

AndroidLab: Training and Systematic Benchmarking of Android Autonomous Agents

Yifan Xu^{1*†}, Xiao Liu^{13*}, Xueqiao Sun^{1†}, Siyi Cheng^{2†}, Hao Yu^{1†}, Hanyu Lai^{1†},
Shudan Zhang^{1†}, Dan Zhang^{1†}, Jie Tang¹, Yuxiao Dong¹

¹Tsinghua University ²Peking University ³Zhipu AI

Abstract

Autonomous agents have become increasingly promising for interacting with the real world. Android systems are widely adopted for mobile interactions, but existing studies for training and evaluating Android agents lack systematic research for both open and closed-source models. In this work, we present ANDROIDLAB as a systematic framework for building Android agents. It provides an operation environment with different modalities and a reproducible benchmark. It supports both large language models (LLMs) and vision language models (VLMs) in the same action space. The ANDROIDLAB benchmark includes predefined Android virtual devices and 138 tasks across nine Apps. By using the ANDROIDLAB environment, we develop an Android Instruction dataset and evaluate its effectiveness on six open LLMs and VLMs. The results show that the data helps increase the average success rate from 4.59% to 21.50% for LLMs, and from 1.93% to 13.28% for VLMs. ANDROIDLAB is open-sourced and publicly available at <https://github.com/THUDM/Android-Lab>.

1 Introduction

Developing autonomous agents capable of executing human instructions on mobile operating systems can greatly change the way we interact with our phones (Burns et al., 2021; Yang et al., 2023b; Wang et al., 2023a; Hong et al., 2023; Rawles et al., 2023; Li et al., 2020; Romao et al., 2019; Rai et al., 2019). Recently, a significant line of research has focused on using large language models (LLMs) (Zeng et al., 2022; OpenAI, 2023; Anthropic, 2023; Team et al., 2024; GLM et al., 2024) and vision language models (VLMs) (OpenAI, 2023; Anthropic, 2023; Hong et al., 2023) as the foundation for mobile agents (Deng et al., 2023; Rawles et al., 2023; Zhou et al., 2023).

Despite these advancements (Zhang et al., 2025a), training and evaluating mobile agents still face challenges due to a lack of standard frameworks. Previous benchmarks (Rawles et al., 2023; Sun et al., 2022; Li et al., 2020) often rely on static environments, where agents are expected to predict actions based on screenshots without actual interactions. AndroidEnv (Toyama et al., 2021) was the first interactive environment for mobile agents, with follow-up efforts (Lee et al., 2024; Rawles et al., 2024) in improving reproducibility. In addition, existing benchmarks (Yang et al., 2023b; Xing et al., 2024; Lee et al., 2024; Rawles et al., 2024) mostly only evaluate closed-source models using prompt-based improvements. This restricts the potential to analyze model behaviors and perform reinforcement learning experiments.

In light of these issues, we aim to develop an Android agent framework for both training and evaluating. In this paper, we propose **ANDROIDLAB**: a standard operating environment and a benchmark for testing agents on Android devices. Figure 1 gives an overview of AndroidLab. First, we define two basic operation modes to align actions and objects for LLMs and VLMs from Android systems: XML mode (using XML data) and SoM mode (using screenshots). Each mode is then supported by both ReAct (Yao et al., 2022b) and SeeAct (Zheng et al., 2024).

Second, we develop the ANDROIDLAB benchmark by building upon this environment. The benchmark contains 138 tasks across nine Apps. To ensure reproducibility and eliminate external network or time dependencies, we use Android virtual devices preloaded with app operation histories and offline data. Specifically, each task is split into multiple required page states as sub-goals. Task completion and progress are precisely checked using UI tree structure matching, allowing nearly all tasks to be assessed without restrictive device states. We also introduce new metrics (reversed redun-

*Yifan and Xiao contributed equally. Emails: xu-yf23@mails.tsinghua.edu.cn, shawliu9@gmail.com

†Work was partially done when interned at Zhipu AI.

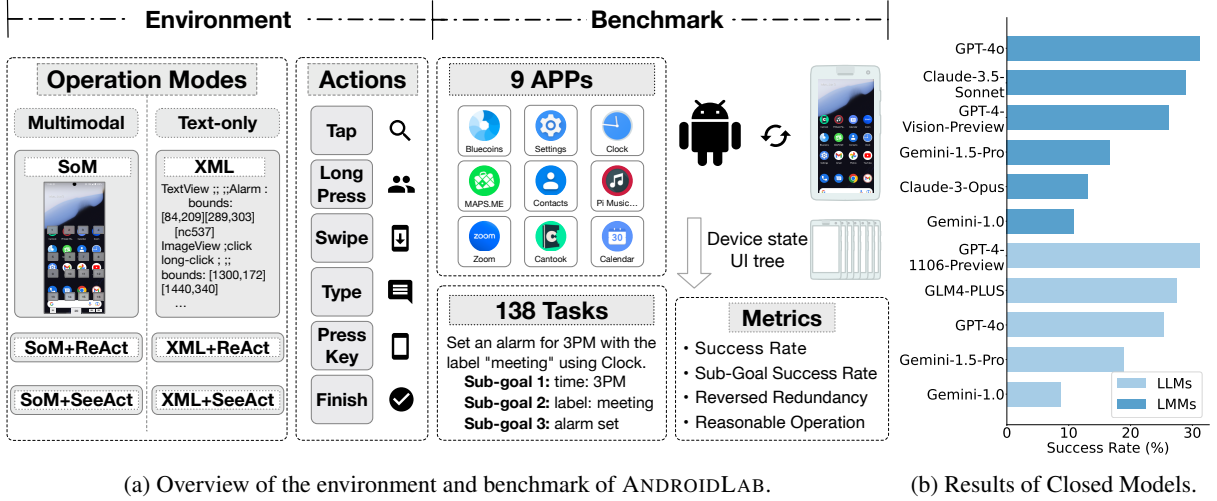


Figure 1: (a) The SoM mode for VLMs and the XML mode for LLMs in ANDROIDLAB ensure an identical action space. ReAct and SeeAct are supported in both modes. (b) The task success rates of proprietary models on ANDROIDLAB. In XML mode, GPT-4-1106-Preview achieves the highest success rate at 31.16%, matching GPT-4o in SoM mode.

dancy and reasonable operation) to measure action efficiency. As a first step, we use the benchmark to evaluate 17 open and closed-source models.

Third, by using ANDROIDLAB’s operation modes and action space, we further construct the Android Instruct dataset. To achieve this, we develop an online annotation tool with the same action space, collecting 10,500 traces and 94,300 steps. Among these, 6,208 steps are from the Apps included in ANDROIDLAB, and we use this portion for model fine-tuning. The resultant dataset includes tasks, phone screen states, XML, and operations. We fine-tune six open LLMs and VLMs. As shown in Figure 2, the dataset help increase the average success rates of LLMs from 4.59% to 21.50%, and of VLMs from 1.93% to 13.28%. Further analysis shows that fine-tuning improves operational accuracy and efficiency, and reduces redundancy in Android agents.

The contributions are summarized as follows:

- We design ANDROIDLAB, which includes a standard operating environment and a benchmark. It unifies the evaluation and training of both LLM- and VLM-based Android agents.
- We develop the ANDROIDLAB benchmark as a reproducible test environment for evaluating Android agents. It covers 138 tasks for nine frequently-used Apps. Currently, the top model (GPT-4o) achieves a success rate of 31.16%, making it a challenging benchmark to date¹.

¹Parts of the AndroidLab benchmark’s SoM modes are

- We construct an Android Instruct dataset based on ANDROIDLAB. It supports both LLM and VLM fine-tuning. Experiments show that open and small models fine-tuned with this dataset can achieve performance comparable to proprietary models, thus offering the balance between efficiency and accuracy.

2 Related Work

Benchmarks for Agents. Recent advancements in large foundation models have led to new agent benchmarks tailored to these models. Agents interact with external environments primarily through writing code (Chen et al., 2021; Zheng et al., 2023; Zhang et al., 2024; Austin et al., 2021; Xia et al., 2024; Zhang et al., 2025b), or invoking APIs (Guo et al., 2024; Li et al., 2023; Peng et al., 2021). Specialized benchmarks have been designed for interaction with operating systems, categorized into Desktop and Mobile. For Desktop, static benchmarks (Mialon et al., 2023; Deng et al., 2023; Kapoor et al., 2024) evaluate agents by single-step operation or operations sequence without a virtual environment. Otherwise, dynamic benchmarks provide interactive web browser (Liu et al., 2018; Zhou et al., 2023; Yao et al., 2022a; Koh et al., 2024) or Unix-like system virtual environment (Hong et al., 2023; Xie et al., 2024), making evaluation more

included in the VisualAgentBench (Liu et al., 2024) as the VAB-Mobile component.

