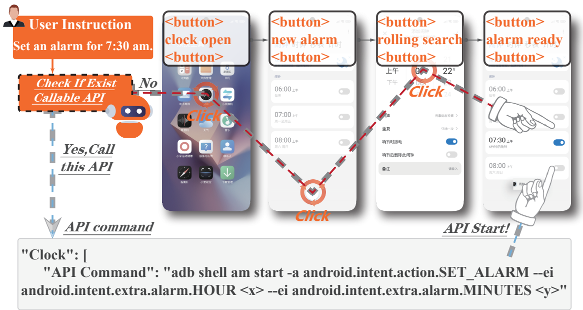
Mobile-Bench: An Evaluation Benchmark for LLM-based Mobile Agents

Shihan Deng[13* ‡], **Weikai Xu**[13* ‡], **Hongda Sun**[23* ‡], **Wei Liu**[3], **Tao Tan**[2], **Jianfeng Liu**[3],
Ang Li[3], **Jian Luan**[3], **Bin Wang**[3], **Rui Yan**[2†] **and Shuo Shang**[1†]

[1]University of Electronic Science and Technology of China
[2]Gaoling School of Artificial Intelligence, Renmin University of China
[3]XiaoMi AI Lab
{dengshihan7, xuwk266, jedi.shang}@gmail.com
{sunhongda98, ruiyan}@ruc.edu.cn

Abstract

With the remarkable advancements of large language models (LLMs), LLM-based agents have become a research hotspot in human-computer interaction. However, there is a scarcity of benchmarks available for LLM-based mobile agents. Benchmarking these agents generally faces three main challenges: (1) The inefficiency of UI-only operations imposes limitations to task evaluation. (2) Specific instructions within a singular application lack adequacy for assessing the multi-dimensional reasoning and decision-making capacities of LLM mobile agents. (3) Current evaluation metrics are insufficient to accurately assess the process of sequential actions. To this end, we propose Mobile-Bench, a novel benchmark for evaluating the capabilities of LLM-based mobile agents. First, we expand conventional UI operations by incorporating 103 collected APIs to accelerate the efficiency of task completion. Subsequently, we collect evaluation data by combining real user queries with augmentation from LLMs. To better evaluate different levels of planning capabilities for mobile agents, our data is categorized into three distinct groups: SAST, SAMT, and MAMT, reflecting varying levels of task complexity. Mobile-Bench comprises 832 data entries, with more than 200 tasks specifically designed to evaluate multi-APP collaboration scenarios. Furthermore, we introduce a more accurate evaluation metric, named CheckPoint, to assess whether LLM-based mobile agents reach essential points during their planning and reasoning steps. Dataset and platform are available at https://github.com/XiaoMi/MobileBench.


User Instruction
Set an alarm for 7:30 am.
Check If Exist Callable API
No
Yes,Call this API
<button> clock open <button>
<button> new alarm <button>
<button> rolling search <button>
<button> alarm ready <button>
Click
Click
Click
API command
API Start!

```
"Clock": [
    "API Command": "adb shell am start -a android.intent.action.SET_ALARM --ei
android.intent.extra.alarm.HOUR <x> --ei android.intent.extra.alarm.MINUTES <y>"
```

Figure 1: For the task of "*Setting an alarm for seven thirty.*", accomplishing it solely through UI operations requires four steps, while API calls can achieve the same task in just one step.

1 Introduction

Interacting with mobile devices using natural language is a long-standing pursuit in human-computer interaction (Bolt, 1980; Karat et al., 2002; Følstad and Brandtzæg, 2017). With the remarkable advancements in large language models (LLM) (Bai et al., 2022; Chowdhery et al., 2022; Du et al., 2021; Touvron et al., 2023; Ouyang et al., 2022), LLM-driven agents are at the forefront, yet their reasoning capability to navigate mobile application functionalities lags behind their proficiency with web pages on PCs (Yao et al., 2022; Sun et al., 2023). To faithfully replicate a typical mobile environment, it's imperative to incorporate a diverse set of applications and leverage authentic data, moving beyond the limitations of purely simulated scenarios. The development challenges in the mobile domain stem from a trio of core issues: a limited understanding of mobile interfaces, a scarcity of application variety, and a lack of real-world data.

Due to Google's breakthrough (Wang et al., 2023) in UI interface representation, LLM agent's understanding of UI pages becomes easier, leading to the creation of UI platforms such as Android-Env (Toyama et al., 2021) and Mobile-Env (Zhang et al., 2023), which tasks are defined within individual games or search engines. However, these works collectively face the following challenges: (1) UI actions depend on the textual descriptions of interfaces, where structured text fails to capture

[*] Equal contribution.
[†] Corresponding authors: Rui Yan and Shuo Shang.
[‡] Work done during the internship at XiaoMi.

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 8813–8831
August 11-16, 2024 ©2024 Association for Computational Linguistics

Platform&BenchMark	**InfoUI**	**API&UI**	**Real APP**	**Real Query**	**Multi-APP**
World of Bits (Shi et al., 2017)	✓	✗	✗	✗	✗
WebShop (Yao et al., 2022)	✓	✗	✗	✗	✗
AndroidEnv (Toyama et al., 2021)	✗	✗	✓	✗	✗
MobileEnv (Zhang et al., 2023)	✓	✗	✓	✗	✗
Mobile-Bench (Ours)	✓	✓	✓	✓	✓

Table 1: Comparison of Mobile-Bench with existing LLM-based agent platforms. 'InfoUI' represents whether UI information is used for interaction with agents, 'API&UI' represents whether the agent's actions like API calls and UI interface operations, 'Real APP' represents whether real applications are used, 'Real Query' represents whether real user queries are used, and 'Multi-APP' represents whether there are tasks involving multiple applications.

the content of graphical buttons or images which can lead to wrong actions. A single API action might be equivalent to dozens of UI steps, leading to UI's inefficiency. (2) Their tasks are far removed from real-world task scenarios encountered in daily use, which require cooperation between multiple applications, with user commands being ambiguous and not specifying target applications. (3) The evaluation of tasks should not solely rely on LLMs, without any objective quantitative metrics.

In fact, voice assistants on mobile phones can meet most of the users' daily needs, yet they do not interact directly with UI interfaces but operate by invoking the APIs (Qin et al., 2023) behind applications. As shown in Figure 1, in mobile applications, APIs are more efficient than UI interfaces; a single API call can be equivalent to multiple UI operations to achieve the same outcome. However, a single API is insufficient for more complex tasks, especially when user commands are unclear, necessitating reliance on LLMs to interpret user intent. Therefore, an agent capable of utilizing both UI and APIs would be best suited for the job. Simultaneously, It requires developing a strategy for the selection and order of the application usage, with human oversight merely focusing on reviewing the outcomes. This is a function that voice assistants currently lack (Wen et al., 2023a,b). To this end, we develop a combination of API and UI actions to circumvent the limitations of UI interfaces, each action can be chosen between UI interactions and API calls; all tasks begin from the mobile HOME page rather than from the launch page of a specific application, enabling the agent to determine single or multiple applications it will use; queries in the task are gathered from real users, and instruction generation is only applied to some complex ones which undergo rigorous manual review; we draw inspiration from objective metrics in software automation testing, named CheckPoint, and have made necessary adjustments to accommodate the unpredictable semantic outputs of LLMs. Above all, we propose a mobile phone environment that includes a platform supporting both API and UI interactions, and a corresponding dataset with multi-APP tasks. Table 1 presents a comparison among recent platforms and benchmark work based on API and UI.

Our contributions are summarized as follows:

(1) To the best of our knowledge, we are the first to establish a running platform for LLM-based mobile agents that simultaneously supports both UI and API calls.

(2) We propose an evaluation dataset containing diverse tasks for multi-APP interactions. Our tasks starting from the home page are more appropriate for testing the planning capabilities for agents. Our dataset and platform will be released soon.

(3) We introduce a new category-based evaluation metric to assess the task completion capabilities of the agent in the context of both UI and API interactions.

2 Related Work

2.1 Mobile Platforms

Prior to the emphasis on LLM-based agents, research efforts were directed towards RL-based agents, exemplified by the Android-Env platform (Toyama et al., 2021). This open-source platform tailored for reinforcement learning experiments within the Android ecosystem, successfully tested various RL-based agents like DDPG (Zhang and Van Huynh, 2023), D4PG (Barth-Maron et al., 2018), MPO (Abdolmaleki et al., 2018), DQN (Mnih et al., 2015), IMPALA (Espeholt et al., 2018) and R2D2 (Kapturowski et al., 2018).

