go to ‘add a language’ page in setting”	ActivityTaskManager	ˆ(.*?)LocalePicker
9	“call 911”	Telecom	ˆ(.*?)Emergency number detected
10	“turn off the call in process”	Telecom	ˆ(.*?)InCallController(.*?)onCallRemoved
11	“create alarm at 06:30 am”	ConditionProviders.SCP	ˆ(.*?)nextUserAlarmTime(.*?)06:30:00
11	“create alarm at 10:30 am”	ConditionProviders.SCP	ˆ(.*?)nextUserAlarmTime(.*?)10:30:00
11	“create alarm at 13:30 pm”	ConditionProviders.SCP	ˆ(.*?)nextUserAlarmTime(.*?)13:30:00
11	“create alarm at 17:30 pm”	ConditionProviders.SCP	ˆ(.*?)nextUserAlarmTime(.*?)17:30:00
11	“create alarm at 20:30 pm”	ConditionProviders.SCP	ˆ(.*?)nextUserAlarmTime(.*?)20:30:00
11	“create alarm at 23:30 pm”	ConditionProviders.SCP	ˆ(.*?)nextUserAlarmTime(.*?)23:30:00



(a) Airplane

(b) Alarm1

(c) Alarm2

(d) Brightness

(e) Call 911

(f) Language

Figure 9: Examples of human expert demonstrations of six representative tasks. The blue and red cursors linked with a white arrow identify the `swiping` action, while the red cursor alone identifies the `tapping` action.

C Agent Details

C.1 Prompt Details for LLM Agents and MLLM Agents

For the agents employing LLMs or MLLMs, we use a complete prompt format described in Table 5. The role description informs the agents with general instructions about the problem, i.e., device control problem. The possible actions are provided as callable functions, options of tapping an element in the list, swiping the screen, and pressing the three navigation buttons over the screen with action press("BACK"), action press("HOME"), and action press("OVERVIEW"). The output format is designed to integrate the Chain-of-Thought (CoT) technique (Wei et al., 2022).

You are an agent that is trained to perform daily tasks on digital devices, such as smartphones. You are given a goal of task instruction to accomplish and a description of screen from Android view hierarchy, which contains elements' numeric tag and description.

Based on the goal of task instruction and UI elements list, you need to select an action option by calling one of the following functions to control the digital device:

1. dual-gesture(touch y: float, touch x: float, lift y: float, lift x: float): This function is used to operate a dual-gesture action. A dual-gesture comprises of four floating point numeric values, in between 0 and 1 indicating a normalized location of the screen in each of x-y coordinates. A dual-gesture action is interpreted as touching the screen at the location of (touch y, touch x) and lifting at the location of (lift y, lift x). The dual-gesture action indicates a tapping action if the touch and lift locations are identical but a swiping action if they differ. A simple use case is dual-gesture(0.5, 0.5, 0.5, 0.5) to tap the center of the screen.

2. tap(numeric tag: int): This function is used to tap an UI element shown on the digital device screen. ”numeric tag” is a tag assigned to an UI element shown on the digital device screen. A simple use case can be tap(5), which taps the UI element labeled with the number 5.

3. swipe(direction: str): This function is used to swipe on the digital device screen, ”direction” is a string that represents one of the four directions: up, down, left, right. ”direction” must be wrapped with double quotation marks. A simple use case is swipe(”up”) which can be used to open the app list in the home screen.

4. press(”HOME”): to press home button.

5. press(”BACK”): to press back button.

6. press(”OVERVIEW”): to press overview button.

Goal: [task instruction].

[few shot prompt]

Now, given the parsed uiautomator xml, you need to think and call the function needed to proceed with the task.

Your output should include three parts in the given format:

- Description: <Describe what you observe in the input>
- 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. You cannot output anything else except a function call>

You can only take one action at a time, so please directly call the function.

Please never take action beside options provided.

Table 5: Prompts used for the LLM Agents and MLLM agents. Parts for [...] are filled in according to different experiments, while the few-shot examples are optional.