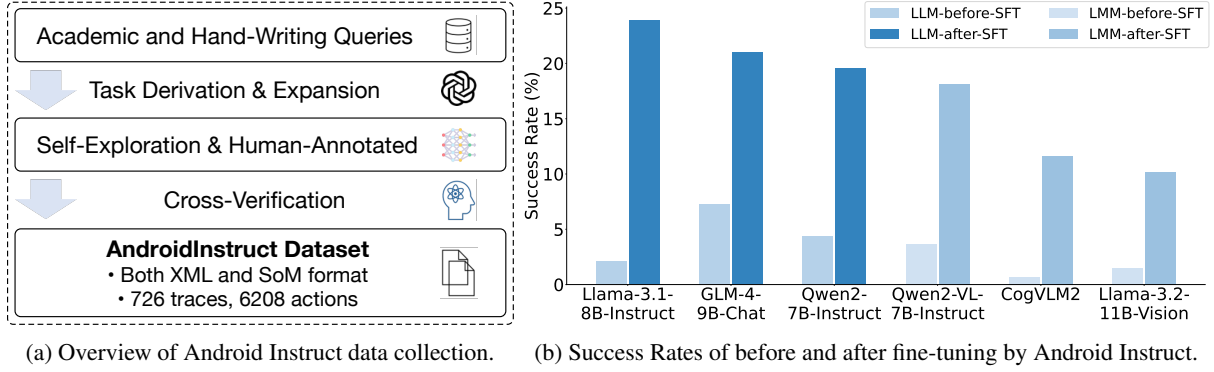


Figure 2: (a) We collect 726 traces containing 6208 fully aligned steps of XML and SoM mode training data. (b) By using the Android Instruct dataset, we train six open text-only and vision language models, increasing average success rates from 4.59% to 21.50% for LLMs and from 1.93% to 13.28% for VLMs, respectively, matching the performance of proprietary models.

flexible and realistic.

Mobile benchmarks for Android began with static systems like PixelHelp (Li et al., 2020) and MetaGUI (Sun et al., 2022) and later expanded through AITW (Rawles et al., 2023), which provided over 5 million images. AndroidEnv (Toyama et al., 2021) introduced dynamic evaluations, while Android Arena (Xing et al., 2024) added cross-app evaluations. Although task diversity was limited, B-MOCA (Lee et al., 2024) standardized the Android Virtual Device. AndroidWorld (Rawles et al., 2024) offers reward signals for 116 tasks across 20 real-world apps but does not support instruction-tuning data construction.

Agents for Interactive System. For Web environments, WebGPT (Nakano et al., 2021) and WebGLM (Liu et al., 2023) integrate LLMs for improved question-answering. MindAct (Deng et al., 2023), WebAgent (Gur et al., 2023), and AutoWebGLM (Lai et al., 2024) focus on executing complex interactive tasks. In mobile agents, early work on Android systems utilized multiple execution modules (Burns et al., 2021; Venkatesh et al., 2023; Li et al., 2020; Zhan and Zhang, 2023). PixelHelp (Li et al., 2020) mapped actions to images, while Auto-GUI (Zhan and Zhang, 2023) used image and text encoders with LLMs for CoT outputs. CogAgent (Hong et al., 2023) achieved SOTA on AITW (Rawles et al., 2023) by combining modules for action prediction. Recent zero-shot mobile agents using GPT-4V (OpenAI, 2023) have shown strong results (Yang et al., 2023b; Zheng et al., 2024; Yan et al., 2023; Wang et al., 2023a), but planning complexity limits inference speed and practical deployability due to security restrictions.

3 ANDROIDLAB

3.1 The Operating Environment

ANDROIDLAB defines a set of action spaces and two operation modes, forming the ANDROIDLAB environment. We adopt the main action space from prior work and add a model return value (finish action). The two basic operation modes are SoM (Yang et al., 2023a) and XML, differing in whether the agent can access a snapshot of the phone screen. For comparison, we also implement ReAct (Yao et al., 2022b) and SeeAct (Zheng et al., 2024). This framework supports real and virtual Android devices and is compatible with Android-like mobile operating systems.

Action Space. Based on the action spaces from AppAgent (Yang et al., 2023b) and Android Env (Toyama et al., 2021), we define four basic phone operations: Tap, Swipe, Type, Long Press, along with two shortcut keys, Home and Back, as the core action space. We add the Finish action as the final step, allowing the agent to return execution results or answers. This action space applies to all modes.

XML Mode. XML mode is tailored for text-only input models (LLM). Inspired by Android Arena (Xing et al., 2024), we redesign the XML compression algorithm to convey screen information. The LLM selects corresponding elements directly for operations.

SoM Mode. SoM mode is for multimodal input models (VLM), based on the Set-of-Mark method (Yang et al., 2023a). Each clickable or focusable element is assigned a serial number, and the VLM selects the element by its number. The selected elements in SoM mode align with those in

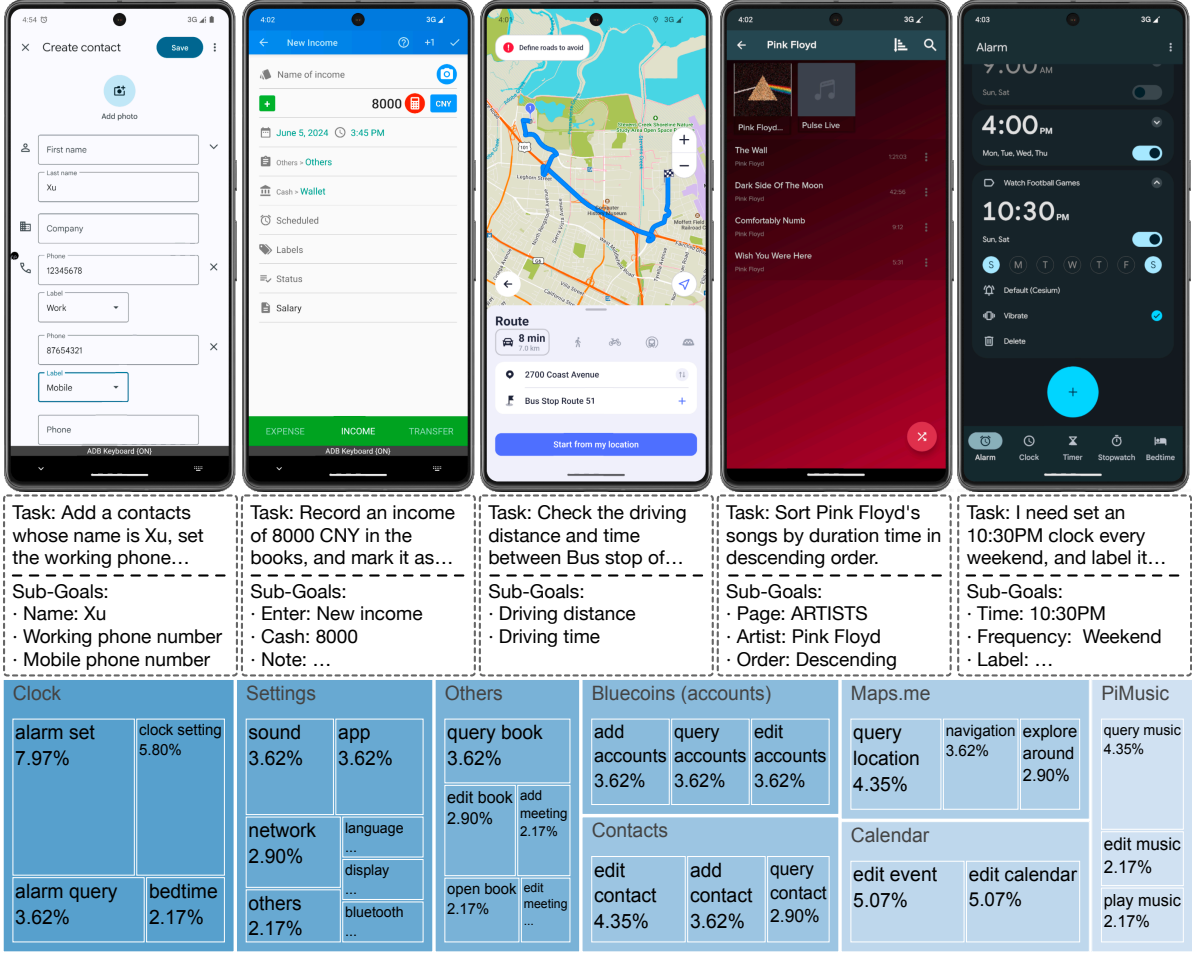


Figure 3: Task examples and the distribution of all apps and subcategories in the ANDROIDLAB benchmark. We decomposed each task into sub-goals and evaluated them independently. A task is considered complete only if all sub-goals are correctly addressed.

the compressed XML list, allowing both modes to interact with the same action space and objects.