More significant research has focused on LLM-based agents (Liu et al., 2024; Sun et al., 2024b,a). Regarding the domain of tool-using agents, they can be categorized into three main types:

1) For mobile tasks. Platforms like AutoDroid, DroidBot-GPT, GPT-Droid, and WebShop (Wen et al., 2023a,b; Liu et al., 2023b; Yao et al., 2022) create an interactive environment enabling LLMs to engage with mobile tasks, and generate human-like operations for automation test. Mobile-Env (Zhang et al., 2023) is specifically designed to evaluate agents' capabilities in handling multi-step interactions.

2) For PC Tasks. Researchers developed Toollama (Qin et al., 2023) to evaluate the capabilities to use tools and API calls. AgentBench (Liu et al., 2023a) presents a standardized Agent task evaluation architecture with strong decoupling and scalability. PPTC Benchmark (Guo et al., 2023) proposed to evaluate the ability of LLM-based agents on PowerPoint tasks.

3) Other Methods. Toolformer (Schick et al., 2023) and HuggingGPT (Shen et al., 2023) evaluate LLM's capability to master tools.

2.2 Benchmarks for LLM agents

To assess agents' proficiency in understanding user interfaces, a diverse dataset covering various tasks is crucial (Liu et al., 2023a). The widely used RICO dataset (Deka et al., 2017) is commonly employed for this purpose, with Screen2Vec (Li et al., 2021) utilizing it to evaluate agent performance.

However, due to the absence of specific standards for evaluating agent performance, efforts have focused on designing evaluation frameworks. PPTC Benchmark (Guo et al., 2023) devised 279 multi-round dialogue tasks for PPT file operations. DroidTask (Wen et al., 2023a) and various unnamed datasets (Liu et al., 2023b; Wen et al., 2023b) covering various mobile applications have also been established. Additionally, Screen2Words used a sampling method to sample screens from the RICO-SCA (Li et al., 2020) dataset and hired professional annotators to generate English summaries for these screens (Wang et al., 2021).

Current evaluation standards align with various works. ToolBench proposes Win Rate gauges the model's solution quality against benchmarks like RoBERTa (Liu et al., 2019), GPT-3 (Brown et al., 2020), PaLM (Chowdhery et al., 2023), OPT (Zhang et al., 2022), ChatGPT (Bubeck et al., 2023) and GPT-4 (OpenAI, 2023). Although Fan (Fan et al., 2024) found that the cost of inference can be reduced by using only the necessary layers for inference, it is still expensive to calculate the win rate. Mobile-Env (Zhang et al., 2023) evaluates agent performance based on the completion status, average steps, and average rewards in WikiHow tasks. PPTC Benchmark (Guo et al., 2023) uses Turn-based and Session-based accuracy. Android in the Wild (Rawles et al., 2023) makes use of Out-of-distribution Generalization. Overall, metrics such as success rate, episode length, and match score are currently the most commonly employed.

3 Our Environment

3.1 Mobile-Bench Benchmark

Data collection. The queries in the dataset are divided into the following three categories:

- ***SAST: Single-App-Single-Task.*** A real dataset containing only one task text, including single-task operations such as opening and closing APP, such as *"Help me open the map"*.

- ***SAMT: Single-App-Multi-Task.*** A real dataset containing multiple task texts, as well as constructed single-APP data. A complex multi-task on single APP, such as *"Help me open the map, and navigate to Eiffel Tower."*.

- ***MAMT: Multi-App-Multi-Task.*** Constructed multi-APP data, complete a complex multi-task, such as *"Help me search for the latest technology news and share it with friends."*

SAST is directly derived from real voice requests processed by the voice assistants loaded on the mobile phone. We select a subset of this query collection, primarily filtering out the portion that requires voice assistant processing and involves multimodal tools. Additionally, querys that exceed permissions or involve privacy are also filtered out.

Since there are fewer SAMT and MAMT data in real data and the quality is not high, refer to Toollama (Qin et al., 2023) method, we use GPT-4 to construct SAMT and MAMT data. For MAMT, we randomly sample 6 applications from the entire application collection, and then provide some examples of real multi-APP data to prompt GPT-4 to select 2-4 applications to generate tasks. By integrating real and constructed data, we create the final dataset. An example of data is shown in Figure 2.

ID: 3

Query:

I want to book a flight from Beijing to Shanghai next Friday. Are there any recommended flights?

APP:

Amap & (Ctrip Travel | Qunar)

CheckPoints:

- **Package:**
  com.autonavi.minimap & (ctrip.android.view | com.Qunar)
- **Key phrase:**
  [flight, Beijing, Shanghai, next Friday]
- **API**:
  - adb shell am start -n com.autonavi.minimap/com.autonavi.-map.activity.SplashActivity
  - adb shell am start -a androidintent.action.VIEW -damapuri://route/plan/?dname=Shanghai com.autonavi.mini-map

Figure 2: A test case in MAMT. & stands for conjunction check, CC; | stands for disjunction check, DC; [ ] stands for sequential check, SC. The package CheckPoint passes when the action history includes either Amap and Ctrip Travel, or Amap and Qunar. Key phrase CheckPoint comes from the orange parts in the case.

APP & API collection. To ensure task comprehensiveness, we select not only the applications included in SAST and SAMT but also the most popular free applications from each category in the APP Store. Obtaining the API is to analyze the package of each application to obtain its external reserved interface (Desnos and Gueguen, 2011). The advantage of this is that the obtained API is naturally classified for the application. Since the description of the API in the decompilation result is not as detailed as the development document, we use the ADB(Android Debug Bridge) command to verify the feasibility of the API one by one. Owing to its debugging properties, system-level APIs can also be invoked normally, allowing access to functions such as checking the battery status and performing memory cleaning. For more specific application names and categories, please refer to Appendix B.3

Dataset statistics. Including several default applications within the system, we collected a total of 29 applications. For applications, we collected a total of 103 usable APIs, which primarily serve the following functions: system calls, opening pages, closing pages, searching for information, viewing details, and controlling device switches. These functions are summarized into the following main aspects: page switch, details view, broadcast, search. In Table 2, we have tabulated the number of APIs and the functional categories covered by APIs, categorized by the type of APP. We organized the available APIs and APP descriptions for each APP, and generated an APP list as the basis for selecting applications, shown in Appendix B.3.

APP Category	API Quantity	APP Number	API Functions
Travel Transportation	5	3	①, ②, ④
Audiovisual Vision	15	5	①, ②, ③
Social Communication	3	1	①, ②, ④
Fashion Shopping	14	6	①, ④
Information News	11	4	①, ②, ④
Practical Tool	38	8	①, ②, ③, ④, ⑤
Home Life	5	1	①, ⑤
Book Reading	7	2	①, ②, ④
Universal Buttons	5	0	⑤

Table 2: Our dataset covers nine major categories of applications, and we compared them based on the API function. The above API functions can be summarized into five categories: ① Page Navigation, ② Viewing Details, ③ Playback, ④ Searching, and ⑤ System Calls.

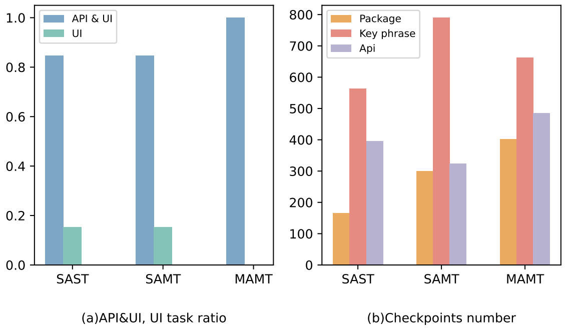


Figure 3: (a) The API&UI, UI task ratio. In SAST and SAMT, API&UI task ratio is 85%, in MAMT, it is 100%. (b) The number of CheckPoints.