For few-shot learning of agents with foundation models, we include a pre-defined number of examples hinting correct actions to take. Specifically, to build a prompt with few-shot examples, the [few shot prompt] part in Table 5 is replaced with the text illustrating the human demonstration. Table 6 shows an exemplary few show prompts, with one transition of the human expert demonstration.

Below illustrates exemplary human demonstration(s), with format:

- Instruction: <The instruction of task>
- Observation: <An observation from environment>
- Action: <An action taken by the human expert>
- Next Observation: <The next observation from environment after the action is taken>
- Reward: <A reward after action is executed>.

- Instruction: turn on alarm at 9 am
- Observation: ['numeric_tag': 0, 'description': 'android.view.View_Appslist', [...] 'numeric_tag': 27, 'description': 'android.widget.FrameLayout']
- Action: swipe(”up”)
- Next observation: ['numeric_tag': 0, 'description': 'android.view.View_Appslist', [...] 'numeric_tag': 30, 'description': 'android.widget.FrameLayout']
- Reward: 0.0

Table 6: An exemplary few-shot prompt with one transition of human expert demonstration for text-based agents. The abbreviated [...] parts are filled with a list of descriptions for UI elements.

C.2 Architecture Design for VLUI Agents

The network architecture for VLUI agents is composed of three components: encoder, attention module, and action head. Given the task instruction c and the visual screen $o_t \in \mathbb{R}^{3\times256\times128}$ at each timestep t, VLUI agents generate action $a_t \in \mathbb{R}^4$ in the form of dual-gesture action.

VLUI agents use visual and text encoders to represent screen images o_t and task instruction c, respectively. The visual encoder embeds visual feature $e_{o_t} \in \mathbb{R}^d$ from the observation o_t, and the text encoder extracts features $e_c \in \mathbb{R}^d$ from the task instruction c. For the visual encoder, we use EfficientNet-b0 (Tan & Le, 2019) pre-trained with ImageNet followed by an adaption layer using a fully connected layer to adapt the output channel to hidden dimension d (Liu et al., 2023). For the text encoder, we use a pre-trained text encoder of Text-to-Text Transfer Transformer (Zhan & Zhang, 2023) which is trained with a dataset composed of demonstrations for solving Android device control problems (Rawles et al., 2023). The text encoder is kept frozen during the training process. The hidden dimension d is set to equal the value of 768 for both visual embedding and text embedding.

The attention module, then, fuses the visual feature e_{o_t} and text feature e_c into a single vision-language embedding $e_{\text{fused}} \in \mathbb{R}^d$. Especially, we use Multi-head attention layer (Vaswani et al., 2017) for cross-attention, with e_c given as query and e_{o_t} given as key and value. Given the fused feature e_{fused}, the action heads predict the action a_t. The action head consists of fully connected (FC) layers with the last layer having an output dimension of 4, accounting for the dimension of $a \in \mathbb{R}^4$. The sequence of three FC layers follows output dimensions of $(1024, 1024, 4)$. We apply the tanh layer to the predicted action, observing improved performances with normalization of the action values.

C.3 Configuration Details for LLM Agents and MLLM Agents

For the experiments in Section 4.2 and Section 4.3, we set the configurations for the foundation models. We use a temperature of 0.1, a top-p of 1, and a top-k of 1 for Gemini-Pro and Gemini-Pro-V. We set the temperature to be 0.0 and top-p with the default value of 1 (as altering only either temperature or top-p from the default setting is suggested) for GPT-4 and GPT-4V.

C.4 Training Details for VLUI Agents

For the experiments in Section 4.2 and Section 4.4, we train VLUI agents with behavior cloning (BC) over 4K steps with a batch size of 512, sampled from a collection of 210 human demonstrations. We use the Adam optimizer (Kingma & Ba, 2017) with a learning rate of 3e-4 and adopt a cosine annealing learning rate scheduler. Each training is conducted on a single NVIDIA RTX A6000 GPU and takes approximately one hour.