These basic operation modes directly require the agent to output operation commands. Based on these two methods, we further test two novel agent frameworks, ReAct (Yao et al., 2022b) and SeeAct (Zheng et al., 2024). These two frameworks allow the agent to observe and reflect on the environment or more easily select specific tasks to execute. Please refer to Appendix B for more details about our operation modes.

ReAct modes. Based on the above two modes, we follow (Yao et al., 2022b) to prompt the model, allowing models to think step by step and output their thought and reasoning process.

SeeAct modes. Following (Zheng et al., 2024), we separate the reasoning and element grounding process. We instruct models to interact for two rounds in a single operation. The models are supposed to generate a detailed description of the desired action

and output the real action, respectively.

3.2 The Reproducible Benchmark

Based on the environment, ANDROIDLAB benchmark offers a deterministic and reproducible evaluation platform, allowing users to perform fair and challenging comparisons of Android agent capabilities. ANDROIDLAB benchmark introduces the following designs:

- We gathered 138 tasks from nine apps, ensuring reproducibility. These tasks, derived from common mobile scenarios, are divided into two types: (a) Operation Tasks, where agents must complete a series of actions to meet a goal, and (b) Query Tasks, where agents answer queries based on phone information.
- Using phone XML data, we identify screen information that uniquely defines task completion, making task completion our primary metric. Additionally, we select auxiliary metrics such as the

proportion of valid actions and the redundancy of successful operation sequences.

3.2.1 Task Formulation

We formalize each task input as a 4-tuple: $\text{Task}(E, I, F, M)$. Here, E represents the execution environment of the task, which, in the context of benchmark testing, is the pre-packaged AVD (Android virtual device) image. This includes a fixed phone screen size, Android version, API level, and a fixed app usage state. I denotes the specific natural language instruction for the task. To avoid confusion during testing, we specify the app required to complete the task in natural language. F represents the agent testing framework. Finally, M denotes the backbone model used to perform the task, primarily referring to LLMs or VLMs.

Thus, we can formally define the two types of tasks included in ANDROIDLAB:

Operation Task. $T(E, I, F, M) \rightarrow (S_1, \dots, S_n)$. The output of this type of task is a sequence of continuous Android virtual machine states.

Query Task. $T(E, I, F, M) \rightarrow (S_1, \dots, S_n, A)$. This type of task assesses the agent’s ability to answer specific questions based on the state sequence after exploration. The model must explore the environment to find the answers and output the correct response.

Based on the above formulation, we design 138 tasks, including 93 Operation Tasks and 45 Query Tasks. Please refer to Appendix A for detailed information.

3.2.2 Reproducible Designs

To ensure our evaluation reflects real-world agent usage scenarios with an appropriate level of difficulty and full reproducibility, we design the tasks with the following considerations:

- **Fixed Evaluation Time and Space:** We use ADB commands at the start of each evaluation to set the machine’s time and virtual geolocation to predetermined values.
- **Offline Testing:** All test apps function offline, with preloaded usage records in the AVD image to ensure normal usability without an internet connection.
- **Predefined Answers:** For query-based tasks, we conduct operations on the corresponding apps in advance to guarantee uniquely determined correct results.

3.2.3 Metrics

Previous evaluations with virtual environments have relied on indirect metrics like single-step accuracy and operation path matching, leading to imprecise assessments. In response, ANDROIDLAB benchmark introduces a task-completion-based evaluation system that judges directly from device and screen states. Our key metrics are:

- **Success Rate:** For Operation Tasks, we divided a complete task into multiple sub-goals and identified the specific page information for each sub-goal completion. By checking and matching specific UI tree elements, we assess each sub-goal completion status individually. The task is considered successfully executed when all sub-goals are completed. We have also set up a few tasks that can directly use the device state to determine if they were completed correctly. For Query Tasks, advanced LLMs verify if the model’s predicted results match the standard answers, avoiding errors from direct string comparisons. We provide an example in Fig 4.
- **Sub-Goal Success Rate:** Tasks are decomposed into sub-goals, and completion is assessed sequentially. This finer metric rewards models with stronger understanding and operational capabilities. Only Operation Tasks include the Sub-Goal Success Rate.
- **Reversed Redundancy Ratio:** As in prior work (Xing et al., 2024), redundancy is measured by comparing the model’s operation path length to a human benchmark. We calculate this for completed tasks and take the reciprocal, so higher values indicate less redundancy. We do not report $SR < 5$ because there are too few completed tasks, which may be affected by a small number of special values. It should also be emphasized that this metric may exceed 100 because the steps of human operation are not necessarily optimal.
- **Reasonable Operation Ratio:** This metric evaluates the proportion of operations after which the screen changed. Unchanged screens indicate the operation was ineffective and thus deemed unreasonable.

By incorporating these metrics, our evaluation system provides a comprehensive and precise assessment of an agent’s performance in completing specified tasks.

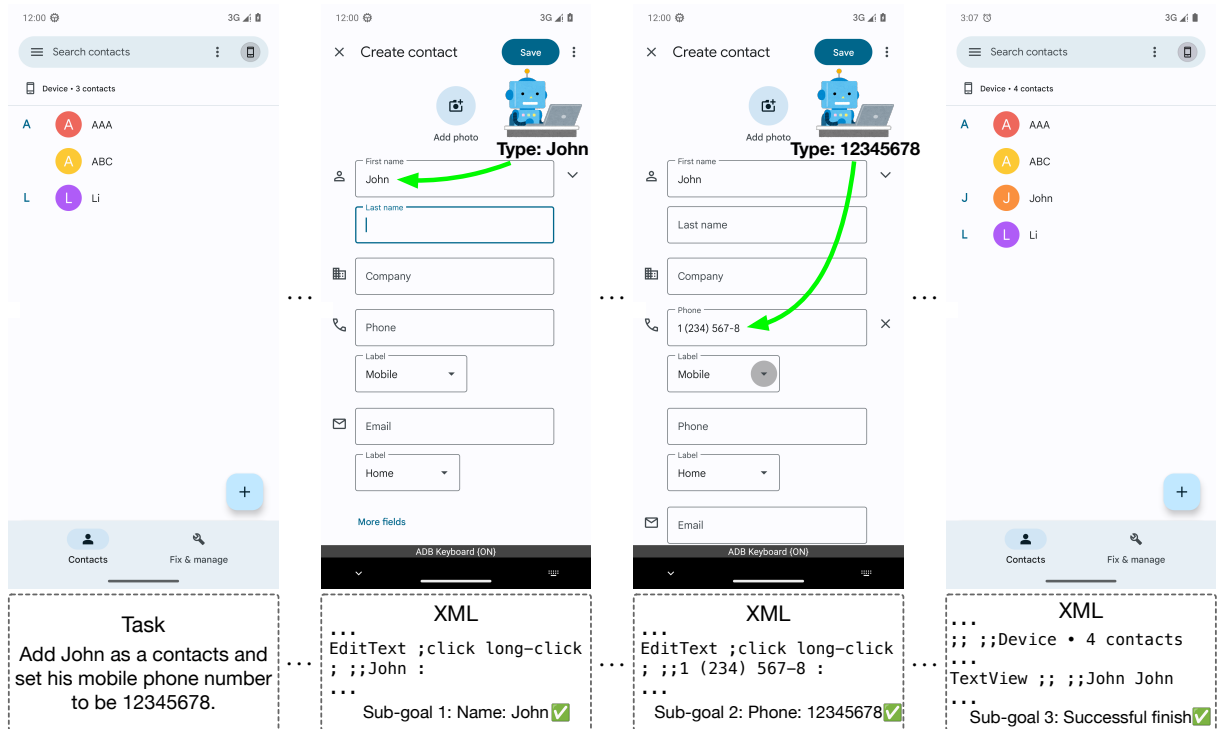


Figure 4: An example of an agent completing all sub-goals of the entire task. We only present the starting and ending steps, along with the steps where the agent completes each sub-goal. It is essential that we record the completion status of each sub-goal. Without this information, we may not be able to obtain detailed information from the XML of the finished page, which could lead to a misjudgment of the task.