In the Mobile-Bench dataset, we collected a total of *332, 300, 200* queries for SAST, SAMT, and MAMT. We sort out the APIs actually used by each task in real voice requests. Provide these API as an example to GPT-4 for query generation. As shown in Figure 3(a), we calculated the ratio of tasks calling APIs, ensuring a sufficient number of tasks in the dataset that include steps to call API. This approach ensures that we have sufficient data to analyze the role of APIs in task completion.

Quality verification. (Bolotova-Baranova et al., 2023) The initial test data originates from software automation tests, but some complex data points are generated by GPT-4. To ensure the quality of our dataset, we randomly sampled 100 data points from each of the SAST, SAMT, and MAMT, resulting in a total of 300 quality test data. We conducted cross-source validation to verify the feasibility of these CheckPoints. The specific formula for calculation is as follows:

$$\text{Overlap}(CP_1, CP_2) = \frac{|CP_1 \cap CP_2|}{|CP_1|} \qquad (1)$$

CP_1,CP_2 representing the CheckPoint sequences generated by $CP_{instruction}$ and CP_{Human}, respectively. In Table 3, we list the human evaluation results for three types of data. From the table, it can be observed that a higher proportion of terminal data corresponds to better data quality. However, all MAMT data is generated by instructions, its quality does not exhibit an unacceptable gap compared to SAST. See appendix B.1 for more analysis.

Statistics	**SAST**	**SAMT**	**MAMT**	**Total**
CP$_{\text{instruction}}$	395	546	513	1454
CP$_{\text{Human}}$	412	598	623	1633
CP$_{\text{instruction} \cap \text{Human}}$	372	466	412	1250
Overlap	0.94	0.85	0.80	0.86

Table 3: Human Evaluation Results

3.2 Test Platform

Overview Mobile-Bench is designed as a universal interaction platform that supports hybrid API and UI interactions. Users are able to construct their own evaluation data following a fixed format, yet they must adhere to our prescribed evaluation method. As shown in Figure 4 users can interact with the environment using the following commands.

- ***Start***: Open the *test environment* and load the preset snapshot using this command. Each test case must start from the same environment.
- ***Stop***: Stop the *test environment* and end test.
- ***Close***: Close the *test environment* and save the test process and results.
- ***Check***: Capture a screenshot snapshot of the current *test environment*.
- ***ReSet***: Load a previously saved environment snapshot into the *test environment*.

Observation space To enable the agent to read information on the android emulator in a human-like manner, we use Appium to obtain page information. Following the method described by Wang (Wang et al., 2023), we convert XML to HTML, as the training data for LLMs is predominantly sourced from the Internet, which includes numerous HTML files. Therefore, we believe that LLM has a better understanding of HTML than XML.

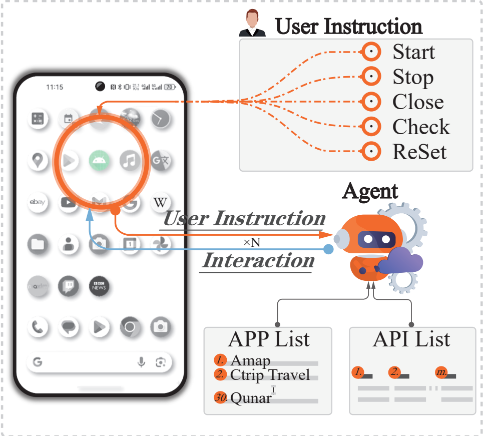


Figure 4: Test Platform Overview. The test platform is linked by the user, the simulator, and the Agent. After the user's instructions are issued, the entire test execution process is completed by the Agent, which can view and manage the test tasks through the preset interface in the cloud.

Given the tree structure of XML, we initially convert the XML into a tree format and subsequently transform the nodes that need to be displayed to the agent into HTML. The agent simulates human interaction with smartphones, performing three major operations: click, input, and scroll. Humans visually identify which elements can be clicked or receive input, and use their fingers to determine if they can scroll the screen. Therefore, we provide the agent with elements that are visible and scrollable. Due to the limit on context length, we only convert the information required by the agent in XML to HTML:

- ***Type***: HTML element categories inherited directly from XML formatted information.
- ***ID***: "*ID*" inherits from the XML "*resource-id*" attribute, uniquely identifying the existence of an element.
- ***Package***: the package name of the current application.
- ***Class***: the class of the element, such as *ImageView, TextView*.
- ***Description & text***: describe the function and shape of the element.
- ***Clickable & Scrollable***: whether the element is clickable and scrollable.
- ***Bounds***: if the element is scrollable, this attribute will be present and scope the scroll component, such as:

$$[\boldsymbol{x_i}, \boldsymbol{y_i}]\,[\boldsymbol{x_j}, \boldsymbol{y_j}]$$

The scrollable rectangle ranges from $[x_i, y_i]$

to $[x_j, y_j]$.

And, there is an example of HTML elements: *<button package="com.ximalaya.ting.android" class="android.widget.Button" clickable="true"> message </button>*

Action space Our Mobile-Bench imitates human behavior in using mobile and summarizes three actions (Zhang et al., 2023) and imitates the process of calling the API on the test platform (Sengupta et al., 2023):

- ***Click***: simulate real user click actions by passing in specific elements.
- ***Scroll***: simulate real user scrolling actions by *tapping - dragging - releasing*.
- ***Input***: simulate real user input actions by *clicking-typing*.
- ***API Call***: launch an activity or send an intent by invoking an API through ADB commands.

3.3 Evaluation Method

CheckPoint. Automated test CheckPoint coverage (Bajunaid and Menascé, 2018) is a test metric for the software execution process. It cannot assist in checking the software results, but it can visually inspect whether the software runs in the specified unit sequence. During data construction, we supply APPs and APIs, which naturally serve as detection indicators. Additionally, we incorporated a CheckPoint to verify if the UI operation correctly clicks on the intended element. After sorting out the above CheckPoints, we constructed the following three CheckPoints:

- ***Package***: the unique package name corresponding to the application. Checking the package can determine whether the correct application is used.
- ***Key phrase***: the key phrase extracted from the query, represents key steps in the UI execution process.
- ***API***: API commands that need to be called during the execution process.

To evaluate the agent's selection and execution capabilities, we divide the inspection granularity into two levels: *CheckPoint$_{l1}$* - whether it uses the correct application, and *CheckPoint$_{l2}$* - whether it follows the predefined paths to complete the task. For *CheckPoint$_{l1}$*, we check the number of correctly called packages. For *CheckPoint$_{l2}$*, we check the number of correctly called package, key phrase, API. For CheckPoints, we identify three logical relationships: sequential, conjunctive, and disjunctive checks. These correspond to the instability of LLM output and its tendency for synonym substitution. The calculation formula for "sequential check" is as follows:

$$Score_{Sequen} = \frac{|\sum_{Str \in SC \cap AH} Str|}{|\sum_{Str \in SC} Str|} \tag{2}$$

SC represent *Sequential Check Set* and *AH* represent *Actions History*. The calculation formulas for conjunctive checks is as follows:

$$Score_{conjun} = \begin{cases} 1, & if\ \forall str \in CC, str \in AH \\ 0, & otherwise \end{cases} \tag{3}$$

CC represent *Conjunctive Check Set*. The calculation formulas for disjunctive checks is as follows:

$$Score_{disjun} = \begin{cases} 1, & if\ \exists str \in DC, str \in AH \\ 0, & otherwise \end{cases} \tag{4}$$

DC represent *Disjunctive Check Set*. The weighted sum of the above three scores will be the final CheckPoint coverage rate.

As shown in Figure 3, the number of key phrase CheckPoints is significantly higher than that of packages, indicating the need for more semantic information to ensure tasks are completed step-by-step. Analyzing the dataset from a proportional perspective, we find that the distributions of the three types of CheckPoints are *0.212, 0.493, 0.294*, with key phrase CheckPoints remaining the most predominant method of checking.

In general, a *test case* should include at least the following contents: ***ID, Query, APP List, CheckPoints(Package, Key phrase, API)***. Figure 2 is a test case that contains the above three CheckPoints.

PassRate. (Qin et al., 2023) We assess an agent's human-computer interaction capabilities by calculating the proportion of queries successfully completed within the specified step limits. During this process, we organized the emulator's current state. Subsequently, GPT-4 evaluates the task completion status. We computed the percentage of pass tasks, yielding a PassRate as an indicator of agent's human-computer interaction capabilities.

