

# MOBILEAGENTBENCH: AN EFFICIENT AND USER-FRIENDLY BENCHMARK FOR MOBILE LLM AGENTS

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

Large language model (LLM)-based mobile agents are increasingly popular due to their capability to interact directly with mobile phone Graphic User Interfaces (GUIs) and their potential to autonomously manage daily tasks. Despite their promising prospects in both academic and industrial sectors, little research has focused on benchmarking the performance of existing mobile agents, due to the inexhaustible states of apps and the vague definition of feasible action sequences. To address this challenge, we propose an efficient and user-friendly benchmark, **MobileAgentBench**, designed to alleviate the burden of extensive manual testing. We initially define 100 tasks across 10 open-source apps, categorized by multiple levels of difficulty. Subsequently, we evaluate several existing mobile agents, including AppAgent and MobileAgent, to thoroughly and systematically compare their performance. All materials will be accessible on our project webpage, contributing to the advancement of both academic and industrial fields.

## 1 INTRODUCTION

With the emergence of large language models (LLMs) (Achiam et al., 2023), researchers have developed various autonomous agents across fields such as robotics (Bousmalis et al., 2023; Reed et al., 2022), games (Wang et al., 2023b; Du et al., 2023), and mobile phones (Yang et al., 2023b; Rawles et al., 2023). Among these, mobile agents have attracted significant attention due to their potential to enhance user experiences and provide intelligent assistance on-the-go.

People have been dreaming of Intelligent Personal Assistants (IPAs) (de Barcelos Silva et al., 2020) that can fully automate daily tasks for decades. Since Apple introduced its digital assistant, Siri (Apple, 2011) in 2011, almost all the leading technology companies have launched their own IPAs, including Microsoft Cortana (Microsoft, 2014), Amazon Alexa (Amazon, 2014), and Google Assistant (Google, 2016). While these digital assistants provide a hands-free human-computer interaction experience using Natural Language Interface (NLI), they can only fulfill relatively simple tasks, such as setting an alarm clock or sending a text message (Li et al., 2024). For third-party apps, developers have to follow and implement the application programming interfaces and protocols, such that when a user issues a very specific command, the system can invoke the corresponding functionality. This limits the usability of those digital assistants.

LLMs contribute significantly to resolving the persistent challenge of understanding user intent. The demonstrated reasoning ability (Qiao et al., 2022) highlights the potential of LLM-based autonomous agents as next-generation Intelligent Personal Assistants (IPAs), which are not limited by the programming interfaces since they directly operate on the Graphic User Interface (GUI) (Wang et al., 2024; Yang et al., 2023b). The GUI can either be represented by a text-based view tree to be consumed by a LLM or a screenshot image that can leverage a Multi-modal LLM (MLLM) (Yin et al., 2023). The action space of the agents composes a series of functions to simulate human operations, such as click, type, swipe, etc. In this way, LLM agents can theoretically achieve whatever human users can do, without any modification of the existing apps.

The promising future of LLM-based smartphone agents attracts more and more researchers to study this topic. However, the scope of benchmarks available for evaluating the performance of these agents remains constrained. Among the existing benchmarks, several prevalent issues are evident: **1. Scalability and usability.** Researchers need to fully understand complicated data structures and

Table 1: Comparison between the proposed and existing benchmarks

Benchmark	Fully Autonomous	Realistic Environment	Flexible Success Condition	Low Code Invasiveness	Large Scale
AppAgent (Yang et al., 2023b)	✗	✓	✓	✓	✗
AITW (Rawles et al., 2023)	✓	✗	✗	✗	✓
AndroidArena (Xing et al., 2024)	✓	✓	✗	✗	✓
AppBuddy (Shvo et al., 2021)	✓	✓	✓	✗	✗
AndroidEnv (Toyama et al., 2021)	✓	✓	✓	✗	✓
B-MoCA (Lee et al., 2024)	✓	✓	✓	✗	✗
AndroidWorld (Rawles et al., 2024)	✓	✓	✓	✗	✓
LLamaTouch (Zhang et al., 2024)	✓	✓	✓	✗	✓
MobileEnv (Zhang et al., 2023)	✓	✓	✓	✗	✓
MobileAgentBench (ours)	✓	✓	✓	✓	✓

tools before extending the benchmark with customized tasks or integrating it into their own codebases (Zhang et al., 2024). **2. Robustness and Flexibility.** Only the annotated task completion path is considered (Rawles et al., 2023; Xing et al., 2024). However, there might be multiple paths to successfully complete a task, which may break the task success judgment logic. **3. Realistic environment.** Some benchmarks evaluate the agent’s performance based on a collection of screenshots but not real devices. It fails if the agent performs an abnormal action and goes to an undefined state (Rawles et al., 2023).

To address the issues described, we propose a robust benchmark, MobileAgentBench, designed to evaluate the capabilities of mobile LLM agents within the Android ecosystem. MobileAgentBench offers several advantages over previous benchmarks due to its ease of use and minimally invasive nature. Specifically, for standard agents, the integration process requires fewer than ten lines of additional code. The benchmark excels in usability and versatility, supporting a broad spectrum of testing tasks across various Android operating system versions and executing on actual devices.

In this initial release, we offer 100 built-in benchmarking tasks spanning ten open-source applications. Notably, MobileAgentBench diverges from traditional approaches by simplifying the extension process. Third-party developers can specify the conditions for task success using just a few lines of Python code, without needing extensive knowledge in Android development. This accessibility makes MobileAgentBench more conducive to developers and researchers from non-Android Development communities. Furthermore, we introduce an innovative method for determining the task-terminating state, rendering the benchmark resistant to the complexities of tracking multiple potential success pathways. This approach ensures that MobileAgentBench provides reliable and precise benchmarking outcomes.

The comparison between the proposed and existing benchmarks is listed in Tab. 1, where fully autonomous represents if the benchmark does not need human supervision or judgment, realistic environment represents the tasks can be run on real devices, success condition flexibility represents it takes all possible success paths into consideration, low code invasiveness represents integrating the benchmark into existing agents does not need significant code changes.

Our **contributions** are summarized as follows:

- We propose a benchmark framework for mobile LLM agents. The new approach addresses common issues of existing benchmarks, making the evaluation process fully autonomous and reliable.
- We implement and test 100 benchmarking tasks with different levels of difficulty. The benchmark is plug-and-play and easy for both developing new agents and evaluating existing agents.
- We evaluate the performance of state-of-the-art mobile LLM agents and perform a solid and systematic comparison with our new benchmark, providing baseline data to the community.