4 Android Instruction Data

Building an open-source, deployable Android operation agent is a significant challenge in AI research. Previous work on Android agents has focused on using powerful closed-source models to design interaction logic (Zheng et al., 2024; Yang et al., 2023b; Wang et al., 2023a), raising concerns about accessibility, privacy, and efficiency. To address this, we aim to build an open-source mobile agent. The main challenge lies in generating training data for mobile operations to handle open-world tasks in diverse environments.

We propose task derivation and expansion methods for task generation, allowing models to generate tasks for specific apps controllably. ANDROID-LAB connects to devices via ADB, enabling compatibility with various real or virtual devices for data generation. Using self-exploration and manual annotation, we generate example operation traces. To make it easier for annotators to work on real devices (rather than emulators), we developed an online annotation tool. This tool uses ADB commands to monitor user interactions on the phone and captures screenshots and page XML before each action. Our Android Instruction data is built

on the Task (E, I, F) framework within ANDROID-LAB’s environment.

4.1 Data Construction

The primary challenges in data construction include generating executable Android instructions and annotating operation path data. Our approach involves three steps:

- **Task Derivation and Expansion:** We use academic datasets (Rawles et al., 2023; Coucke et al., 2018) and manually write instructions to seed task generation. Language models are employed to create additional tasks, which are reviewed and added to the dataset, ensuring realistic and executable instructions.
- **Self-Exploration:** LLMs and VLMs are used for automatic task exploration, outputting *finish* when done. Initially, manual selection was used to verify results, but a reward model later replaced it after gathering 500 traces. In this step, we ultimately removed the self exploration operation data from the instruction fine-tuning dataset and only retained the reward model for the next step.
- **Manual Annotation:** This process involves four steps: (1) **Instruction Check**, where annotators

evaluate the feasibility of the given task; (2) **Preliminary Familiarization**, allowing them to explore the app interface before performing tasks; (3) **Task Execution**, in which the annotators execute and document each task step; and (4) **Cross-Verification**, where reward models and a second annotator reviews the task trace to ensure its accuracy.

This combination of autonomous and manual processes resulted in 10.5k traces and 94.3k steps, and we use 726 traces and 6208 steps derived from the Apps included in the ANDROIDLAB benchmark for training. More details are in Appendix C.

4.2 Annotation Tool

To more accurately and efficiently record operation trajectories and page information (XML), we design an annotation tool.

Acquisition of Page Information: ADB is the most common tool for retrieving page XML (Yang et al., 2023b; Rawles et al., 2024). However, it may fail on pages with dynamic elements, such as playback bars, as it waits for all UI components to become idle. If this takes too long, XML acquisition times out. To overcome this, we reimplemented XML retrieval using the Android Accessibility Service, letting annotators manually decide the best moment to extract page data.

Recording Operation Trajectories: We record three main user actions: clicks, swipes, and text input. Annotators perform clicks and swipes on the phone, while ADB captures screen events to classify them. For text input, we use the ADB keyboard to complete input in one step. Before each action, annotators must record the current page to ensure consistency between the data and the observed interface.

4.3 Training

To explore the effectiveness of our dataset on lightweight open-source models, we select Llama-3.1-8B-Instruct, GLM-4-9B-Chat, Qwen2-7B-Instruct, Llama-3.2-11B-Vision-Instruct, Qwen2-VL-7B-Instruct and CogVLM2 (cogvlm2-llama3-chat-19B) as the training backbones for LLMs and VLMs, respectively. Due to our preliminary experiments showing that training agents from base models yields better results, we select the base versions of all models for fine-tuning, except for Qwen2-VL-7B-Instruct (as no open-source base model is available). However, we still report the instruct ver-

sions as baselines because the base models cannot follow instructions without further tuning. For all training sessions, we use a batch size of 32 and a maximum sequence length of 4096, training for five epochs. The learning rate is set to $1e-5$.

5 Experiments

5.1 Experiment Setup

Evaluation Settings. In preliminary tests, we found that even though we specified the use of certain apps in the instructions, agents failed to complete tasks because they could not launch the respective apps correctly. To avoid errors caused by a single reason, we start tasks directly within the specified app in the formal experiments and then allow the agent to proceed. Additionally, we set a maximum execution step limit of 25 for each task, with a 3-second interval for the virtual machine to respond to each operation. We generate by greedy search for each task of all models.

Baseline Models. For large language models (LLMs) with text-only input capability, we selected GPT-4o (OpenAI, 2023), GPT-4-1106-Preview (OpenAI, 2023), Gemini-1.5-Pro (Team et al., 2024), Gemini-1.0 (Team et al., 2024), GLM-4-PLUS (GLM et al., 2024), Llama-3.1-8B-Instruct (Touvron et al., 2023), GLM-4-9B-Chat (GLM et al., 2024) and Qwen2-7B-Instruct (Bai et al., 2023) as the baselines for testing in the XML mode. For vision language models (VLMs) with image input capability, we chose GPT-4o (OpenAI, 2023), GPT-4-Vision-Preview (OpenAI, 2023), Gemini-1.5-Pro (Team et al., 2024), Gemini-1.0 (Team et al., 2024), Claude-3.5-Sonnet, Claude-3-Opus (Anthropic, 2023), Llama-3.2-11B-Vision-Instruct (Touvron et al., 2023), Qwen2-VL-7B-Instruct (Wang et al., 2024) and CogVLM2 (Wang et al., 2023b) as the baselines for testing in the SoM mode. We also further evaluated the performance of GPT-4o and Gemini-1.5-Pro under the ReAct and SeeAct frameworks in both modes.

5.2 Main Results

As shown in Table 1, in the XML mode, GPT-4-1106-Preview outperforms the other models with a Success Rate (SR) of 31.16%, the highest in this mode while also achieving the best Sub-Goal Success Rate (Sub-SR) at 38.21%. Although GPT-4o exhibits slightly lower SR (25.36%), it achieves the highest Reversed Redundancy Ratio (RRR) at 107.45, indicating its strong ability to reduce unnec-

Table 1: **Results of XML and SoM modes.** SR, Sub-SR, RRR, and ROR stand for Success Rate, Sub-Goal Success Rate, Reversed Redundancy Ratio, and Reasonable Operation Ratio, respectively. For all these metrics, a higher value means better. **-ft** represents a finetuned model. In each mode, **Bold** represents the best result. We do not report RRR score if SR < 5.

| Mode | Model | SR | Sub-SR | RRR | ROR |
|---------|------------------------------|--------------|--------------|---------------|--------------|
| XML | GPT-4o | 25.36 | 30.56 | 107.45 | 86.56 |
| | GPT-4-1106-Preview | 31.16 | 38.21 | 66.34 | 86.24 |
| | Gemini-1.5-Pro | 18.84 | 22.40 | 57.72 | 83.99 |
| | Gemini-1.0 | 8.70 | 10.75 | 51.80 | 71.08 |
| | GLM4-PLUS | 27.54 | 32.08 | 92.35 | 83.41 |
| | LLaMA3.1-8B-Instruct | 2.17 | 3.62 | - | 52.77 |
| | Qwen2-7B-Instruct | 4.35 | 4.95 | - | 67.26 |
| | GLM4-9B-Chat | 7.25 | 9.06 | 54.43 | 58.34 |
| XML+SFT | LLaMA3.1-8B-ft | 23.91 | 30.31 | 75.58 | 92.46 |
| | Qwen2-7B-ft | 19.57 | 24.40 | 77.31 | 92.48 |
| | GLM4-9B-ft | 21.01 | 26.45 | 74.81 | 93.25 |
| SoM | GPT-4o | 31.16 | 35.02 | 87.32 | 85.36 |
| | GPT-4-Vision-Preview | 26.09 | 29.53 | 99.22 | 78.79 |
| | Gemini-1.5-Pro | 16.67 | 18.48 | 105.95 | 91.52 |
| | Gemini-1.0 | 10.87 | 12.56 | 72.52 | 76.70 |
| | Claude-3.5-Sonnet | 28.99 | 32.66 | 113.41 | 81.16 |
| | Claude-3-Opus | 13.04 | 15.10 | 81.41 | 83.89 |
| | CogVLM2 | 0.72 | 0.72 | - | 17.97 |
| | LLaMA3.2-11B-Vision-Instruct | 1.45 | 1.45 | - | 50.76 |
| | Qwen2-VL-7B-Instruct | 3.62 | 4.59 | - | 84.81 |
| SoM+SFT | CogVLM2-ft | 11.59 | 16.06 | 57.37 | 85.58 |
| | LLaMA3.2-11B-Vision-ft | 10.14 | 12.98 | 61.67 | 87.85 |
| | Qwen2-VL-7B-Instruct-ft | 18.12 | 22.64 | 65.23 | 88.29 |

essary operations. The ROR metric shows that both models in the GPT-4 series perform comparably, with around 86% of operations being reasonable but with room for improvement in efficiency. Other models, such as Gemini-1.5-Pro, show moderate performance, with ROR around 80, but lag in SR.