Average steps. (Zhang et al., 2023) We quantified the step size required by Mobile-Bench to complete tasks as a metric for evaluating the efficiency of the agent. In Mobile-Bench, a 'step' is

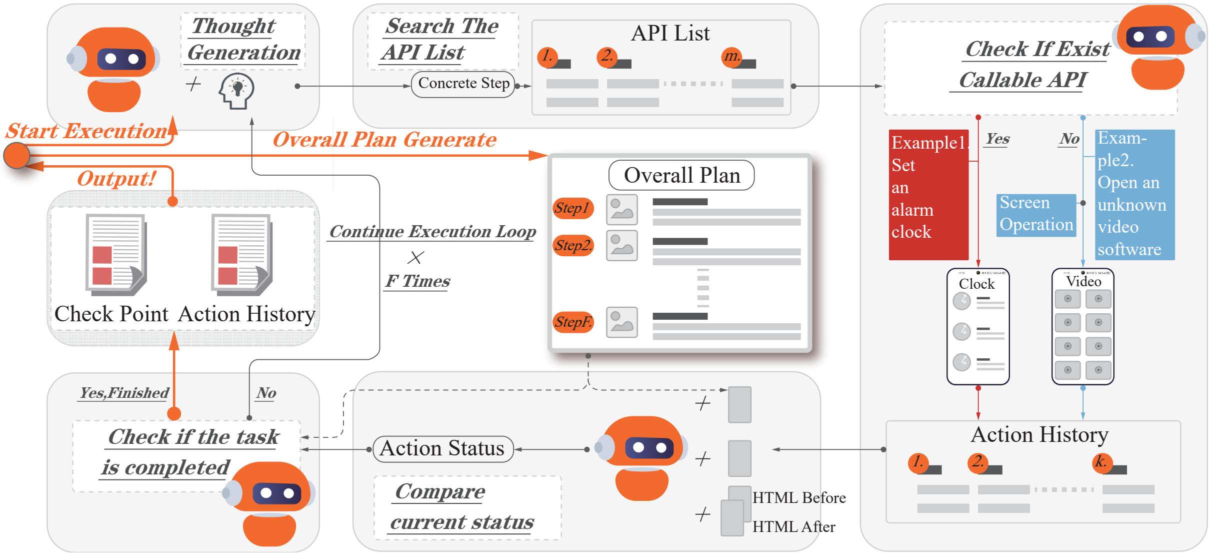


Figure 5: Baseline Model Overview. The entire process framework consists of sensors, reflection components, controllers, execution components, and environments. Once a task starts, these components will run iteratively until the task is completed or the maximum number of steps is reached.

defined as the completion of a UI operation or the execution of an API call.

4 Experiment

4.1 Baseline Model

Our model's architecture, illustrated in Algorithm 1, begins by obtaining the smartphone's UI information in XML format through Appium and transforms it into HTML format through a heuristic algorithm. Subsequently, as illustrated in Figure 5 leveraging the HTML, task details, and APP list, LLM generates a comprehensive task plan, outlining the necessary applications and corresponding sub-tasks. As the collection of APIs is organized based on the classification of APPs, we can get the API set that may be used in plan.

The task plan is executed iteratively. In each iteration, the model either performs an API call or a UI operation. After each execution, the model records the success or failure of the action in its history, generates the subsequent thought, and evaluates whether the task has been completed. For the actual running process of an algorithm, please refer to the appendix C.7.

4.2 Setup

We evaluate four popular LLMs on the proposed Mobile-Bench task set: GPT-3.5-turbo (Ouyang et al., 2022), GPT-4 (Nori et al., 2023), LLaMA-13B and LLaMA-70B(Touvron et al., 2023), while ChatGPT-3.5 and GPT-4 are accessed through the online APIs of OpenAI. The experiments are conducted with a 3-shot in-context learning under sampling temperature of 0.1. Recognizing that task execution incurs costs, we preset different maximum step limits for tasks based on their difficulty levels. For the three categories of SAST, SAMT, and MAMT, we set the max step to 10, 20, and 50 respectively. Owing to the limit of budget, only GPT-3.5 utilizes an interface with a context length of 16K. GPT-4 uses a standard interface, which necessitated compression and trimming of actions

Algorithm 1 Baseline Model

```
Input: description of the Task, $Task$; APP list, $L_{APP}$; API
    list, $L_{API}$; max loop step, $M_{\text{step}}$; initial thought, $Tho$;
Output: actions history, $AH$; total steps, $Step$; finish flag,
    $Finish$;
$Html \leftarrow Appium(Emulator)$
$Plan \leftarrow LLM(Task, L_{APP})$
$Step = 0,\ Finish = False$
$AH = [\,]$
while $(Step \leq M_{step}) and (Finish \neq True)$ do
    $Step++$;
    $Html \leftarrow Appium(Emulator)$;
    $API \leftarrow LLM(Task, L_{API}, AH, Tho, Plan, Html)$
    if $API$ then
        $Action(API)$
        $AH.append(API)$
    else
        $UI \leftarrow LLM(Task, AH, Tho, Plan, Html)$
        $Action(UI)$
        $AH.append(UI)$
    end if
    $Html \leftarrow Appium(Emulator)$;
    $Tho \leftarrow LLM(Task, AH, Plan, Html)$
    $Finish \leftarrow LLM(Task, AH, Tho, Html)$
end while
```

Metric	LLaMA-13B			LLaMA-70B			GPT-3.5-turbo			GPT-4		
	SAST	SAMT	MAMT	SAST	SAMT	MAMT	SAST	SAMT	MAMT	SAST	SAMT	MAMT
Average #Steps	7.43	18.76	49.52	5.97	16.63	48.91	4.53	**12.06**	48.73	**3.79**	13.94	**44.86**
PassRate	44.58	27.67	8	56.62	54	13.5	64.94	**64**	15.5	**80.96**	63	**26.5**
CheckPoint$_{l1}$	46.08	43.67	28.74	56.62	61	39.98	66.75	67	43.16	**81.57**	**72.66**	**61.34**
CheckPoint$_{l2}$	34.85	29.13	21.39	63.12	62.73	41.21	76.21	71.29	44.09	**83.76**	**77.35**	**52.98**

Table 4: Results of the agents based on different LLMs on Mobile-Bench dataset. On MAMT data, due to context length limitations, a compression is applied to the actions history by retaining only the most recent 20 entries.

history. See Appendix A for other settings.

4.3 Results

As observed in Table 4, it can be observed that GPT-3.5 outperforms GPT-4 in PassRate on SAMT(64%>63%), and it requires fewer steps to complete the task(12.06<13.94). To investigate this phenomenon, we analyze the output files and find that models with poorer performance exhibit PassRate misjudgments: they prematurely terminate even when the task is not completed. This phenomenon is also present in LLaMA, which exhibits a high PassRate (44.58%) but low CheckPoint coverage (34.85%). At the same time, we delved into why the results for MAMT are so low(15.5%, 26.5%). Our analysis revealed that LLMs often exhibit greedy exploration behavior when completing tasks, meaning they struggle to determine when to exit the current application and transition to the next one. This tendency is particularly prevalent in certain generation tasks. Moreover, as the actions history increases, its ability to accurately judge task progress becomes increasingly challenging. For more detailed result, please refer to Table 7.

Settings	**Average #Steps**	**CheckPoint$_{l2}$**	**PassRate**
SAST (GPT-4)	**3.79**	**83.76**	**80.96**
SAMT (GPT-4)	13.94	77.35	63
MAMT (GPT-4)	44.86	52.98	26.5
SAST (w/o API)	6.13	72.73	74.39
SAMT (w/o API)	16.86	56.74	48
MAMT (w/o API)	49.17	31.69	9.5

Table 5: API Ablation Study based on GPT-4.

4.4 Impact of API Calls

API Calls can accelerate task execution, as a single call often replaces several sequential UI steps. From another perspective, the ability of the agent to select appropriate APIs and input parameters warrants further investigation. Choosing the wrong API may lead the task in an incorrect direction or require a significant number of steps to rectify. Therefore, in Table 5, we evaluate and analyze the impact of introducing APIs on task completion based on GPT-4.