## 2 RELATED WORK

### 2.1 MOBILE LLM AGENTS

Studies before the LLM-era employed reinforcement learning (RL) to solve the autonomous GUI navigation problem (Gur et al., 2018). The recent advancement of LLM and MLLM becomes the dominant agent paradigm because of the greater ability of UI understanding and reasoning. Early studies focused on web agents, which achieve task automation within browsers (Deng et al., 2024; Zhou et al., 2023). Recently, more and more studies started to investigate agents on mobile devices, especially on the Android platform, as Android smartphones are the most widely used personal computing devices.

Mobile LLM agents share a similar algorithm. The full input prompt often consists of four main components: the user prompt (task description), the current UI view hierarchy (VH) description, the action function list, and historical information. Specifically, the action list mainly includes click, swipe, type, and other common UI operations. If MLLM is used, the current screenshot is also a part of the input. The LLM/MLLM is asked to think of the next action based on the current and historical states and call the correct function to perform the given task step by step. The agent finally parses the model response and sends control signals to the Android device using Android Debug Bridge (ADB)<sup>1</sup>, UIAutomator<sup>2</sup>, or other higher-level UI automation frameworks.

Despite the similarity of the high-level ideas, researchers have developed different techniques to improve performance and efficiency. Among these works, AndroidArena (Xing et al., 2024) transforms the long and overwhelming view hierarchy XML into a compressed representation and assigned UI elements with unique node IDs, which shortens the prompt and makes the system more efficient. MobileAgent (Wang et al., 2024) observes that GPT-4V lacks the capability of UI element localization, and employs an Optical Character Recognition (OCR) model to locate and localize text views. Moreover, it uses the CLIP (Radford et al., 2021) and Grounding DINO (Liu et al., 2023) models to detect icons. AppAgent (Yang et al., 2023b) uses SoM (Yang et al., 2023a) prompts to localize UI elements and breaks tasks into two phases, exploration, and deployment. During the exploration phase, the agent automatically interacts with the apps and summarizes the observations into a document. In the deployment phase, it employs the Retrieval Augmented Generation (RAG) (Lewis et al., 2020) technique to utilize the summarized knowledge and improve success rate. CogAgent (Hong et al., 2023) proposes its own highly efficient 18B-parameter MLLM, which can be loaded on a single commercial GPU. Furthermore, Octopus v2 (Chen & Li, 2024) proposes a compact 3B-parameter model, which unlocks the potential to run mobile LLM agents on-device in an efficient and privacy-preserving manner.

### 2.2 BENCHMARKS FOR MOBILE LLM AGENTS

Since the Mobile LLM agent is a newly emerging research field, the choice of benchmarks is very limited. Some studies rely on verifying the task execution status manually to evaluate the performance (Yang et al., 2023b), which is tedious and time-consuming. To expedite the agent development, we need a fully autonomous benchmarking system to report various metrics, especially the task success rate. However, automatically judging if a task is completed successfully is non-trivial. The main challenge is caused by the dynamic nature of the GUI navigation task – the agent may perform random actions and drive the app to an unknown state.

AITW (Rawles et al., 2023) is a popular benchmark for mobile LLM agents. It has a large scale, but it’s based on static screenshot images. Thinking of each app state (screenshot) as a node, and each action as an edge, we can build a State Transferring Graph (STG) based on the screenshots and the human-annotated actions. It fails immediately if the agent performs actions in a non-considered sequence and leads the STG to a non-existent node, even if the agent can eventually complete the task.

The only solution is to identify task successes on real devices. One approach is to match the agent’s actions with the annotated ground truth (GT). A step-wise matching algorithm is not accurate, because the agent may not finish the task exactly in the same order with GT. AndroidArena (Xing

<sup>1</sup> <https://developer.android.com/tools/adb>

<sup>2</sup> <https://developer.android.com/training/testing/other-components/ui-automator>

et al., 2024) proposes an adaptive method of calculating the longest common subsequence of the agent and GT action sequences, which is illustrated as follows, where  $a$  and  $\hat{a}$  are the GT and actual actions, respectively. The common subsequences are marked in red.

$$\begin{aligned}
 a &= ABC & (1) \\
 \hat{a} &= \color{red}{A}XY\color{red}{B}UCVW & (2)
 \end{aligned}$$

AndroidArena (Xing et al., 2024) treats a task as successful if the GT is a subsequence of the actual action sequence. It addresses the issue of redundant actions but is still not optimal. A simple counter-example can be navigating from the page 1 to page 3 by clicking the next page button two times. If the agent performs the following sequence of actions: clicking the next page button, clicking the previous page button, and clicking the next page button, it doesn't navigate to the correct page but is still a subsequence of the GT. This method gives false positive results if the redundant action has a side effect.

A concurrent work, LlamaTouch (Zhang et al., 2024), addresses this problem by examining the final UI state, which is similar to our approach. We observe that despite the infinite feasible action sequences, the final success state convergence to one. The success or failure can be determined by checking the final UI state. An edge case is that some tasks may not have a direct UI representation, for example, the result of a network request triggered by a button may not be directly reflected on the current UI page. Thus, only checking the UI state is not sufficient and we need to incorporate actions, such as the clicking event, into consideration. LlamaTouch (Zhang et al., 2024) matches the click action by mapping the coordinate to a UI element, based on the view bounding boxes. However, it may not always be accurate. The process of finding the correct view to respond to a clicking event is called a hit test, and it's only accurate if performed by the Android UI system. This is because app developers can modify the touchable area, making it different from the view border to get better user experience.

Fig. 1 shows an example of enlarging the touchable area of a button view. In Fig. 1, touching point 1, the button does not respond because it's outside of the touchable area. Touching point 2, the button responds because it's inside the button view. Touching point 3, although it's outside of the visible button view, the button still responds because it's inside the extended touchable area. To overcome this difficulty, we utilize the Android Accessibility Service<sup>3</sup> to capture app events faithfully and forward them to the benchmark server. The details of our implementation are described in Sec. 3.1.

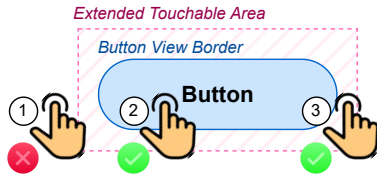


Figure 1: Extended touchable area.

Another concurrent work, AndroidWorld(Rawles et al., 2024), offers 126 tasks across 20 apps and has its own agent, M3A, which is benchmarked against others using automatic evaluations and precise rewards. We all emphasize the importance of real-device benchmarks to truly reflect performance and employ Python to gauge task success. This enhances flexibility and accuracy by capturing dynamic system states instead of relying on static UI matching. However, our approach differs significantly: we prioritize agent-centric development while AndroidWorld focuses on benchmark-centric strategies. Our MobileAgentBench integrates seamlessly with minimal code adjustments, supporting existing agents' action spaces through the Android Accessibility Framework. This enables less intrusive adaptations and broader evaluations beyond mere success rates, facilitating a more detailed assessment.