In the SoM mode, GPT-4o again shows dominance, reaching an SR of 31.16% and a Sub-SR of 35.02%, the highest in both categories. GPT-4-Vision-Preview follows closely, but models like Claude-3.5-Sonnet exceeded GPT-4o in RRR (113.40), demonstrating a higher efficiency in task completion with fewer redundant steps.

Fine-tuning improves several models across both modes, notably boosting the Success Rate and ROR of all fine-tuned open-source models. Fine-tuning notably increased the Success Rate (SR) for models like LLaMA3.1-8B and Qwen2-7B, raising their SR from 2.17 to 23.91 and 4.35 to 19.57, respectively. The Reasonable Operation Ratio (ROR) also

saw improvements, with models such as CogVLM2 jumping from 17.97 to 85.58 after fine-tuning.

5.3 Additional Findings

Influence of Instruction Tuning. Instruction tuning significantly enhances the performance of models across all four metrics in both XML and SoM modes, lifting the average success rates from 4.59% to 21.50% for LLMs and from 1.93% to 13.28% for VLMs. Notably, GLM4-9B’s success rate rose to 21.01%, with its Reasonable Operation Ratio (ROR) improving to 93.25, indicating better operational efficiency. The Reversed Redundancy Ratio (RRR) saw consistent gains, demonstrating reduced unnecessary actions, such as GLM4-9B improving its RRR from 54.43 to 74.81.

In SoM mode, models like CogVLM2, LLaMA3.2-11B, and Qwen2-VL-7B showed significant advancements across all four metrics. Qwen2-VL-7B’s SR increased from 3.62 to

18.12%, and its ROR rose to 88.29. The SubSR and RRR also benefited from tuning, marking improved task breakdown and reduced redundancy. After tuning, the best-performing open-source LLMs are approaching the level of GPT-4o, while the top VLMs have surpassed Gemini-1.5-Pro, reflecting comprehensive improvements across success, operational efficiency, and task execution. The tuned models’ effective actions (ROR) have also surpassed those of most closed-source models, demonstrating enhanced precision.

Table 2: The impact of the ReAct and SeeAct frameworks on SR results. Notably, model performance is significantly improved in XML+ReAct mode. Full results of this table are shown in Appendix D.3

| Mode | Model | SR |
|------------|----------------|-------|
| XML | GPT-4o | 25.36 |
| | Gemini-1.5-Pro | 18.84 |
| XML+ReAct | GPT-4o | 33.33 |
| | Gemini-1.5-Pro | 31.16 |
| XML+SeeAct | GPT-4o | 24.64 |
| | Gemini-1.5-Pro | 21.01 |
| SoM | GPT-4o | 31.16 |
| | Gemini-1.5-Pro | 16.67 |
| SoM+ReAct | GPT-4o | 31.88 |
| | Gemini-1.5-Pro | 15.94 |
| SoM+SeeAct | GPT-4o | 30.43 |
| | Gemini-1.5-Pro | 21.01 |

Table 3: Average generation tokens of different modes. We used the LLaMA3 tokenizer for calculation. FT represents instruction tuning models.

| Mode | FT | XML/SoM | ReAct | SeeAct |
|-------------------|------|---------|-------|--------|
| #Avg. Gen. Tokens | 4.96 | 23.56 | 67.89 | 129.12 |

Analysis of Agent Frameworks. We assess ReAct and SeeAct frameworks with GPT-4o and Gemini-1.5-Pro in XML and SoM modes. Table 2 shows ReAct significantly improves performance only in XML mode. SeeAct does not enhance performance consistently due to the model’s reasoning limitations with multimodal input. We also compare the SoM framework and bbox-only and show SoM is better; please refer to Appendix D.2 for more detail. ReAct and SeeAct frameworks increase token usage, harming efficiency. As per Table 3, XML+ReAct settings produce an average of 67.89 tokens, while models post-Instruction Tuning averaged only 4.96 tokens.

Influence of Windows Size. As shown in Fig-

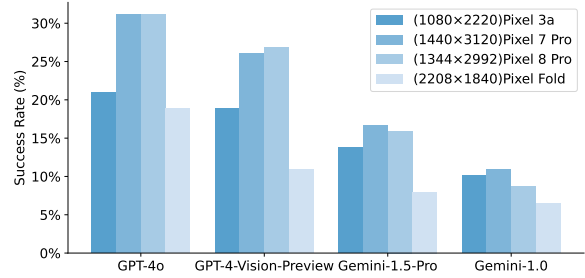


Figure 5: The performance of four models across four different device types is presented. Among these, the Pixel 3a is a smaller-sized phone, the Pixel 7 Pro and Pixel 8 Pro are of sizes comparable to commonly used phones, and the Pixel Fold is akin to a tablet.

ure 5, experiments with three Android VMs of varying sizes in SoM mode show optimal agent performance on screens matching commonly used smartphones (e.g., Pixel 7 Pro, Pixel 8 Pro). Performance drops on smaller (Pixel 3a) and larger screens (Pixel Fold) due to increased scrolling needs and landscape orientation challenges, respectively.

6 Conclusion

In this work, we introduced ANDROIDLAB, a framework tackling challenges in training and evaluating Android agents. ANDROIDLAB provides a operating environment, unified action spaces, and a benchmark of 138 tasks across nine apps. We defined a method for using the UI tree and device state to identify sub-goals, enabling our metrics to support task completion via any paths and ensuring fair and consistent comparisons. Our experiments demonstrated that fine-tuned open-source models have shown considerable improvements, while top-performing closed-source models like GPT-4o and Claude-3.5-Sonnet continue to lead in success rates and efficiency. ANDROIDLAB offers a reproducible benchmark, open datasets, and tools, advancing research in efficient and privacy-preserving mobile agents.

Acknowledgment

This work is supported by the Natural Science Foundation of China (NSFC) 62495063, 624B2084, and 62276148. The GPU compute used in this work is sponsored by Zhipu AI. The corresponding author: Yuxiao Dong (yuxiaod@tsinghua.edu.cn).

Limitations

Limited Expandability of Evaluation Tasks. All evaluation tasks in our study are predefined and hardcoded. This means that if new evaluation tasks need to be added in the future, they must be individually and manually integrated, which is a time-consuming and not easily scalable process.

Fixed Wait Time for Actions. In the action space, the model waits for a fixed period after selecting each action to allow the device to respond. However, this fixed waiting time does not account for the variability of response times across Android devices. Such variability can be attributed to several factors, including the device model, age, and user-specific configurations. Consequently, it is challenging to establish a universally applicable wait time for responses.

Lack of Cross-Platform Capability. It is important to note that our evaluation framework is limited to the Android operating system and cannot be used to evaluate models operating on other systems, such as iOS or other device platforms. This limitation renders our framework applicable solely to a single platform. Although some tools (e.g., XCUITest, WebDriverAgent) can transform iOS operations and page information into an XML-like format, we have observed that, since these tools are third-party software, the page information obtained through this transformation process is not entirely consistent with the results directly retrieved from Android devices. This discrepancy fails to meet the requirement for fairness, and the UI tree structures are also not completely aligned. Therefore, we do not plan to extend to other platforms.

Potential Risks

Risk Avoidance in Benchmark Design. In the design of our benchmark, we have avoided potentially risky operations such as payments and sending messages. Additionally, our benchmark is tested on virtual machines without an internet connection, further preventing the actual execution of these operations. However, in real-world scenarios where agents are used, special attention should be paid to the correctness of such operations when the user provides these kinds of tasks. We plan to add sensitive operation protection in future systems, meaning these operations require explicit user consent before execution.

Ensuring XML Quality for Apps. The XML quality of certain apps might be poor, possibly loading

too much or too little content. In actual deployment, it is essential to carefully inspect the XML quality of each app to ensure accurate usage.