From Table 5, it can be seen that even in SAST, the PassRate has decreased by 6.57% (from 80.96 to 74.39). Furthermore, the values for CheckPoints$_{l2}$ exhibit a more pronounced decrease after API removal, with a drop exceeding 20% in SAMT. Simultaneously, we have observed varying increases in the average number of steps, which align with our expectations. We analyzed the results and found that the inability to accurately scroll pages, inefficient exploration of page functionality, and failure to click graphical buttons are the primary reasons for the low efficiency of UI operations.

Settings	**Average #Steps**	**CheckPoint$_{l1}$**	**CheckPoint$_{l2}$**	**PassRate**
SAST (GPT-4)	**3.63**	**82**	**79.74**	**76**
SAST (w/o thought)	8.86	**82**	29.16	24
SAST (w/o plan)	3.98	76	74.54	72
SAMT (GPT-4)	**13.94**	**63**	**72.66**	**77**
SAMT (w/o thought)	19.54	**63**	18.31	20
SAMT (w/o plan)	17.09	52	58.02	62

Table 6: Thought and Plan Ablation Study on SAST (subset 50) and SAMT (subset 200) based on GPT-4.

4.5 Impact of Plan and Thought

Since observation-thought-action is already a standardized process in the agent direction(Qin et al., 2023), and verified by experimental results, planning and thought before action are essential. From the experimental results, we can find that without the observation-thought step, the agent is almost unable to complete the task(77->20, 76->24), which is because it cannot determine the next action category and the current task status. In more complex tasks SAMT, losing the plan has more negative consequences(77->62). But they will have almost no impact on *CheckPoint$_{l1}$*(82->82 63->63), because the application selection is almost done by the API Call.

5 Conclusion

In this work we have proposed an agent capability testing environment that supports API and UI interaction on mobile phone. This holds significant importance for exploring how LLMs can be integrated with mobile operating systems. Additionally, it can serve as a valuable reference for developing testing platforms for operating systems to evaluate the capabilities of LLM agents. We collected and released a test dataset containing tasks for multiple APPs, ensuring its quality through human verification. Based on this data set and environment, we tested the planning, decision-making and execution of various LLM-based agents. Please refer to the Section 6 for the limitations of our benchmark.

6 Limitations

While general large models exhibit strong capabilities in reasoning and planning, they tend to have pronounced illusions in API calls. As a result, the language model may become confused about the application's functionality, leading to a reluctance to continue and complete the task. Therefore, fine-tuning a model for instructions is highly necessary.

Automatic CheckPoint is a process evaluation metric, making it challenging to assess the quality of the final outcome. This depends on whether the agent has obtained the necessary information (actions) on the required pages.

The enhancement of the agent's capabilities relies on extensive API and SDK libraries, requiring substantial support from application development companies.

7 Ethics Statement

We have rigorously refined our dataset to remove any elements that could compromise personal privacy, thereby guaranteeing the highest level of protection for individual data. The evaluation of our work was carried out through a meticulously randomized selection of IT professionals. This process ensured a gender-balanced and educationally diverse panel, reflecting a wide spectrum of perspectives and expertise.

8 Acknowledgements

We thank the Xiaoai Voice Department of Xiaomi Technology Corporation for their raw data support for this project. We additionally thank our crowd annotators for their diligent work, Junfeng Peng and Yifan Cheng for contributing to the human performance estimates, and the anonymous reviewers for their constructive comments. This work was supported by the NSFC (U2001212, 62032001, and 61932004).

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. 2018. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*.

Y Bai, S Kadavath, S Kundu, A Askell, J Kernion, A Jones, A Chen, A Goldie, A Mirhoseini, C McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback (arxiv: 2212.08073). arxiv.

Noor Bajunaid and Daniel A Menascé. 2018. Efficient modeling and optimizing of checkpointing in concurrent component-based software systems. *Journal of Systems and Software*, 139:1–13.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*.

Valeriia Bolotova-Baranova, Vladislav Blinov, Sofya Filippova, Falk Scholer, and Mark Sanderson. 2023. Wikihowqa: A comprehensive benchmark for multi-document non-factoid question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5291–5314.

Richard A Bolt. 1980. "put-that-there" voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 262–270.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*, pages 845–854.

Anthony Desnos and Geoffroy Gueguen. 2011. Android: From reversing to decompilation. *Proc. of Black Hat Abu Dhabi*, 1:1–24.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2021. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360*.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*.

Asbjørn Følstad and Petter Bae Brandtzæg. 2017. Chatbots and the new world of hci. *interactions*, 24(4):38–42.

Yiduo Guo, Zekai Zhang, Yaobo Liang, Dongyan Zhao, and Duan Nan. 2023. Pptc benchmark: Evaluating large language models for powerpoint task completion. *arXiv preprint arXiv:2311.01767*.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. 2018. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*.

Clare-Marie Karat, John Vergo, and David Nahamoo. 2002. Conversational interface technologies. *The human-computer interaction handbook*, pages 169–186.

Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, and Brad A Myers. 2021. Screen2vec: Semantic embedding of gui screens and gui components. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023a. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Yuhan Liu, Xiuying Chen, Xiaoqing Zhang, Xing Gao, Ji Zhang, and Rui Yan. 2024. From skepticism to acceptance: Simulating the attitude dynamics toward fake news. *arXiv preprint arXiv:2403.09498*.

Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2023b. Chatting with gpt-3 for zero-shot human-like mobile automated gui testing. *arXiv preprint arXiv:2305.09434*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Harsha Nori, Nicholas King, Scott Mayer McKinney, Dean Carignan, and Eric Horvitz. 2023. Capabilities of gpt-4 on medical challenge problems. *arXiv preprint arXiv:2303.13375*.

OpenAI. 2023. Gpt-4 technical report.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Aritro Sengupta, Amit Singh, and BM Vinjit. 2023. A platform independent and forensically sound method to extract whatsapp data from mobile phones. *International Journal of Electronic Security and Digital Forensics*, 15(3):259–280.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR.

Hongda Sun, Hongzhan Lin, Haiyu Yan, Chen Zhu, Yang Song, Xin Gao, Shuo Shang, and Rui Yan. 2024a. Facilitating multi-role and multi-behavior collaboration of large language models for online job seeking and recruiting. *arXiv preprint arXiv:2405.18113*.

Hongda Sun, Yuxuan Liu, Chengwei Wu, Haiyu Yan, Cheng Tai, Xin Gao, Shuo Shang, and Rui Yan. 2024b. Harnessing multi-role capabilities of large language models for open-domain question answering. In *Proceedings of the ACM on Web Conference 2024*, pages 4372–4382.

Hongda Sun, Weikai Xu, Wei Liu, Jian Luan, Bin Wang, Shuo Shang, Ji-Rong Wen, and Rui Yan. 2023. Determlr: Augmenting llm-based logical reasoning from indeterminacy to determinacy. *arXiv preprint arXiv:2310.18659*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Bryan Wang, Gang Li, and Yang Li. 2023. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–17.

Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2words: Automatic mobile ui summarization with multimodal learning. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 498–510.

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

Hao Wen, Hongming Wang, Jiaxuan Liu, and Yuanchun Li. 2023b. Droidbot-gpt: Gpt-powered ui automation for android. *arXiv preprint arXiv:2304.07061*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Bolun Zhang and Nguyen Van Huynh. 2023. Deep deterministic policy gradient for end-to-end communication systems without prior channel knowledge. *arXiv preprint arXiv:2305.07448*.

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

A Settings

We conduct experiments on the Android 14.0 version emulator and use Appium UiAutomator2 Driver for automated testing. Before each execution of a task, we load a snapshot to ensure the emulator in the same environment every time. For all applications, we have logged in to the account in advance to ensure that the full function of the application can be used. Since we tests in the real world, we filtered out any tasks that included payments.

B Details of Dataset

B.1 Dataset quality analysis

The root cause of low-quality data often lies in the inaccuracies in the descriptions of applications. Additionally, ambiguity in query generation also plays a significant role. For example, in the query *”Help me find pictures related to Beijing”*, although the user has not explicitly specified the source application, for a human, the expected result would likely be a search engine or a map application, as the images are not likely to be from the user themselves. However, for LLM, because the statement includes the word “pictures”, it might be reasonable for it to spend all its time searching for pictures in the gallery application, even though this effort would ultimately be in vain. CheckPoint coverage is calculated as the weighted sum of the scores for the three types of CheckPoints mentioned above.