### 3 MOBILEAGENTBENCH

#### 3.1 METHOD

MobileAgentBench runs on real Android devices, supporting both physical devices and emulators. It sets up environment and then invokes the agent execution function. While the agent is operating

<sup>3</sup> <https://developer.android.com/reference/android/accessibilityservice/AccessibilityService>

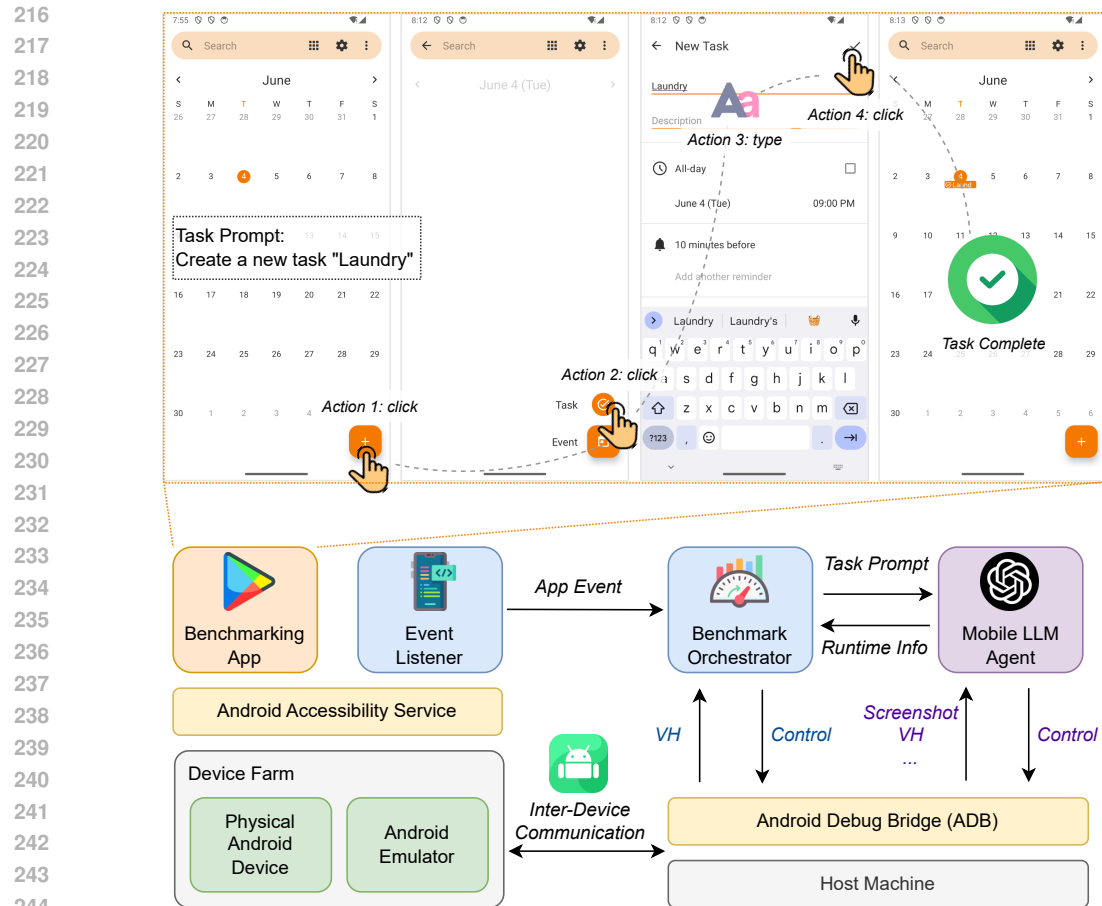


Figure 2: Overview of the MobileAgentBench architecture.

the device, MobileAgentBench judges the task success status in real time without any side effects. After the agent stops or exceeds the maximum steps, MobileAgentBench automatically switches to the next task. The whole process is fully automated and requires no human supervision.

The task success judgment mechanism is implemented by matching the final UI state, instead of examining the action sequence. This is because there might be multiple paths towards task completion, but the final success state converges to one. For example, if the task is to go to the settings page, agents may mistakenly open random pages before they correctly find the desired settings page. Matching the action sequence is difficult because of the randomness. On the contrary, checking if the top page is the settings page is easy and reliable. No matter what operations the agent does, we treat it as a success as long as the settings page is detected. ADB and UIAutomator are used to fetch the VH information. For each task, there is a Python file that defines the task success criteria, making it easy to extend and customize tasks for third-party developers.

As some tasks may not have a direct reflection on the current UI page, only checking the VH information is not sufficient. An example can be editing a note and saving the changes. Clicking the save button, the app may only pop up a temporary alert to indicate the save action has succeeded and stays on the current page. When the benchmark checks the current view state, it doesn't know if the save button is clicked or not. As a benchmark, it cannot go to other pages because it may change the app state and affect the next action of the agent. Since we want to determine the task success in real-time to collect how many steps the agent takes, it is not feasible to check UI states of other pages after the agent stops. Besides, some agents have the problem of not being able to stop gracefully even the task is completed. We address this issue by incorporating app events, especially button click action signals. For the above-mentioned task, we can use the view hierarchy to check if

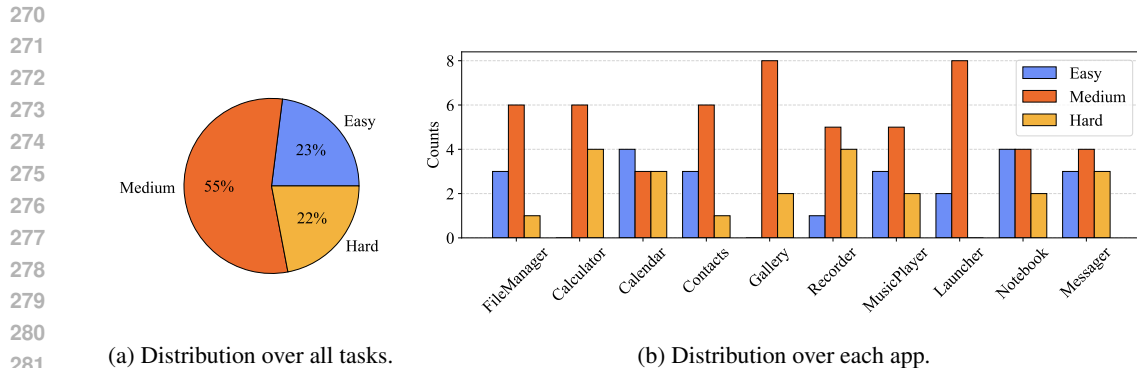


Figure 3: The distribution of task difficulty levels.

the note is edited correctly, and then mark the task as a success if the save button clicking signal is received afterwards.