Privacy Issues and Solutions. One major ethical concern in applying Android agents involves privacy issues. The evaluation process of models trained with user data could potentially lead to the leakage of private information. To mitigate this, we propose the Android Instruction Dataset, which is annotated by humans and ensures the removal of sensitive private information. This dataset allows models to achieve performance close to proprietary models without compromising user privacy.

Existing agent technologies often require extensive device information to function correctly, which involves transmitting private data to servers hosting these models. Our framework provides an alternative solution by enabling open-sourced models to achieve competitive performance and allowing for the private deployment of models. This eliminates the need to send data to external servers, enhancing user information security. Future work will focus on advancing on-device model training to further address privacy concerns comprehensively.

Preventing Misuse in Sensitive Applications. Another concern is the potential misuse of Android agents in sensitive applications, such as web scraping, targeted advertising, and monetary transactions. The Android Instruction Dataset we provide is generated from predefined seeds, excluding dangerous actions, to minimize misuse.

References

- Anthropic. 2023. [Introducing claude](#).
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. [Program synthesis with large language models](#).
- Jinze Bai, Shuai Bai, et al. 2023. [Qwen technical report](#).
- Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. 2021. Mobile app tasks with iterative feedback (motif): Addressing task feasibility in interactive visual environments. *arXiv preprint arXiv:2104.08560*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray,

- Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. [Evaluating large language models trained on code](#).
- Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. [Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces](#).
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*.
- Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.
- Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. [Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models](#).
- Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*.
- Wenyi Hong, Wei Han Wang, Qingsong Lv, Jiazhen Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. [Cogagent: A visual language model for gui agents](#).
- Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. 2024. Omniaact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*.
- Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.
- Hanyu Lai, Xiao Liu, Iat Long Long, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. 2024. Autowebglm: Bootstrap and reinforce a large language model-based web navigating agent. *arXiv preprint arXiv:2404.03648*.
- Juyong Lee, Taywon Min, Minyong An, Changyeon Kim, and Kimin Lee. 2024. [Benchmarking mobile device control agents across diverse configurations](#).
- Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Apibank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.
- Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. [Mapping natural language instructions to mobile UI action sequences](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8198–8210, Online. Association for Computational Linguistics.
- Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. [Reinforcement learning on web interfaces using workflow-guided exploration](#). In *International Conference on Learning Representations (ICLR)*.
- Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zeng, Zhengxiao Du, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. Webglm: Towards an efficient web-enhanced question answering system with human preferences. *arXiv preprint arXiv:2306.07906*.
- Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Long, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, et al. 2024. Visualagentbench: Towards large multimodal models as visual foundation agents. *arXiv preprint arXiv:2408.06327*.
- Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. [Gaia: a benchmark for general ai assistants](#).
- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.
- OpenAI. 2023. [Gpt-4 technical report](#).
- Yun Peng, Shuqing Li, Wenwei Gu, Yichen Li, Wenxuan Wang, Cuiyun Gao, and Michael Lyu. 2021. [Revisiting, benchmarking and exploring api recommendation: How far are we?](#)
- Divyanshu Rai, Sumbul Siddiqui, Mahesh Pawar, and Sachin Goyal. 2019. Robotic process automation: the virtual workforce. *International Journal on Future Revolution in Computer Science & Communication Engineering*, 5(2):28–32.

- Christopher Rawles, Sarah Clinckemaulle, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2024. [Androidworld: A dynamic benchmarking environment for autonomous agents](#).
- 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*.
- Mário Romao, Joao Costa, and Carlos J Costa. 2019. Robotic process automation: A case study in the banking industry. In *2019 14th Iberian Conference on information systems and technologies (CISTI)*, pages 1–6. IEEE.
- Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. 2022. [Meta-gui: Towards multi-modal conversational agents on mobile gui](#).
- Gemini Team, Machel Reid, Nikolay Savinov, and Denis Teplyashin et al. 2024. [Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context](#).
- 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*.
- Sagar Gubbi Venkatesh, Partha Talukdar, and Srin Narayanan. 2023. [Ugifi: Ui grounded instruction following](#).
- Bryan Wang, Gang Li, and Yang Li. 2023a. [Enabling conversational interaction with mobile ui using large language models](#).
- Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. 2024. Qwen2-vl: Enhancing vision-language model’s perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.
- Wei Han Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. 2023b. [Cogvlm: Visual expert for pretrained language models](#).
- Xiao Xia, Dan Zhang, Zibo Liao, Zhenyu Hou, Tianrui Sun, Jing Li, Ling Fu, and Yuxiao Dong. 2024. Scenegenerator: Precise industrial scene generation with coding agent. *arXiv preprint arXiv:2410.21909*.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. [Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments](#).
- Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 2024. Understanding the weakness of large language model agents within a complex android environment. *arXiv preprint arXiv:2402.06596*.
- An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, Zicheng Liu, and Lijuan Wang. 2023. [Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation](#).
- Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023a. [Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v](#).
- Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023b. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.
- Zhuosheng Zhan and Aston Zhang. 2023. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*.
- Dan Zhang, Tao Feng, Lilong Xue, Yuandong Wang, and Jie Tang. 2025a. Parameter-efficient fine-tuning for foundation models. *arXiv preprint arXiv:2501.13787*.
- Dan Zhang, Sining Zhou, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. 2025b. Datascope: An llm agent benchmark for data science. *arXiv preprint arXiv:2502.13897*.
- Shudan Zhang, Hanlin Zhao, Xiao Liu, Qinkai Zheng, Zehan Qi, Xiaotao Gu, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Naturalcodebench: Examining

coding performance mismatch on humaneval and natural user prompts. *arXiv preprint arXiv:2405.04520*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*.

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

A Details of Tasks

In our experiment, we use various apps to conduct various tests (succinctly presented in Table 4). The following mobile apps are chosen:

- **Bluecoins:** A personal finance management app used for tracking expenses and income.
- **Calendar:** A calendar app helps in organizing schedules and setting reminders.
- **Cantook:** An e-book reader for storing, managing, and reading e-books.
- **Clock:** A clock app for displaying the time, setting alarms, and using a stopwatch.
- **Contacts:** A contact management app for storing and organizing contact information.
- **Maps.me:** An offline map app for navigation and exploring locations.
- **PiMusic:** A music player app for organizing and playing locally stored music files.
- **Settings:** A settings app for configuring device settings and preferences.
- **Zoom:** A video conferencing app for hosting and joining online meetings.

The selection of these apps goes through multiple iterations to ensure their suitability for our evaluation purposes. A key criterion for the final selection is that each app functions independently, without requiring an internet connection or user account login. This ensures that the evaluations can be consistently replicated under the same conditions, eliminating external dependencies and reducing the risk of privacy breaches. As a result, this approach maintains the reliability and reproducibility of our results.

B Detail of Operation Modes

B.1 XML mode

As shown in Figure 6, in this mode, we prompt models with a task description, interaction history, and current compressed XML information. The models are supposed to output an action in function-call format. The actions are applied on coordinates shown in XML.

B.2 SoM mode

As shown in Figure 7, in this mode, we prompt models with a task description, interaction history, and current screenshot with a set of marks (Yang et al., 2023a). The models are also supposed to output an action in function-call format. Different from XML mode, the actions are performed on specified elements via marked indices.

B.3 ReAct mode

We follow (Yao et al., 2022b) for ReAct prompting. In this mode, we perform both text-only and multi-modal testing. The text-only and multi-modal prompts are based on Section B.1 and Section B.2 respectively. We both add prompts that allow models to think step by step before output actions.

B.4 SeeAct mode

We follow (Zheng et al., 2024) for SeeAct prompting. The raw prompts of SeeAct are designed for web browsers. To adopt that in Android environments, we make some modifications, and the final prompts are shown in Figure 8 for multi-modal testing and Figure 9 for text-only testing.

For multi-modal and text-only testing, the information on mobile phones is given by screenshots and compressed XML respectively. The models are supposed to generate detailed description of the action and its corresponding element and parameters in round 1, and the expected function-call format in round 2.

C Details of Android Instruction Dataset

This combination of autonomous and manual processes resulted in 10.5k traces and 94.3k steps, and we use 726 traces and 6208 steps derived from the Apps included in the ANDROIDLAB benchmark for training. We provide statistics of the Android Instruct dataset in Fig 10.