D Experiment Details

D.1 Dataset Collection

For the few-show learning of LLM and MLLM agents and training of VLUI agents, we collect human expert demonstrations. The collectors (graduate students) are instructed to complete the six representative tasks in each *training* environment. The definitions of action space for the collected demonstration are in two modes: the action space defined with action options and the action space as a set of dual-gesture actions. The end of each episode is determined by the ADB-based success detector, and we exclude the demonstrations with failures.

For the experiments in Section 4.2, we exploit training environments with identifying (ID) numbers from 000 to 034. Hence, a total number of 210 trajectories of demonstrations are prepared. For agents using foundation models, each transition (task instruction, observation, action, next observation, reward) in the trajectories is sampled as a few-shot example, similar to prior methods (Zhang et al., 2023; Rawles et al., 2023). For VLUI agents, each triplet (task instruction, observation, action) in the trajectories is used as a data point for composing the training batch. For the experiments in Section 4.4, we leverage varying numbers of training environments 7, 21, and 35 where the corresponding identifying (ID) numbers of the environments are from 000 to 006, from 000 to 021, and from 000 to 034, respectively. The total number of demonstrations for each setting is 42, 126, and 210, respectively.

D.2 Performances of VLUI Agents in Training Environments

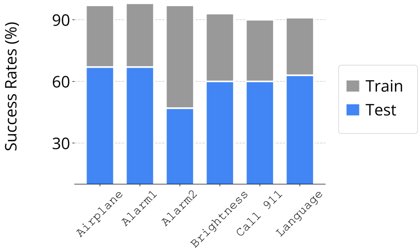
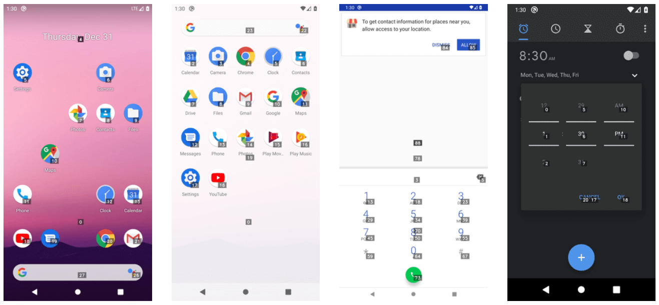


Figure 10: Success rates of VLUI agents trained with BC on training and test environments. The differences between the success rates demonstrate the headroom for the generalization ability.

Figure 10 displays the differences in the success rates of VLUI agents in training and test environments. The challenges with diverse device configurations degenerate the performances of the VLUI agents, from higher than 90% in the training environments to less than 70% in the test environments.

D.3 Examples of Visual Inputs with SoM Prompting for MLLM Agents

In Section 4.3, we investigate the effects of SoM prompting that several prior works (Yan et al., 2023; Yang et al., 2023b) adopted. Figure 11 presents several examples of visual inputs used for analysis.



(a) (b) (c) (d)

Figure 11: Examples of visual inputs for MLLM agents with SoM prompting. The overlaid numeric tags for selecting icons at (a) the home screen or (b) the menu screen of applications list can be beneficial for MLLM agents, while the tags on (c) dial buttons or (d) clock UI may confuse them.

E Additional Experiments

E.1 MLLM Agents with Dual-Gesture Actions

We have implemented additional interfaces for agents employing foundation models and action options, as introduced in Section 3.1. However, it is still questionable whether defining action options is truly desirable for these agents. To answer this, we conduct a comparison between agents only generating actions in the dual-gesture action format and the agents using additional action options. In this experiment, we examine MLLM agents employing GPT-4V in a zero-shot manner on two selected tasks (`Airplane` and `Alarm 1`) with only one run for simplicity.

As shown in Table 7, we observe that the agents benefit from employing additional action options. In the experiments, the agents without additional options exhibit several successful trials on `Alarm 1`, by including the bounding box location of all UI elements in the observation prompt for these agents. However, we observe that the agents lack generating diverse dual-gesture actions but only perform tapping actions. With these results, we examine the proficiency of LLM agents and MLLM agents with the action options in Section 4.2.