To faithfully receive the app event signals, we make an Android app using the Android Accessibility Service. Android Accessibility Service was originally designed to help people with disabilities. It runs in the background and invokes a callback function when the Android system fires an accessibility event. Such events include most UI state transitions by the user (agent) interactions, such as button clicking, window changing, etc, which fulfills the needs of the proposed benchmark.

The overview of MobileAgentBench is shown in Fig. 2. The benchmarking apps run on real devices from a device farm, which can either be a physical device or an emulator. The device talks to the host machine of the benchmark and the agent via ADB. The benchmark and the agent use ADB to retrieve app state information, such as screenshots, view hierarchy, and send control signals. The benchmark invokes the agent with the current benchmarking task prompt and collects runtime information from the agent, such as the LLM input and output. Whenever the event listener app receives an app event, it forwards the event to the benchmark server via a socket, so the benchmark can assess the task success status using both VH and the actions. At the top of Fig. 2, we show a sample task workflow, “Create a new task *Laundry*” with the Calendar app. The agent needs to perform 4 actions to complete the task: clicking the add button, clicking the task button, typing the task name “Laundry”, and clicking the save button. The benchmark checks the content of the task name input box view and listens to the save button clicking event to determine if the task is finished successfully.

### 3.2 TASK DESCRIPTION

In our initial version of the benchmark, we implement 100 tasks over 10 daily apps. The 10 apps are from SimpleMobileTools<sup>4</sup>, an open-source project of Android apps. These apps have simple and straightforward user interfaces, without any advertisements or unnecessary permissions, and thus are great for benchmarking use cases. The full list of app names is shown in Fig. 3b.

We carefully design the benchmarking tasks, so that they can simulate a normal user’s daily activity and have multi-level difficulties. The distribution of task difficulty levels is shown in Fig. 3. Difficulties are defined by the minimum steps to finish a task, which is cross-verified by 3 human experts independently. A task would be classified as an *easy* task if it can be done within 2 steps, *medium* if greater or equal to 3 while less than 6, and otherwise, *hard*.

### 3.3 USAGE

The benchmark APIs are designed to be user-friendly and as less invasive as possible. For a standard agent, it takes less than 10 lines of additional code to integrate. List. 1 shows the pseudo-code of the benchmark usage. First, we need to import the benchmark Python library and initialize the benchmark orchestrator. Next, the main agent entrance function should be defined. It takes and

<sup>4</sup> <https://simplemobiletools.com>, GPL-3.0

only takes one parameter, the task prompt. The agent iteratively performs actions to finish the task. Before and after the agent performs one action, the orchestrator functions are called, so information such as the time spent and LLM output can be collected. The program starts with the orchestrator run function. It calls the agent entrance function with each task prompt and automatically switches to the next task after the task finishes. The task success status is judged on the fly.

```

1 from mobile_agent_benchmark.task_orchestrator import TaskOrchestrator
2 orchestrator = TaskOrchestrator() # the MobileAgentBench orchestrator
3 # the agent entrance function
4 def agent_run(task_prompt):
5     while not done:
6         orchestrator.before_one_action()
7         # the agent invokes a LLM to think about the next action
8         action = llm_think(task_prompt, screenshot)
9         agent_perform(action)
10        orchestrator.after_one_action(action)
11 orchestrator.run(agent_run)

```

Listing 1: Pseudo code of integrating MobileAgentBench into an existing Mobile LLM Agent.

## 4 EXPERIMENTS AND AGENT EVALUATIONS

### 4.1 METRICS

We define 6 metrics to comprehensively benchmarking mobile agents:

- Success Rate (SR):  $SR = N_{success}/M_{tasks}$ , where  $N_{success}$  is the number of successful tasks, judged by the benchmark system.  $M_{tasks}$  is the number of total benchmarking tasks. This metric reflects the agent’s ability to correctly finish a task end-to-end.
- Step-wise Efficiency (SE).  $SE = S_{actual}/S_{min}$ , where  $S_{actual}$  is the number of actual steps the agent takes to successfully finish a task, and  $S_{min}$  is the minimum number of steps. This metric tells us if the agent performs unnecessary or redundant actions and reflects the efficiency of the agent. Failure tasks are not taken into account.
- Latency. The average time spent in seconds before and after one action. This metric tells us how long a user needs to wait between two actions.
- Tokens. The number of LLM input and output tokens. For simplicity, we use the GPT-4V (gpt-4-vision-preview) standard (OpenAI) to calculate the number of tokens for all models, which gives us a rough estimation of the LLM cost. For text, 1 token is 4 characters. For an image, it’s divided into multiple  $512 \times 512$  tiles, and each tile is 170 tokens. 85 base tokens are applied to each image as well.
- False Finish Rate (FFR).  $FFR = N_{early}/M_{failure}$ , where  $N_{early}$  is the number of early stopped tasks and  $M_{failure}$  is the total number of failure tasks. This metric represents how likely the agent falsely thinks it has finished the task and stopped early.
- Over Execution Rate (OER).  $OER = N_{late}/M_{success}$ , where  $N_{late}$  is the number of late stopped tasks and  $M_{success}$  is the total number of successful tasks. Symmetrically to FFR, this metric reveals how likely the agent falsely thinks the task is not finished successfully.

### 4.2 ENVIRONMENT SETUP

Five popular mobile LLM agents, AndroidArena (Xing et al., 2024), AutoDroid (Wen et al., 2023), AppAgent (Yang et al., 2023b), CogAgent (Hong et al., 2023), and MobileAgent (Wang et al., 2024) are evaluated with the proposed benchmark. We choose the Google Pixel 3a emulator and Android 14 operating system to run the benchmarking apps. Besides, the Android 9 operating system is used for AutoDroid as some of the dependency libraries do not support the newer Android systems.

As there are no local neural networks used in AndroidArena, AutoDroid, and AppAgents, these agents are executed on an Apple Macbook Pro with the M1 Max chip. CogAgent and MobileAgent require local model referencing, so they are executed on a workstation equipped with a single Nvidia RTX 4090 GPU, with 24 GB GPU memory.

Table 2: Agent performance results with multiple metrics

Agent	Models	SR $\uparrow$	SE $\downarrow$	Latency $\downarrow$	Tokens $\downarrow$	FFR $\downarrow$	OER $\downarrow$
AndroidArena (Xing et al., 2024)	GPT4-V (Achiam et al., 2023)	0.22	<b>1.13</b>	18.61	750.47	<b>0.09</b>	0.33
AutoDroid (Wen et al., 2023)	GPT3.5 Instructor (Su et al., 2022)	0.27	3.10	<b>4.85</b>	963.48	0.93	<b>0.01</b>
AppAgent (Yang et al., 2023b)	GPT4-V (Achiam et al., 2023)	<b>0.40</b>	1.29	26.09	1505.09	0.17	0.40
CogAgent (Hong et al., 2023)	CogVLM (Wang et al., 2023a)	0.08	2.42	6.76	<b>579.84</b>	1.00	0.04
MobileAgent (Wang et al., 2024)	GPT4-V (Achiam et al., 2023) GroundingDINO (Liu et al., 2023) ViT-B/32 (Radford et al., 2021) DamoOCR (Wang et al., 2022)	0.26	1.33	15.91	1236.88	0.19	0.31

The self-exploration feature is turned on for AppAgent. When performing a task, it can reference the previously summarized document. For CogAgent, we use 4-bit quantization to load the model due to GPU memory limitation. CogAgent is implemented in its vanilla flavor, *i.e.*, given the current screenshot, ask for the next action. No history information is provided.