Table 4: List of Android Eval apps used along with corresponding example task, sub-goals, and the number of tasks.

| APP | Example Task | Sub-Goals | # tasks |
|-----------|---|---|---------|
| Bluecoins | Record an income of 8000 CNY in the books, and mark it as "salary". | <ul style="list-style-type: none"> · type: income · cash: 8000 CNY · note: salary | 15 |
| Calendar | Edit the event with title "work", change the time to be 7:00 PM. | <ul style="list-style-type: none"> · title: work · state: editing · date: today · time: 7 PM | 14 |
| Cantook | Mark Hamlet as read. | <ul style="list-style-type: none"> · book: Hamlet · state: 100% read | 12 |
| Clock | I need set an 10:30PM clock every weekend, and label it as "Watch Football Games". | <ul style="list-style-type: none"> · time: 10:30PM · frequency: every weekend · label: Watch Football Games | 27 |
| Contacts | Add a contacts whose name is Xu, set the working phone number to be 12345678, and mobile phone number to be 87654321. | <ul style="list-style-type: none"> · name: Xu · working phone number: 12345678 · mobile phone number: 87654321 | 15 |
| Maps.me | Check the driving distance and time between Bus stop of 2700 Coast Avenue and Bus Stop Route 51. | <ul style="list-style-type: none"> · driving distance: 7.0km · driving time: 8 min | 15 |
| PiMusic | Sort Pink Floyd's songs by duration time in descending order. | <ul style="list-style-type: none"> · page: ARTISTS · artist: Pink Floyd · order: descending by duration | 12 |
| Setting | Show battery percentage in status bar. | <ul style="list-style-type: none"> · battery percentage: displayed | 23 |
| Zoom | I need to join meeting 1234567890 without audio and video. | <ul style="list-style-type: none"> · meeting ID: 1234567890 · audio: off · video: off | 5 |

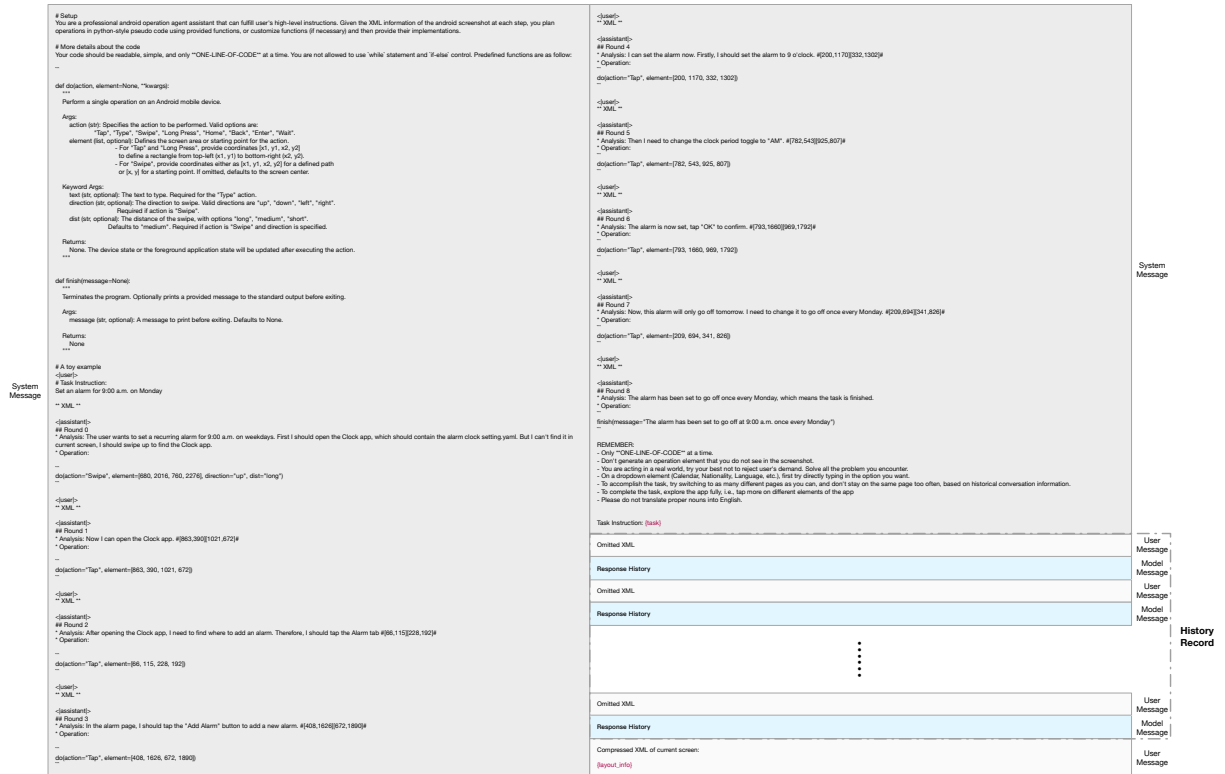


Figure 6: Prompts of XML Mode for Text-only Testing

C.1 Details of Human Annotation

In the process of constructing our data, we utilize crowdsourced annotations. To ensure that the privacy information of the annotators is not disclosed, we adopt the following measures:

1. Before the annotation begins, we explicitly inform the annotators that the annotated data will be used to fine-tune models, and part of the data will be open-sourced. Annotators who disagree may opt out of the annotation process.
2. During the annotation process, all annotated data are first stored locally by the annotators. If an annotator believes that specific data involves privacy disclosure, they may choose not to use it or skip the task.
3. After the annotation is completed, we mask and replace sensitive information such as usernames and chat logs before using the data for training. Additionally, such data will not be open-sourced.

All annotators sign formal contracts and are compensated according to reasonable standards.

C.2 Instructions Given To Annotators

We provide the instructions given to the annotators below. Note that our targets are expanded by hand-written instructions or academic datasets with available licenses.

Task Overview

For each labeling task, a target task will be given, such as: *Navigate to XXX using Amap (Gaode Map)*.

The annotator must complete the task using their phone and follow the labeling process described below to ensure it is accurately executed and recorded.

To perform this annotation task, you must install ADB (Android Device Bridge) on your computer to control the phone and install the corresponding APK. Since the task involves collecting low-level information, we will require the phone to enable multiple permissions. Still, we guarantee that the information will not be transmitted in real-time during collection. The transmitted information includes the operation details, screenshots before and after each operation, and the corresponding XML files (only containing information from the current page). You can review and decide whether to keep the annotation data. If the annotation process involves screenshots or other information that you do