B.2 Prompts for Instruction Generation

Below we list the detailed prompt for instruction generation, including single-APP-multi-task description, multi-APP-multi-task description.

single-APP-multi-task description:

You will be provided with an application with descriptions, an available API list including adb command, function description and parameter information. You should create 5 varied, innovative, and detailed multi task queries that employ this application as a tool, API can be used as an auxiliary.

Each query should include the necessary parameters. Note that you shouldn't ask ‘which APP to use’, rather, simply state your needs that can be addressed by these APPs. You should also avoid asking for the input parameters required by the APP call, but instead directly provide the parameter in your query. Those related APP and APIs have to strictly come from the provided lists.

At the same time, you also need to provide the CheckPoint of this query, including package, key phrase and API. The package comes from the package corresponding to the APP to be used. Key phrase is the key click element or key input character that the Android emulator will perform when executing this query, which is used to check whether the query has been completed. Key phrase should be noun and part of query, should be kept as short as possible.

Key phrase can contain multiple pieces of information, "|" means the query passes when any of the following texts are completed. "|" is used to separate synonymous expressions of the same noun; "&" indicates that the query must be passed when all texts are completed; sequential CheckPoints are stored in "[ ]", and the count increases by one for each passed element. The "ADB Command" to be used is stored in the API, which may also be empty.

Deliver your response in this format:

```
[{
   "id": "number"
   "query": "text"
   "APP": "APP name"
   "CheckPoint": {
      "package": "APP package name"
      "key phrase": ["text1", ... ]
      "API: ["API1", ... ]"
      }
}
```

...
]

multi-APP-multi-task description:

You will be provided with some APPs with descriptions, available API list including adb command, function description and parameter information. You should create 3 varied, innovative, and detailed multi queries that employ multi-APP as a tool, API can be used as an auxiliary.

Each query should include the necessary parameters. Note that you shouldn't ask 'which APP to use', rather, simply state your needs that can be addressed by these APPs. You should also avoid asking for the input parameters required by the APP call, but instead directly provide the parameter in your query. Those related APPs and APIs have to strictly come from the provided lists. You should first think about possible related APP combinations, then give your query. Keep in mind that each query should call upon two to four APPs.

At the same time, you also need to provide the CheckPoint of this query, including package, key phrase and API. The package comes from the package corresponding to the APP to be used. Key phrase is the key click element or key input character that the Android emulator will perform when executing this query, which is used to check whether the query has been completed. Key phrase should be noun and part of query, should be kept as short as possible.

Key phrase can contain multiple pieces of information, "|" means the query passes when any of the following texts are completed. "|" is used to separate synonymous expressions of the same noun; "&" indicates that the query must be passed when all texts are completed; sequential CheckPoints are stored in "[ ]", and the count increases by one for each passed element. The "ADB Command" to be used is stored in the API, which may also be empty. For different queries, overlap of related APPs should be as little as possible.

Deliver your response in this format:

```
[{
   "id": "number"
   "query": "text"
   "APP": ["APP name1", ...  ]
   "CheckPoint": {
      "package": ["APP package name1", ...  ]
      "key phrase": ["text1", ... ]
      "API: ["API1", ... ]"
      }
}
...
]
```

B.3 APP&API statistics

As can be seen from Figure 6, each functional area contains at least one application and its corresponding API. These applications are sufficient to meet the daily needs of users. In other words, our simulation environment is almost consistent with the real daily use environment, and it is consistent with the real daily use environment. Open world information exchange. There are so many practical tools that are the basic functions of mobile . They have been automatically installed and completed during system installation, and standard API interfaces for tools are easier to obtain. Our next step is to increase the number of APIs and SDKs for third-party applications.

B.4 Case study

CheckPoints is a group of words, including packages, key phases, and API, which represent the package name, action keywords, and API instructions of the application respectively. We regularize these words and action histories to check whether they select a sufficient and correct number of applications, UI elements, and APIs to accomplish the given task.

Next, we will give an example of CheckPoints in Figure 7 and Figure 8.

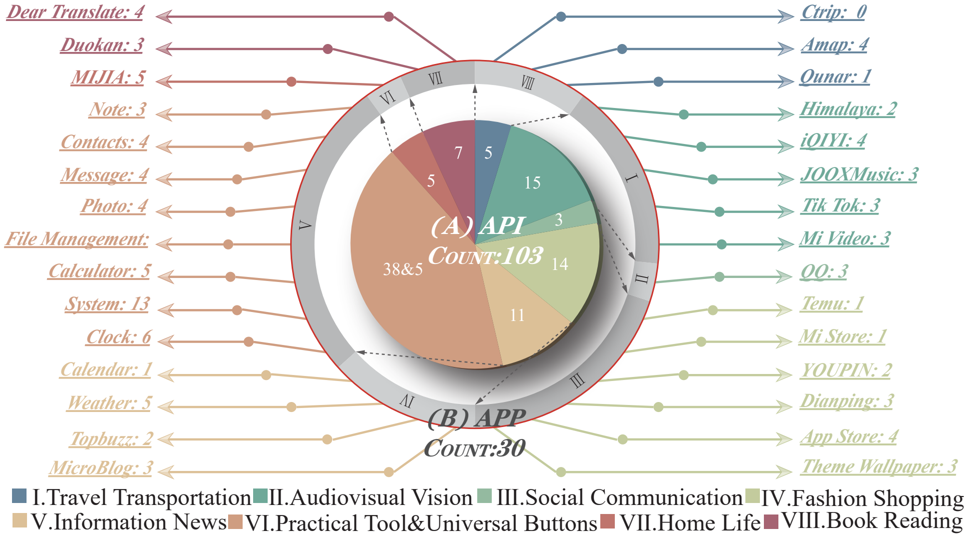


Figure 6: APP classification and quantity chart: The largest category is utility tools, where we categorize fundamental mobile applications. Their distinctive feature is the use of standard API interfaces, and the API functionality is more comprehensive.

ID: 7

Query:

Open Ctrip Travel.

APP:

Ctrip Travel

CheckPoints:

- **Package:**
  ctrip.android.view
- **Key phrase:**
  Ctrip
- **API**:
  - ctrip.android.view/.ui.LocalAppsActivity

Figure 7: A test case in SAST.

ID: 2

Query:

Open Himalaya and play the history.

APP:

ximalaya

CheckPoints:

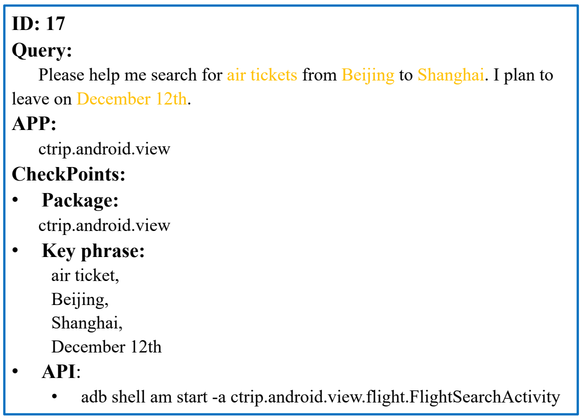
- **Package:**
  com.ximalaya.ting.android
- **Key phrase:**
  Play history | history
- **API**:
  - com.ximalaya.ting.android/.host.activity.MainActivity

Figure 8: A test case in SAMT.

Figure 9 and Figure 10 is an example of a data set and action history. Note that CheckPoints only check successfully executed instructions in the action history. From the action history, we can see that the emulator successfully opened the application by API, perform tasks in ctrip package, and selected "air ticket", "Beijing", and "Shanghai" elements, but failed to input the correct date. According to the definitions of level 1 and level 2 CheckPoints, level 1 CheckPoint score counts package CheckPoints covered, and the score of the example is 1/1, level 2 CheckPoint score counts all CheckPoints covered, and the score of the example is 5/6.

B.5 Supplementary experiments

As can be seen from the table 7, categories with smaller average execution steps generally have higher success rates and CheckPoints scores. Among them, the travel transportation task has the largest average number of execution steps and the lowest PassRate. We can think that more complex tasks require longer execution steps, and the PassRate and CheckPoint score of complex tasks are lower. Travel transportation task contains more uncertainties and it is difficult to determine whether it is completed, so the PassRate is the lowest.