	MLLM agents (dual-gesture actions)	MLLM agents (action options)
`Airplane`	00	30
`Alarm 1`	30	50

Table 7: Success rates of MLLM agents with different action spaces. MLLM agents (dual-gesture actions) generate the actions in only dual-gesture action format, and MLLM agents (action options) leverage the additional action options we define.

E.2 LLM Agents with Open-Source Models

While employment of foundation models for mobile device control agents is gaining interests (Wen et al., 2023; Yang et al., 2023b), many approaches still rely on closed-source models. However, leveraging closed-source foundation models lies with severe limitations, such as difficulties in fine-tuning. Instead, one can employ open-source LLMs which can benefit from high flexibility in usage. In this experiment, we examine the proficiency of LLM agents with open-source models.

We study open-source models: Llama2-chat (abbreviated as Llama2) (Touvron et al., 2023), Llama3, and AgentLM (Zeng et al., 2023). Llama2 and Llama3 are open-source models that have shown compatible performances with several closed-sourced models, and AgentLM is an instruction-tuned version of Llama2 in a collection of numerous agent tasks (including web tasks). For Llama2 and AgentLM, we use 7b and 13b size models. For Llama3, we use an 8b size model. We set the temperature value to be 0.1 and report across three different runs.

Table 8 show the success rates of LLM with open-source models. We observe that these agents severely lack the proficiency in performing tasks that we select. While the agents can perform sub-tasks of opening the target application on the home screen or entering the menu screen in some trials, as observed in the rollouts, they fail to complete the instructed tasks in limited allowed steps. The open-source models also struggle with generating actions in the format we instruct, while closed-source models rarely generate actions in the wrong format. With these pilot test results, we primarily focus on examining the efficacy of training agents from scratch with VLUI agents.

	LLM agents (Llama2-7b)	LLM agents (Llama2-13b)	LLM agents (AgentLM-7b)	LLM agents (AgentLM-13b)	LLM agents (Llama3-8b)	LLM agents (Gemini-Pro)	LLM agents (GPT-4)
`Airplane`	00 ± 00	00 ± 00	00 ± 00	00 ± 00	63 ± 00	87 ± 07	53 ± 03
`Alarm 1`	00 ± 00	00 ± 00	00 ± 00	00 ± 00	00 ± 00	27 ± 03	42 ± 13
`Brightness`	00 ± 00	00 ± 00	00 ± 00	00 ± 00	17 ± 03	05 ± 03	73 ± 12

Table 8: Success rates of LLM agents with open-source models Llama2 and AgentLM in zero-shot scenario. The agents do not complete any tasks that LLM agents with closed-source models of GPT-4 or Gemini (in zero-shot) have achieved.

E.3 VLUI Agents with Reinforcement Learning

We study VLUI agents trained using reinforcement learning (RL) algorithms, by using the success signal r_t as a sparse reward. Formally, the RL agent is trained to maximize the expected return, denoted as follows:

$$\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right],$$

where $\gamma \in [0, 1]$ is the discount factor and T is the length of episode. In this experiment, we focus on offline RL setting (Lange et al., 2012; Levine et al., 2020), where the agent learns from a pre-collected dataset generated by some behavior policy. Specifically, We utilize the implicit Q-learning (IQL; Kostrikov et al. 2022) algorithm.

IQL is one of the popular actor-critic algorithms. The actor network π parameterized with θ infers the action a_t at each time step t, given the observation o_t and task instruction c_t, The critic network Q parameterized with ϕ estimates the value of action predicted by the actor. IQL leverages expectile regression for robust value estimation and improves the policy using advantage-weighted regression (Peters & Schaal, 2007; Nair et al., 2020). In particular, IQL introduces a separate value network V parameterized with ψ for robust learning. The loss function for the critic in IQL is formulated as:

$$L_V(\psi) = \mathbb{E}_{(o_t,a_t^*)\sim\mathcal{D}}\Big[L_2^\tau\big(Q_{\hat{\phi}}(o_t,a_t) - V_\psi(o_t)\big)\Big],$$