### 4.3 RESULTS

The main experiment results are shown in Tab. 2. We observe that AppAgent has the highest success rate, benefiting from the self-exploration mechanism. CogAgent has the lowest success rate, most likely caused by the naive agent implementation, which limits the usage of history information. Although AutoDroid has a similar success rate to MobileAgent, the step-wise efficiency is significantly lower, possibly caused by the weaker reasoning capability of the GPT-3.5 model used by AutoDroid. Latency-wise, both AutoDroid and CogAgent have very low latency, indicating the high inferencing cost with the GPT-4V model. AppAgent needs to look up the app document, thus consuming more tokens than others. On the other hand, because of the naive agent implementation of CogAgent, it consumes the least number of tokens. AutoDroid and CogAgent have very high FFR, indicating they always stop early when the task is not finished yet. AppAgent, although having the highest task success rate, is not good at determining the task success, it cannot stop gracefully after finishing a task and has a high OER.

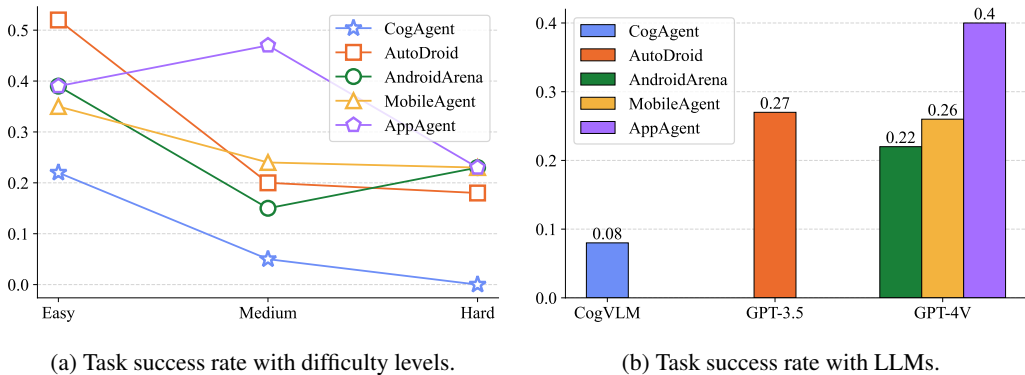


Figure 4: Task success rate.

The task success rates for each agent with difficulty levels are shown in Fig. 4a. From Fig. 4a, we can see most agents have higher success rates when handling easier tasks, which is as expected.



432 Interestingly, AppAgent has a higher success rate when performing medium-level tasks. This is  
 433 because we set the maximum execution steps as twice the minimum steps, which would make the  
 434 agents have very limited steps to correct their earlier steps for easy tasks. For example, an agent can  
 435 only use 1 additional step to correct and finish the task for a 1-step easy task. However, for medium  
 436 and hard tasks, there is significantly more space to correct the previous actions.

437 Fig. 4b shows the averaged task success rate over the backbone LLM models. It is interesting that  
 438 AutoDroid, although using a text-based GPT-3.5 model, outperforms some other agents that use  
 439 the more advanced GPT-4V model. This reveals that the textual view hierarchy contains the most  
 440 important information for GUI navigation tasks. However, we believe that visual screenshots are  
 441 helpful for other types of tasks, for example, if the task involves recognizing an image on the screen,  
 442 or if the textual view hierarchy is not available.

## 444 5 LIMITATIONS AND FUTURE WORK

446 While the proposed MobileAgentBench is efficient, user-friendly, and addresses many issues of the  
 447 existing benchmarks for mobile LLM agents, there are two main limitations that the authors would  
 448 like to improve in the future. Firstly, although the use of Python code snippets as the configuration  
 449 of task success conditions is easy for researchers in the AI/ML community, it is still difficult for  
 450 people without a technical background. We will explore new methods to automatically build the  
 451 task configuration without writing any code in the future work.

## 453 6 CONCLUSION

455 In this paper, we propose a new benchmark, MobileAgentBench, for mobile LLM agents on the  
 456 Android platform. With the 100 built-in benchmarking tasks, researchers can test and evaluate  
 457 existing and new agents automatically on real Android devices. Extending the benchmark to support  
 458 customized tasks is also easy, as only basic Python coding skills are needed. Leveraging the Android  
 459 Accessibility Services and only checking the final app state, MobileAgentBench can detect task  
 460 completion status faithfully. We report the evaluation results of 5 popular agents across multiple  
 461 metrics, and they can be used as strong baselines to advance future mobile LLM agent development.

## 463 7 REPRODUCIBILITY

465 Our source code is anonymously available at [https://anonymous.4open.science/r/  
 466 mobile-agent-bench-E727](https://anonymous.4open.science/r/mobile-agent-bench-E727).

## 468 REFERENCES

- 470 Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Ale-  
 471 man, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical  
 472 report. *arXiv preprint arXiv:2303.08774*, 2023.
- 473 Amazon. Alexa. <https://alexa.amazon.com>, 2014.
- 474 Apple. Siri. <https://www.apple.com/siri/>, 2011.
- 475 Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza,  
 476 Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving  
 477 foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.
- 478 Wei Chen and Zhiyuan Li. Octopus v2: On-device language model for super agent. *arXiv preprint  
 479 arXiv:2404.01744*, 2024.
- 480 Allan de Barcelos Silva, Marcio Miguel Gomes, Cristiano André da Costa, Rodrigo da Rosa Righi,  
 481 Jorge Luis Victoria Barbosa, Gustavo Pessin, Geert De Doncker, and Gustavo Federizzi. Intelli-  
 482 gent personal assistants: A systematic literature review. *Expert Systems with Applications*, 147:  
 483 113193, 2020.