ID: 17

Query:

Please help me search for air tickets from Beijing to Shanghai. I plan to leave on December 12th.

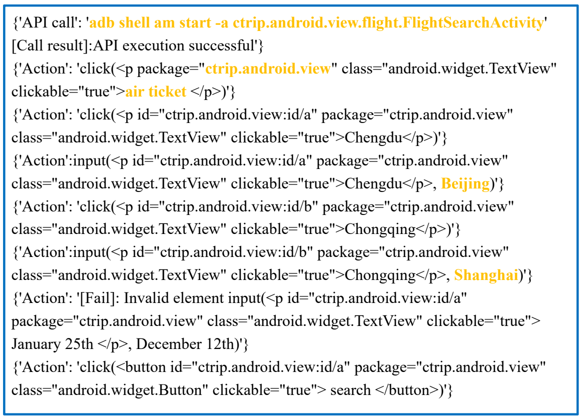
APP:

ctrip.android.view

CheckPoints:

- **Package:**
  ctrip.android.view
- **Key phrase:**
  air ticket,
  Beijing,
  Shanghai,
  December 12th
- **API**:
  - adb shell am start -a ctrip.android.view.flight.FlightSearchActivity

Figure 9: A test case in SAMT.



```
{'API call': 'adb shell am start -a ctrip.android.view.flight.FlightSearchActivity'
[Call result]:API execution successful'}
{'Action': 'click(<p package="ctrip.android.view" class="android.widget.TextView"
clickable="true">air ticket </p>)'}
{'Action': 'click(<p id="ctrip.android.view:id/a" package="ctrip.android.view"
class="android.widget.TextView" clickable="true">Chengdu</p>)'}
{'Action':input(<p id="ctrip.android.view:id/a" package="ctrip.android.view"
class="android.widget.TextView" clickable="true">Chengdu</p>, Beijing)'}
{'Action': 'click(<p id="ctrip.android.view:id/b" package="ctrip.android.view"
class="android.widget.TextView" clickable="true">Chongqing</p>)'}
{'Action':input(<p id="ctrip.android.view:id/b" package="ctrip.android.view"
class="android.widget.TextView" clickable="true">Chongqing</p>, Shanghai)'}
{'Action': '[Fail]: Invalid element input(<p id="ctrip.android.view:id/a"
package="ctrip.android.view" class="android.widget.TextView" clickable="true">
January 25th </p>, December 12th)'}
{'Action': 'click(<button id="ctrip.android.view:id/a" package="ctrip.android.view"
class="android.widget.Button" clickable="true"> search </button>)'}
```

Figure 10: A action history of a test case in SAMT.

APP Category	Case Quantity	Average #Steps	PassRate(%)	CheckPoint$_{l1}$	CheckPoint$_{l2}$
Travel Transportation	18	8.17	39	83	68
Audiovisual Vision	34	4.03	82	68	72
Social Communication	30	6.40	77	57	63
Fashion Shopping	35	7.97	54	63	61
Information News	24	6.46	67	83	68
Practical Tool	61	2.08	89	87	89
Home Life	46	1.67	89	72	91
Book Reading	23	4.17	78	74	84
Universal Buttons	61	1.20	98	98	99

Table 7: Results on SAST classified by APP categories

C Details for Baseline Model

C.1 Examples for HTML

Figure 11 shows the correspondence between the components in the UI page and the corresponding HTML code. It is easy to find that most components have text descriptions, but the switch of the alarm clock does not have a corresponding text description, and LLM will hardly think of it. To click this button, therefore, component function exploration is what we need to do next.

C.2 Prompts for application Selection and Planning

You are a large language model agent stored on a mobile phone, below I will provide you with a task, the environment of the current mobile phone interface(Apps information).

Please help me choose the correct APP to perform the task based on the Apps information. If the APP you want is not available on the current page, you can go to play store and download a suitable APP.

On this basis, you should make a simple plan for completing the task.

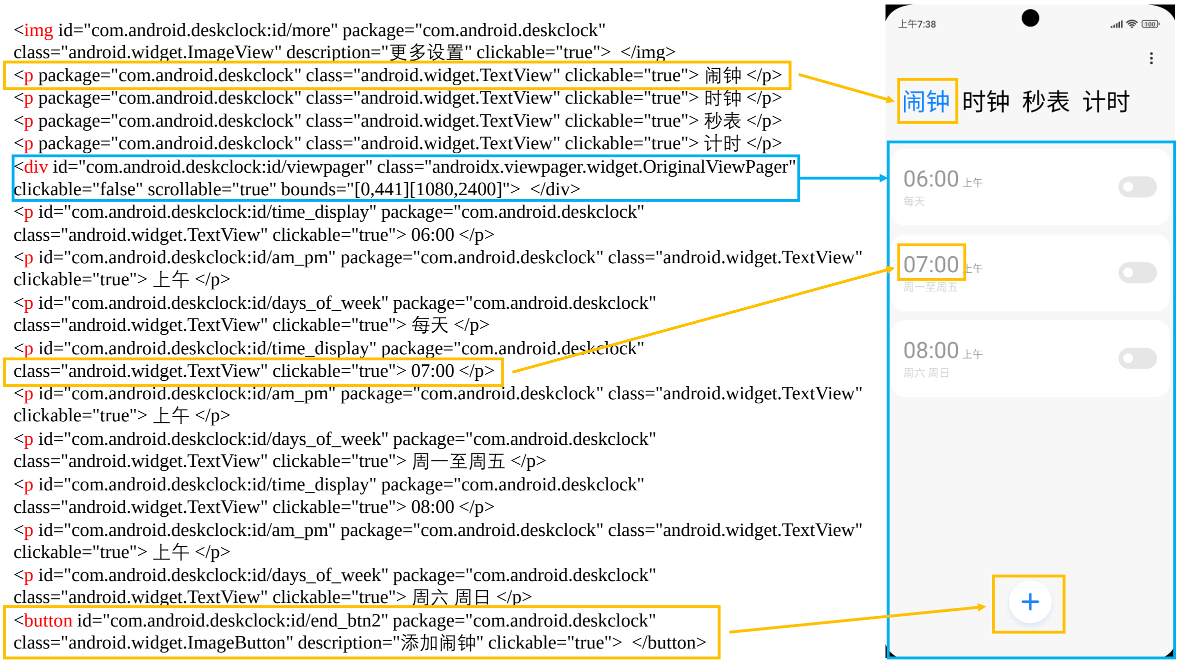
Let's Begin!

C.3 Prompts for API Selection

You are the greatest large language model agent stored on a mobile phone. You will be provided with a API list that can be called by mobile phone, the task you need to complete, the thought about what have done and what need to do now.

You are just the first step to interact with the phone, and your follow-up is UI interaction components. If you find that there is no suitable API and the next step is UI interaction, please answer directly sorry. You should not use the API to complete the work that has been completed by the UI interactive components in the previous steps.

Your decision should consider the following factors:



```
<img id="com.android.deskclock:id/more" package="com.android.deskclock"
class="android.widget.ImageView" description="更多设置" clickable="true">  </img>
<p package="com.android.deskclock" class="android.widget.TextView" clickable="true"> 闹钟 </p>
<p package="com.android.deskclock" class="android.widget.TextView" clickable="true"> 时钟 </p>
<p package="com.android.deskclock" class="android.widget.TextView" clickable="true"> 秒表 </p>
<p package="com.android.deskclock" class="android.widget.TextView" clickable="true"> 计时 </p>
<div id="com.android.deskclock:id/viewpager" class="androidx.viewpager.widget.OriginalViewPager"
clickable="false" scrollable="true" bounds="[0,441][1080,2400]">  </div>
<p id="com.android.deskclock:id/time_display" package="com.android.deskclock"
class="android.widget.TextView" clickable="true"> 06:00 </p>
<p id="com.android.deskclock:id/am_pm" package="com.android.deskclock" class="android.widget.TextView"
clickable="true"> 上午 </p>
<p id="com.android.deskclock:id/days_of_week" package="com.android.deskclock"
class="android.widget.TextView" clickable="true"> 每天 </p>
<p id="com.android.deskclock:id/time_display" package="com.android.deskclock"
class="android.widget.TextView" clickable="true"> 07:00 </p>
<p id="com.android.deskclock:id/am_pm" package="com.android.deskclock" class="android.widget.TextView"
clickable="true"> 上午 </p>
<p id="com.android.deskclock:id/days_of_week" package="com.android.deskclock"
class="android.widget.TextView" clickable="true"> 周一至周五 </p>
<p id="com.android.deskclock:id/time_display" package="com.android.deskclock"
class="android.widget.TextView" clickable="true"> 08:00 </p>
<p id="com.android.deskclock:id/am_pm" package="com.android.deskclock" class="android.widget.TextView"
clickable="true"> 上午 </p>
<p id="com.android.deskclock:id/days_of_week" package="com.android.deskclock"
class="android.widget.TextView" clickable="true"> 周六 周日 </p>
<button id="com.android.deskclock:id/end_btn2" package="com.android.deskclock"
class="android.widget.ImageButton" description="添加闹钟" clickable="true">  </button>
```

Figure 11: An example for HTML. The orange box illustrates clickable elements, and the blue frame illustrates the scrollable range.