$$L_Q(\phi) = \mathbb{E}_{(o_t,a^*,r_t,o_{t+1})\sim\mathcal{D}}\Big[\big(r_t + \gamma \cdot V_\psi(o_{t+1}) - Q_\phi(o_t,a_t^*)\big)^2\Big],$$

with $L_2^\tau(u)$ defined to be $|\tau - \mathbb{1}(u<0)| \cdot u^2$, a value function V parameterized with ψ, Q-function Q parameterized with ϕ, and the dataset of human demonstrations $\mathcal{D} = \{(o_t, a_t^*, r_t, o_{t+1})\}$. Then, the actor is trained with advantage-weighted behavioral cloning objective defined as:

$$L_\pi(\theta) = \mathbb{E}_{(o_t,a_t^*)\sim\mathcal{D}}\Big[\exp\big(\beta \cdot A(o_t,a_t)\big) \cdot \big(a_t - a_t^*\big)^2\Big],$$

with action prediction a_t predicted by the actor network π, the advantage $A(o_t,a_t) = Q_{\hat{\phi}}(o_t,a_t) - V_\psi(o_t)$, and an inverse temperature $\beta \in [0,\infty)$.

The policy architecture for VLUI agents follows the same architecture of VLUI agents trained with BC, described in Appendix C.2. Similarly, the hyperparameters for optimizers and other training details remain the same, except that we iterate the training over 20K steps to ensure the convergence of training. For the training, we employ 35 training environments, namely 210 successful human expert demonstrations.

Figure 12 shows the success rates of VLUI agents trained with IQL, compared with VLUI agents trained with BC. We observe that the agents trained with IQL do not exhibit compatible performances with the agents trained with BC across all tasks. We assume these results originated from training instability due to sparse rewards, as more training steps are required for the convergence of IQL training. However, as observed in Kostrikov et al. (2022), we expect that offline RL can provide potential benefits over vanilla BC training, such as utilizing failure demonstrations. We leave training VLUI agents with IQL more efficiently and with higher proficiencies as future work.

E.4 VLUI Agent with Representation Model of Varying Capacity

We conduct the effect of representation models with varying capacities on the robustness of VLUI agents. Specifically, we compare the VLUI agents equipped with visual encoders using EfficientNet-b0, EfficientNet-b3, and EfficientNet-b7 (with increasing numbers of parameters with values of 5.3M, 12M, and 66M, respectively), which are pre-trained with ImageNet.

Figure 13 demonstrates the experimental results. To illustrate, our experiment indicates no significant improvements by employing representation models with increased model sizes. With these results, we expect that increasing the training data diversity is more desirable than increasing the model sizes in our current benchmark setting. Also, we add that the higher model capacity can be beneficial for developing multi-task policies with a greater number of tasks.

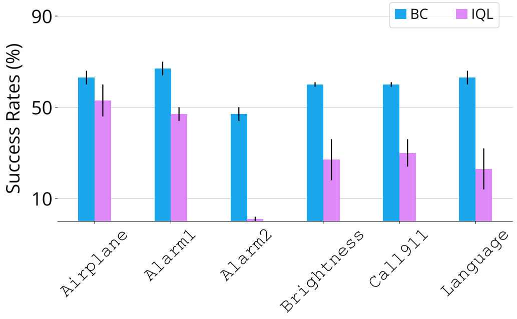


Figure 12: Success rates of VLUI agents trained with BC and IQL. Training with IQL does not result in as high performances as training with BC, presumably due to training instability.

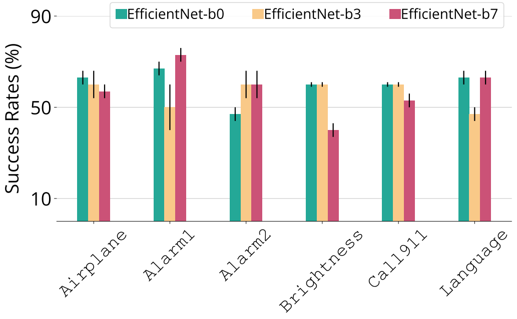


Figure 13: Success rates ov VLUI agents with varying size of visual encoders. We do not observe significant benefits by increasing the model capacities of representation models.