- 486 Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su.  
487 Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing*  
488 *Systems*, 36, 2024.
- 489 Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek  
490 Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language  
491 models. In *International Conference on Machine Learning*, pp. 8657–8677. PMLR, 2023.
- 492 Google. Google assistant. <https://assistant.google.com>, 2016.
- 493 Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to navigate the  
494 web. *arXiv preprint arXiv:1812.09195*, 2018.
- 495 Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan  
496 Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. *arXiv*  
497 *preprint arXiv:2312.08914*, 2023.
- 500 Juyong Lee, Taywon Min, Minyong An, Changyeon Kim, and Kimin Lee. Benchmarking mobile  
501 device control agents across diverse configurations. *arXiv preprint arXiv:2404.16660*, 2024.
- 502 Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal,  
503 Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented genera-  
504 tion for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:  
505 9459–9474, 2020.
- 506 Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu,  
507 Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the  
508 capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.
- 509 Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei  
510 Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for  
511 open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.
- 512 Microsoft. Cortana. <https://www.microsoft.com/en-us/cortana>, 2014.
- 513 OpenAI. Gpt token calculation. <https://openai.com/api/pricing/>.
- 514 Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei  
515 Huang, and Huajun Chen. Reasoning with language model prompting: A survey. *arXiv preprint*  
516 *arXiv:2212.09597*, 2022.
- 517 Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal,  
518 Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual  
519 models from natural language supervision. In *International conference on machine learning*, pp.  
520 8748–8763. PMLR, 2021.
- 521 Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the  
522 wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023.
- 523 Christopher Rawles, Sarah Clinckemaiellie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Mary-  
524 beth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A  
525 dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*,  
526 2024.
- 527 Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov,  
528 Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al.  
529 A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.
- 530 Maayan Shvo, Zhiming Hu, Rodrigo Toro Icarte, Iqbal Mohamed, Allan D Jepson, and Sheila A  
531 McIlraith. Appbuddy: Learning to accomplish tasks in mobile apps via reinforcement learning.  
532 In *Canadian AI*, 2021.
- 533 Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih,  
534 Noah A Smith, Luke Zettlemoyer, and Tao Yu. One embedder, any task: Instruction-finetuned  
535 text embeddings. *arXiv preprint arXiv:2212.09741*, 2022.

- 540 Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali  
541 Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning  
542 platform for android. *arXiv preprint arXiv:2105.13231*, 2021.
- 543
- 544 Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao  
545 Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv*  
546 *preprint arXiv:2401.16158*, 2024.
- 547 Peng Wang, Cheng Da, and Cong Yao. Multi-granularity prediction for scene text recognition. In  
548 *European Conference on Computer Vision*, pp. 339–355. Springer, 2022.
- 549
- 550 Weihang Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang,  
551 Lei Zhao, Xixuan Song, et al. Cogvlm: Visual expert for pretrained language models. *arXiv*  
552 *preprint arXiv:2311.03079*, 2023a.
- 553 Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe,  
554 explain, plan and select: Interactive planning with large language models enables open-world  
555 multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023b.
- 556
- 557 Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yun-  
558 hao Liu, Yaqin Zhang, and Yunxin Liu. Empowering llm to use smartphone for intelligent task  
559 automation. *arXiv preprint arXiv:2308.15272*, 2023.
- 560 Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. Understanding the  
561 weakness of large language model agents within a complex android environment. *arXiv preprint*  
562 *arXiv:2402.06596*, 2024.
- 563 Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark  
564 prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*,  
565 2023a.
- 566
- 567 Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent:  
568 Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023b.
- 569 Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on  
570 multimodal large language models. *arXiv preprint arXiv:2306.13549*, 2023.
- 571
- 572 Danyang Zhang, Hongshen Xu, Zihan Zhao, Lu Chen, Ruisheng Cao, and Kai Yu. Mobile-env:  
573 an evaluation platform and benchmark for llm-gui interaction. *arXiv preprint arXiv:2305.08144*,  
574 2023.
- 575 Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and  
576 Mengwei Xu. Llamatouch: A faithful and scalable testbed for mobile ui automation task evalua-  
577 tion. *arXiv preprint arXiv:2404.16054*, 2024.
- 578
- 579 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,  
580 Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building  
581 autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.
- 582
- 583
- 584
- 585
- 586
- 587
- 588
- 589
- 590
- 591
- 592
- 593

## 594 A APPENDIX

### 595 A.1 BENCHMARKING APPS

596 We choose 10 open-source daily apps from SimpleMobileTools<sup>5</sup> in our initial benchmark, a collec-  
599 tion of simple Android apps without advertisements or unnecessary permissions. However, users  
600 can easily add their customized tasks with other apps. The reason that we choose open-source apps  
601 is that they are static and reproducible. Commercial apps often employ online A/B testing (?) mech-  
602 anisms and may display UI elements and functionalities differently to different users, or change  
603 them over time.

604 The full list of the currently built-in apps is shown as follows:

- 605 • Calculator
- 606 • Calendar
- 607 • Contacts
- 608 • FileManager
- 609 • Gallery
- 610 • AppLauncher
- 611 • Messenger
- 612 • MusicPlayer
- 613 • Notes
- 614 • Recorder

### 615 A.2 CUSTOMIZATION

616 The 100 built-in tasks make the benchmark work out of the box. However, third-party researchers  
617 may want to implement their customized tasks. For example, benchmarking their agent performance  
618 on commercial apps. This can be done easily without modifying the benchmark library. First, imple-  
619 ment the customized task class inheriting the Task class provided by the benchmark framework. The  
620 most important method to implement is `check_finish`, which defines how to judge if the given  
621 task is finished successfully. If necessary, the `setup` and `teardown` function can also be imple-  
622 mented to prepare and reset the environment to execute the current task. Then, add the task names  
623 to a JSON file and pass it to the benchmark’s task orchestrator. When running, the task orchestrator  
624 will invoke each task (built-in or customized) one by one, automatically.

### 625 A.3 EXPERIMENTS WITH ONLINE APPS

626 In this paper, we use open-source and offline apps because we want to create a fully controlled  
627 and reproducible environment for all agents. We picked apps such as Calendar and SMS that are  
628 commonly used by people for everyday tasks. Online apps, on the contrary, have dynamic content,  
629 which may make the agent’s performance non-deterministic. Furthermore, even if the online app  
630 versions are the same (e.g. installing apps with the same APK), modern apps often have online A/B  
631 testing systems, which means the functionality or even UI appearance may not be the same for every  
632 user.

633 However, some researchers might be interested in benchmarking their agents with online apps. The  
634 proposed benchmark is designed to be easy to extend, so researchers can freely add their customized  
635 tasks with online apps. To demonstrate the extensibility of MobileAgentBench, we have imple-  
636 mented 6 tasks across 3 popular online apps: CNN News, Amazon Shopping, and Google Maps.  
637 Here is the full list of the currently implemented tasks.

- 638 • [Google Map] navigate from Bellevue, WA to Redmond, WA
- 639 • [Google Map] search Redmond, WA
- 640 • [Amazon Shopping] search nike and then show me shoes

641 <sup>5</sup> <https://simplemobiletools.com>

- [Amazon Shopping] show me the size guide for nike shoes
- [CNN News] show news under the climate section
- [CNN News] enable the CNN sound effect when opening the app

Tab. 3 shows experiment results on the subset of online apps. The metrics are the same as defined in Sec. 4.1.