1. You need to first judge based on the UI information and actions complete whether the planned action has been completed.

2. You must only choose one API that should be executed most at present to finish the first action in next actions.

3. If there is no suitable API, you can just say sorry without providing any additional suggestions.

Strings within "<>" needs to be replaced with specific parameters, you must return a fully executable adb command. Perhaps you can hand over this task to the UI interaction module.

[API list]:

[Examples]:

"adb shell input tap <x> <y>" is strictly prohibited as an answer. Your [Answer] can only follow the two templates: "Yes, the most suitable API function call is [adb command]" or "Sorry, [explain]".

Let's Begin!

C.4 Prompts for UI Selection

You are a large language model agent stored on a mobile phone, You need to give the current one-step action that needs to be taken to complete the task. Below I will provide you with a task, a plan, the environment of the current mobile phone interface(UI information), action history, though about the current status of task completion.

You need to select the most suitable one element and give the corresponding one action based on the UI information and thought. You need to first judge based on the UI information and action history whether the planned action has been completed. Your selection should also consider action history, and have the courage to try new buttons instead of the same buttons from history.

Action can only be the following three functions:

1. click(element)

Click a element, only when clickable="true", the element can be clicked.

2. input(element, text)

When you decide to enter, you first need to select the unit by clicking.

3. scroll $[x_{start}, y_{start}][x_{end}, y_{end}]$

Scroll the screen from $[x_{start}, y_{start}]$ to $[x_{end}, y_{end}]$. The four parameters you fill in cannot be directly the same as x_{min}, y_{min}, x_{max}, y_max. x cannot exceed (x_{min}, x_{max}), and y cannot exceed $(y_{min}, y_{max)}$.

[Examples]:

Remember:

1.Click and input have higher priority than scrolling. Scrolling is only considered when all elements of the current interface are indeed irrelevant to the task.

2.When you fail to try repeatedly in one interface, maybe you can try to turn back to select other options.

3.When you need to switch APPs, you need to return to the desktop first.

4.When input fails multiple times, you should first select it with click.

Let's Begin!

C.5 Prompts for Thought Generation

You are a large language model agent stored on a mobile phone, below I will provide you with a task, a plan, the environment of the current mobile phone interface before action (Previous UI information), current action, the environment of the current mobile phone interface(Now UI information), action history. Action history records completed operations, including click, input, scroll and API list.

You need to summarize these four aspects: changes in the UI page, actions that have been completed, task progress, one next action.

[one next action] need to choose one among click, input, scroll and one API as the next action, and give one and only one operation object. [One next action] strictly refer to [current action] and [action history] result to do the next action.

[action history] are all previous historical actions, and [current action] is the current action that causes the UI page to change.

[Examples]:

Let's Begin!

C.6 Prompts for Task Completion

You are a large language model agent stored on a mobile phone, below I will provide you with a task, the environment of the current mobile phone interface(UI information), historical action information. You need to judge whether the current task has been completed based on the current environment and historical action information.

C.7 Algorithm Examples

This is a running process of the algorithm on a test case of SAMT

[data]:

```
{
  "id": 2,
  "query": [
    "Play recent records in history with Himalaya."
  ],
  "check_point": {
    "activity":[
      "com.ximalaya.ting.android.host.activity.MainActivity",
      & "com.ximalaya.ting.android.host.activity.MainActivity"
    ],
    "key phrase": [
      "Playing history" | "history "
    ],
  "package": "com.ximalaya.ting.android"
  },
```

"domain": "smartApp/Ximalaya"
}

According to algorithm 1, LLM generates a plan based on the query in data as a task and a given list of available applications as follows:

[Task]: Play recent records in history with Himalaya.
[app list]:
[
{
"name": "ctrip",
"function_description": "As an authoritative online travel service company in the industry, Ctrip's travel hotel booking platform covers approximately 1.4 million hotels around the world; air ticket products cover more than 300 international airlines; these airlines operate flights to major cities around the world, providing more than 2 million Air routes connect more than 5,000 cities in about 200 countries and regions..."
},
...
]
[Plan]: I should open the Himalaya app and find the "Playback History" option in the app to view recent playback history.

Then the task plan is executed iteratively. In each iteration, the model accepts task, app list, action history, plan, and current HTML as input, and outputs execution API calls or UI operation instructions.

A short example of AH(action history), thought, Html will be shown below:

[Action History]:
{'API call': 'adb shell am start -n com.ximalaya.ting.android/.host.activity.MainActivity. [Call result]:API execution successful'}
{'Action': '[Fail]: Invalid element click(<div id="com.ximalaya.ting.android:id/xm_ad_host_count_down_click_lay_no_click_view" package="com.ximalaya.ting.android" class="android.view.View" description="Skip ads" clickable="true"> </div>)'}
{'Action': 'click(<button id="com.ximalaya.ting.android:id/tab_myspace_and_listen" package="com.ximalaya.ting.android" class="android.widget.RadioButton" clickable="true"> my </button>)'}
[thought]:
Changes: The current page is the main page of the Ximalaya app. There is no change between the previous and current pages.
Task progress: From the current action and action history, I am currently on the main page of the Ximalaya app, ready to check the play history.
Actions completed: From the current action and action history, I have opened the Ximalaya app.
One next action: Click on the "play history" option to view the recent play history.
[HTML]:
<p package="com.ximalaya.ting.android" class="android.widget.TextView" clickable="true"> membership </p>
<p id="com.ximalaya.ting.android:id/main_tv_user_level" package="com.ximalaya.ting.android" class="android.widget.TextView" clickable="true"> user level </p>
<p id="com.ximalaya.ting.android:id/main_tv_listen_duration_title" package="com.ximalaya.ting.android" class="android.widget.TextView" clickable="true"> Listen (minutes) </p>
<p package="com.ximalaya.ting.android" class="android.widget.TextView" clickable="true"> fans </p>
<p package="com.ximalaya.ting.android" class="android.widget.TextView" clickable="true"> Follow </p>
<p package="com.ximalaya.ting.android" class="android.widget.TextView" clickable="true"> live </p>
<button package="com.ximalaya.ting.android" class="android.widget.Button" clickable="true"> message

</button>
<button package="com.ximalaya.ting.android" class="android.widget.Button" clickable="true"> history </button>
<button package="com.ximalaya.ting.android" class="android.widget.Button" clickable="true"> favorite </button>
<button package="com.ximalaya.ting.android" class="android.widget.Button" clickable="true"> local </button>
<button package="com.ximalaya.ting.android" class="android.widget.Button" clickable="true"> purchased </button>
<img package="com.ximalaya.ting.android" class="android.widget.ImageView" description="play" clickable="true"> </img>
A example of api action or ui action will be shown below:
[Action]: [adb shell am start -n com.ximalaya.ting.android/.host.activity.MainActivity]
[Action]: click(<button package="com.ximalaya.ting.android" class="android.widget.Button" clickable="true"> history </button>)

After successful execution, the current action will be added to the action history, the updated HTML of the emulator will be read, and handed over to LLM to generate a new thought and determine whether the task is over.
[thought]:
Changes: The current page is the "My" page in the Ximalaya app.
Actions Complete: I have opened the Ximalaya app and clicked the "my" button, then clicked the "play history" button.
Task progress: The current mission progress is to view the play history.
One next action: Click on the "play" item to continue playing.
[Finished]: No, task is not finished.