Table 3: Agent performance results on Online Apps

Agent	SR $\uparrow$	SE $\downarrow$	Latency $\downarrow$	Tokens $\downarrow$	FFR $\downarrow$	OER $\downarrow$
AndroidArena (Xing et al., 2024)	0.33	2.40	13.95	2305.31	<b>1.00</b>	0.50
AutoDroid (Wen et al., 2023)	0.00	N/A	<b>2.49</b>	<b>524.96</b>	N/A	<b>0.00</b>
AppAgent (Yang et al., 2023b)	<b>0.50</b>	1.78	19.8	1518.62	0.33	0.33
CogAgent (Hong et al., 2023)	0.17	2.33	7.23	587.91	<b>1.00</b>	<b>0.00</b>
MobileAgent (Wang et al., 2024)	<b>0.50</b>	<b>1.33</b>	13.65	1225.67	0.33	<b>0.00</b>

#### A.4 TASK SUCCESS CONDITION

List 2 shows a simple example of the `check_finish` function. To eliminate the difficulty of dealing with multiple possible action paths to success, we determine if the task is successful by only observing the final state. In this example, the task is to open the About page of the current app. We check if the Frequently Asked Questions text view (a child view in the About page) exists in the current UI page. If it exists, it means the About page is opened successfully and the task is completed. Otherwise, the task is not completed yet. It gives faithful results no matter how many ways there are to open the About page.

MobileAgentBench automatically invokes the `check_finish` function at the proper time: after one agent action, and when getting one app event. The `check_finish` function can be either stateless or stateful, depending on the task. The function has two parameters, the `view_client` and the `app_event`. `view_client` provides a set of useful functions to obtain the current page’s view hierarchy. `app_event` provides the most recent user action sent from the Android Accessibility Service, such as a button-clicking event. It helps to check the task-finishing status when it’s not directly reflected in the current UI view.

```

683 1 def check_finish(self, view_client, app_event) -> bool:
684 2     title_views = view_client.findViewById("Frequently asked
685 3     questions")
686 4     if title_views is not None:
687 5     return True
688 6     return False

```

Listing 2: A simple example of the `check_finish` function

#### A.5 VIEW HIERARCHY AND ACCESSIBILITY EVENTS

MobileAgentBench utilizes `AndroidViewClient`, a popular Android automation framework, to obtain view hierarchy. `AndroidViewClient` has multiple backends, including `UIAutomator`, `ViewServer`, and `CulebraTester2`. In this paper, we use the `UIAutomator` as the backend to conduct experiments. However, it’s worth exploring other backends that may have better performance. After getting the view hierarchy, the benchmark can access the properties of the views displayed on the screen. This provides a strong signal to check the current app status. For example, the benchmark can check if the query was correctly entered in a search bar.

To listen to the Android UI events, including button clicks, page changes, etc., we implement an Android app, `AndroidEventListener`. It uses the `AccessibilityService` and registers an event handler. Before launching the benchmark, the `AndroidEventListener` needs to be turned on on the system

702 settings, accessibility page. Then the process runs in the background and does not affect the fore-  
 703 ground testing apps. Once an event is captured, it serializes the event object into JSON and sends  
 704 it to the benchmark server using a socket. The benchmark consumes all events one by one after the  
 705 agent performs an action.  
 706

#### 707 A.6 TASK DESCRIPTION

708  
 709 The full list of tasks is shown in Tab. 4. All tasks are carefully designed to reflect the functions of  
 710 the corresponding app, with different difficulty levels. The difficulty levels are determined by the  
 711 minimum steps to accomplish the task.  
 712

713 Table 4: Sample task prompts with their difficulty levels

715 App Name	716 Difficulty Level	717 Task Prompt
718 Calculator	719 Medium	720 Calculate the result of '3 + 5'
721 Calculator	722 Medium	723 Use Unit converter function to 724 calculate how many kilometers 1mile is equal to
725 Calculator	726 Medium	727 Calculate the result of '3 ^ 3'
728 Calculator	729 Medium	730 Calculate the result of '12 ÷ 3'
731 Calculator	732 Medium	733 Calculate the result of '12 - 4'
734 Calculator	735 Hard	736 Calculate the result of '18+(24×3)-(9÷3)'
737 Calculator	738 Medium	739 Calculate the result of '12*4'
740 Calculator	741 Hard	742 Calculate the result of '19.7 - 81.3'
743 Calculator	744 Medium	745 Calculate the result of '12 × 5'. 746 However, during the input process, 747 the number '4' was mistakenly entered instead of '5'. 748 Correct this by first enter 'C' to delete '4' 749 and re-entering '5' and then perform the calculation
750 Calculator	751 Medium	752 Calculate the result of ' $\sqrt{16} + 3$ '
753 Calendar	754 Medium	755 Create a new event 'laundry' and then search for it
756 Calendar	757 Easy	758 switch to daily view
759 Calendar	760 Medium	761 Show events in simple event list, 762 delete the laundry and meeting events.
763 Calendar	764 Medium	765 Create a new event 'laundry'
766 Calendar	767 Medium	768 Create a new task, named 'laundry', 769 with the description of 'wash all my clothes'. 770 Mark it as all-day.
771 Calendar	772 Easy	773 show events of next month
774 Calendar	775 Easy	776 open about page
777 Calendar	778 Easy	779 search event 'laundry'
780 Calendar	781 Medium	782 change snooze time to 1 minute
783 Calendar	784 Medium	785 go to settings and make weeks start on Monday
786 Contacts	787 Easy	788 open About View
789 Contacts	790 Medium	791 Create a new contact, 792 his First Name is Yuzai, and his Phone Number is 793 123456789

756	Contacts	Medium	Delete contact Yuzai
757	Contacts	Easy	Set the contact Yuzai to Favorite
758	Contacts	Medium	Change phone filter, which means don't show phone storage in contacts view
759	Contacts	Medium	Set not show contact's Email in the contact profile screen
760	Contacts	Medium	Change the contact Yuzai's number to 987654321 and save it
761	Contacts	Medium	Remove the dialog button, and then return to the main view
762	Contacts	Easy	Search contact Yuzai
763	Contacts	Medium	Sort contacts by Data created time, Descending
764	Contacts	Medium	Sort contacts by Data created time, Descending
765	Contacts	Medium	Sort contacts by Data created time, Descending
766	Contacts	Medium	Sort contacts by Data created time, Descending
767	Contacts	Easy	Search contact Yuzai
768	Contacts	Medium	Sort contacts by Data created time, Descending
769	Contacts	Medium	Sort contacts by Data created time, Descending
770	Contacts	Medium	Sort contacts by Data created time, Descending
771	File Manager	Easy	open the folder 'Downloads' and check the properties of the file 'testfile.txt'
772	File Manager	Easy	check the storage page
773	File Manager	Medium	Create a new file named 'testfile.txt' in the 'Downloads' folder
774	File Manager	Medium	Delete file named 'testfile.txt' in the 'Downloads' folder
775	File Manager	Medium	Delete the txt file in Download folder
776	File Manager	Medium	Delete the txt file in Download folder
777	File Manager	Medium	Delete the txt file in Download folder
778	File Manager	Hard	Delete all videos in Download folder
779	File Manager	Medium	Hide the folder named 'hidden' and make sure File Manager Stop showing hidden media
780	File Manager	Medium	open the folder 'Downloads' and rename the file 'Testfile.txt' to 'testfile1.txt'
781	File Manager	Medium	open the folder 'Downloads' and rename the file 'Testfile.txt' to 'testfile1.txt'
782	File Manager	Easy	Search a file named 'testfile.txt'
783	File Manager	Medium	Search a file named 'testfile.txt'
784	File Manager	Easy	Search a file named 'testfile.txt'
785	File Manager	Medium	In the main page, sort the folder by size in descending order
786	File Manager	Medium	In the main page, sort the folder by size in descending order
787	Gallery	Medium	filter media in the gallery and only show images and videos
788	Gallery	Medium	filter media in the gallery and only show images and videos
789	Gallery	Medium	Go to the downloads folder, group the images by file type
790	Gallery	Medium	Change the view type to list view
791	Gallery	Medium	Go to Gallery settings and enable play videos automatically
792	Gallery	Medium	Enable remember the last video playback position in settings
793	Gallery	Medium	Enable remember the last video playback position in settings
794	Gallery	Medium	Go to Downloads Folder and set the first image as favorite
795	Gallery	Medium	Go to Downloads Folder and set the first image as Home screen wallpaper
796	Gallery	Medium	Go to Downloads Folder and set the first image as Home screen wallpaper
797	Gallery	Medium	show hidden items in the gallery in settings
798	Gallery	Medium	show hidden items in the gallery in settings
799	Gallery	Medium	sort the gallery by size ascendingly
800	Gallery	Medium	sort the gallery by size ascendingly
801	Gallery	Medium	Change the date and time format to 24-hour format in gallery settings
802	App Launcher	Medium	Add Chrome and Camera to launcher
803	App Launcher	Medium	Add Chrome and Camera to launcher
804	App Launcher	Medium	Check who is the contributor of the app
805	App Launcher	Easy	Hide app name in Launcher
806	App Launcher	Medium	Open About page and go Frequently Asked Questions
807	App Launcher	Medium	Open About page and go Frequently Asked Questions
808	App Launcher	Medium	Remove Chrome from Launcher
809	App Launcher	Medium	Rename Chrome in Launcher to MyChrome

810	App Launcher	Easy	Search for Chrome in Launcher
811	App Launcher	Medium	Change Setting Close this app at launching a different one to false
812	App Launcher	Medium	Sort apps by custom
813	App Launcher	Medium	Sort apps by title descending
814	App Launcher	Medium	Sort apps by title descending
815	App Launcher	Medium	Sort apps by title descending
816	SMS Messenger	Medium	Add a number '123456789' to block list
817	SMS Messenger	Medium	Change the Font size to 'Large' in the settings interface
818	SMS Messenger	Medium	open the conversation with contact number '123456789',
819	SMS Messenger	Medium	and check for a random message's properties
820	SMS Messenger	Medium	and check for a random message's properties
821	SMS Messenger	Hard	start a conversation with number '123456789',
822	SMS Messenger	Hard	send a message 'i luv u', and check for message properties
823	SMS Messenger	Hard	start a conversation with number '123456789',
824	SMS Messenger	Hard	and send a message 'i luv u',
825	SMS Messenger	Hard	back to the main page and
826	SMS Messenger	Hard	search for the contact '123456789'
827	SMS Messenger	Medium	make the conversation with number '123456789' archived
828	SMS Messenger	Easy	search for the contact '123456789' at top search bar
829	SMS Messenger	Easy	search for the contact '123456789' at top search bar
830	SMS Messenger	Easy	search message 'i luv u' at the top search bar
831	SMS Messenger	Easy	show me the archived conversations
832	SMS Messenger	Easy	show me the archived conversations
833	SMS Messenger	Hard	start a conversation with number '123456789',
834	SMS Messenger	Hard	and send a message 'i luv u'
835	Music Player	Medium	sort the album by 'year'
836	Music Player	Medium	config equalizer to Heavy Metal
837	Music Player	Medium	config equalizer to Heavy Metal
838	Music Player	Medium	create a new playlist:test
839	Music Player	Hard	create a new playlist: test, and search for it
840	Music Player	Hard	create a new playlist: test,
841	Music Player	Hard	and sort all playlist by descending order
842	Music Player	Medium	open faq page
843	Music Player	Medium	open faq page
844	Music Player	Easy	open setting page
845	Music Player	Medium	open setting page
846	Music Player	Medium	sort the playlist by 'desc'
847	Music Player	Easy	rescan media files
848	Music Player	Easy	search playlist 'Test'
849	Notes	Medium	add a new note named 'TODO List'
850	Notes	Medium	add a new note named 'TODO List'
851	Notes	Medium	delete the "to_do.list" and "meeting" note
852	Notes	Easy	Check the item 'eggs' for shopping_list
853	Notes	Medium	use the pin '2580' to open the locked note 'password.list'
854	Notes	Medium	use the pin '2580' to open the locked note 'password.list'
855	Notes	Medium	add a new Checklist named 'TODO List'
856	Notes	Hard	create a checklist named
857	Notes	Hard	'Shopping list' and add an item named 'Milk'
858	Notes	Easy	open the note 'meeting'
859	Notes	Easy	open the note 'meeting'
860	Notes	Easy	open about page
861	Notes	Medium	open about page
862	Notes	Medium	rename the current note to 'finished_task'
863	Notes	Easy	search 'secret' in note 'Charles's secrets'



864	Voice Recorder	Medium	change bitrate to 32 kbps
865	Voice Recorder	Medium	delete the last recorded audio
866	Voice Recorder	Hard	delete all recorded audio
868	Voice Recorder	Medium	go to settings and empty the recycle bin
869	Voice Recorder	Medium	use mp3 as the format for new recordings
871	Voice Recorder	Easy	go to recycle bin page
872	Voice Recorder	Medium	change settings, so that the deleted items will not go to recycle bin
874	Voice Recorder	Medium	rename the first audio to 'voice.m4a'
875	Voice Recorder	Hard	rename all audio to voice1.m4a, voice2.m4a, and so on
876	Voice Recorder	Hard	
877	Voice Recorder	Medium	change app theme to dark red

---

879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917