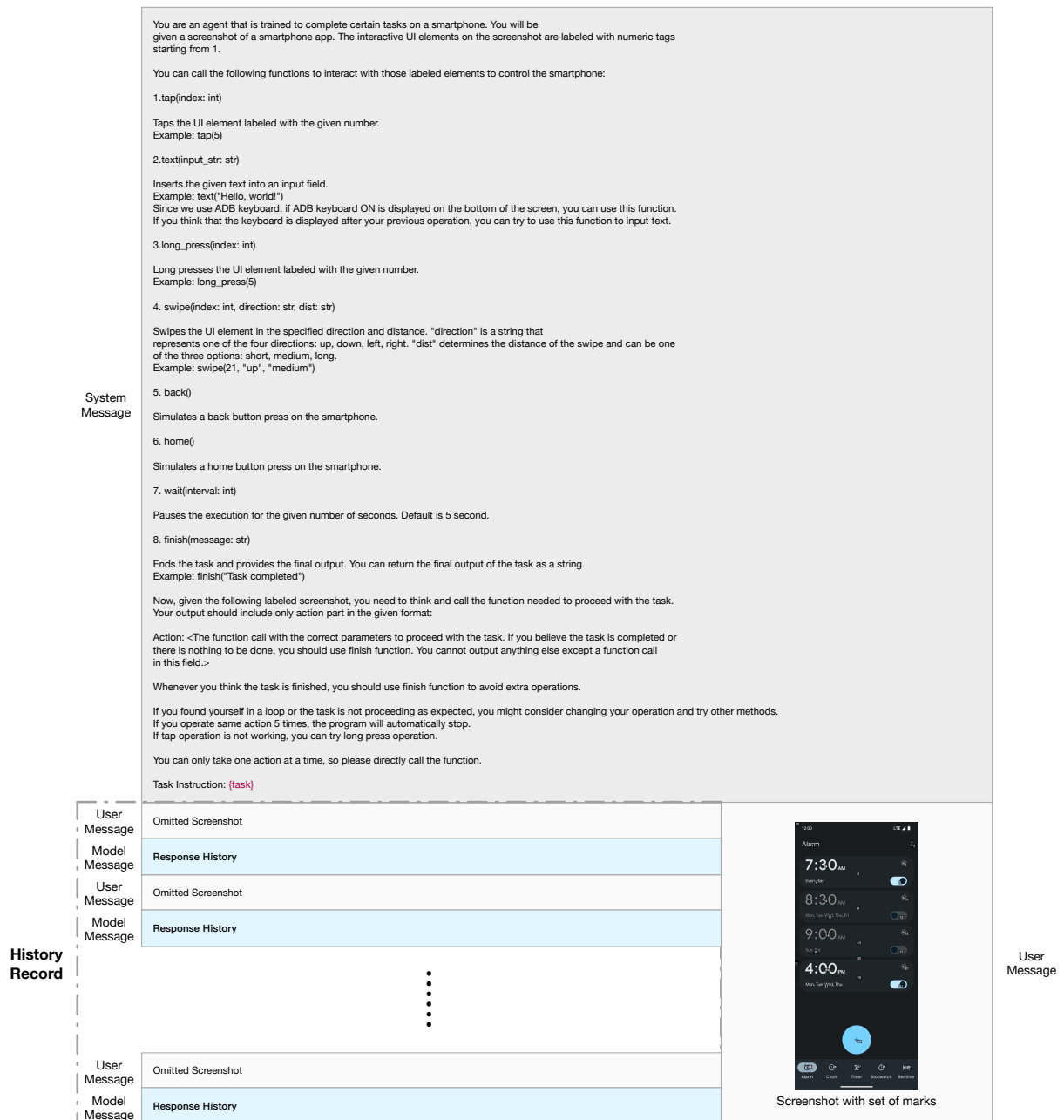


Figure 7: Prompts of SoM Mode for Multi-modal Testing

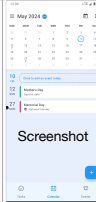
| | | | |
|---------|------------------|--|---|
| | System Message | <p>You are assisting humans doing smartphone navigation tasks step by step. At each stage, you can see the smartphone by a screenshot and know the previous actions before the current step decided by yourself that have been executed for this task through recorded history. You need to decide on the first following action to take.</p> <p>Here are the descriptions of all allowed actions: "Tap", "Type", "Swipe", "Long Press", "Home", "Back", "Enter", "Wait".</p> | |
| | User Message | <p>You are asked to complete the following task: {task}</p> <p>Previous Actions:</p> <p>{previous_actions}</p> <p>The screenshot below shows the smartphone you see. Think step by step before outlining the next action step at the current stage. Clearly outline which element in the smartphone users will operate with as the first next target element, its detailed location, and the corresponding operation.</p> <p>To be successful, it is important to follow the following rules:</p> <ol style="list-style-type: none">1. You should only issue a valid action given the current observation.2. You should only issue one action at a time.3. Terminate when you deem the task complete. |  |
| Round 1 | Model Generation | Action Generation | |
| | User Message | <p>(Reiteration) First, reiterate your next target element, its detailed location, and the corresponding operation.</p> <p>(Final Answer) Below is a multi-choice question, where the choices are elements in the smartphone. From the screenshot, find out where and what each one is on the smartphone, taking into account both their text content and path details. Then, determine whether one matches your target element if your action involves an element. Choose the best matching one.</p> <p>{option_prompt}</p> <p>Conclude your answer using the format below. Ensure your answer is strictly adhering to the format provided below.</p> <p>Predefined functions are as follow:</p> <pre>... def do(action, element=None, **kwargs): """ Perform a single operation on an Android mobile device. Args: action (str): Specifies the action to be performed. Valid options are: "Tap", "Type", "Swipe", "Long Press", "Home", "Back", "Enter", "Wait". element (list, optional): Defines the screen area or starting point for the action. - For "Tap" and "Long Press", provide coordinates [x1, y1, x2, y2] to define a rectangle from top-left (x1, y1) to bottom-right (x2, y2). - For "Swipe", provide coordinates either as [x1, y1, x2, y2] for a defined path or [x, y] for a starting point. If omitted, defaults to the screen center. Keyword Args: text (str, optional): The text to type. Required for the "Type" action. direction (str, optional): The direction to swipe. Valid directions are "up", "down", "left", "right". Required if action is "Swipe". dist (str, optional): The distance of the swipe, with options "long", "medium", "short". Defaults to "medium". Required if action is "Swipe" and direction is specified. Returns: None. The device state or the foreground application state will be updated after executing the action. """ ... def finish(message=None): """ Terminates the program. Optionally prints a provided message to the standard output before exiting. Args: message (str, optional): A message to print before exiting. Defaults to None. Returns: None """ </pre> <p>Your code should be readable, simple, and only "ONE-LINE-OF-CODE" at a time. You are not allowed to use 'while' statement and 'if-else' control. Please do not leave any explanation in your answers of the final standardized format part, and this final part should be clear and certain.</p> <p>Example if you want to swipe up from an element located at [680,2016][760,2278] with a long distance:</p> <pre>do(action="Swipe", element=[680, 2016, 760, 2278], direction="up", dist="long") ...</pre> <p>Example if you deem the task complete and want to finish with a message:</p> <pre>finish(message="The alarm on 9:00 AM weekday has been set") ...</pre> | |
| Round 2 | Model Generation | Action Grounding | |

Figure 8: SeeAct Prompts for Multi-modal Testing

| | | |
|---------|------------------|--|
| | System Message | <p>You are assisting humans doing smartphone navigation tasks step by step. At each stage, you can see the smartphone by compressed layout information and know the previous actions before the current step decided by yourself that have been executed for this task through recorded history. You need to decide on the first following action to take.</p> <p>Here are the descriptions of all allowed actions: "Tap", "Type", "Swipe", "Long Press", "Home", "Back", "Enter", "Wait".</p> |
| | User Message | <p>You are asked to complete the following task: {task}</p> <p>Previous Actions:</p> <p>{previous_actions}</p> <p>The compressed layout information below shows the smartphone you see.</p> <p>{layout_info}</p> <p>Think step by step before outlining the next action step at the current stage. Clearly outline which element in the smartphone users will operate with as the first next target element, its detailed location, and the corresponding operation.</p> <p>To be successful, it is important to follow the following rules:</p> <ol style="list-style-type: none"> 1. You should only issue a valid action given the current observation. 2. You should only issue one action at a time. 3. Terminate when you deem the task complete. |
| Round 1 | Model Generation | <p>Action Generation</p> <p>(Reiteration) First, reiterate your next target element, its detailed location, and the corresponding operation.</p> <p>(Final Answer) Below is a multi-choice question, where the choices are elements in the smartphone. From compressed layout information, find out where and what each one is on the smartphone, taking into account both their text content and path details. Then, determine whether one matches your target element if your action involves an element. Choose the best matching one.</p> <p>{option_prompt}</p> <p>Conclude your answer using the format below. Ensure your answer is strictly adhering to the format provided below.</p> <p>Predefined functions are as follow:</p> <pre> ... def do(action, element=None, **kwargs): """ Perform a single operation on an Android mobile device. Args: action (str): Specifies the action to be performed. Valid options are: "Tap", "Type", "Swipe", "Long Press", "Home", "Back", "Enter", "Wait". element (list, optional): Defines the screen area or starting point for the action. - For "Tap" and "Long Press", provide coordinates [x1, y1, x2, y2] to define a rectangle from top-left (x1, y1) to bottom-right (x2, y2). - For "Swipe", provide coordinates either as [x1, y1, x2, y2] for a defined path or [x, y] for a starting point. If omitted, defaults to the screen center. Keyword Args: text (str, optional): The text to type. Required for the "Type" action. direction (str, optional): The direction to swipe. Valid directions are "up", "down", "left", "right". Required if action is "Swipe". dist (str, optional): The distance of the swipe, with options "long", "medium", "short". Defaults to "medium". Required if action is "Swipe" and direction is specified. Returns: None. The device state or the foreground application state will be updated after executing the action. """ ... def finish(message=None): """ Terminates the program. Optionally prints a provided message to the standard output before exiting. Args: message (str, optional): A message to print before exiting. Defaults to None. Returns: None """ </pre> <p>Your code should be readable, simple, and only "ONE-LINE-OF-CODE" at a time. You are not allowed to use 'while' statement and 'if-else' control. Please do not leave any explanation in your answers of the final standardized format part, and this final part should be clear and certain.</p> <p>Example if you want to swipe up from an element located at [680,2016][760,2276] with a long distance:</p> <pre>do(action="Swipe", element=[680, 2016, 760, 2276], direction="up", dist="long")</pre> <p>Example if you deem the task complete and want to finish with a message:</p> <pre>finish(message="The alarm on 9:00 AM weekday has been set")</pre> |
| | User Message | |
| Round 2 | Model Generation | <p>Action Grounding</p> |

Figure 9: SeeAct Prompts for Text-only Testing

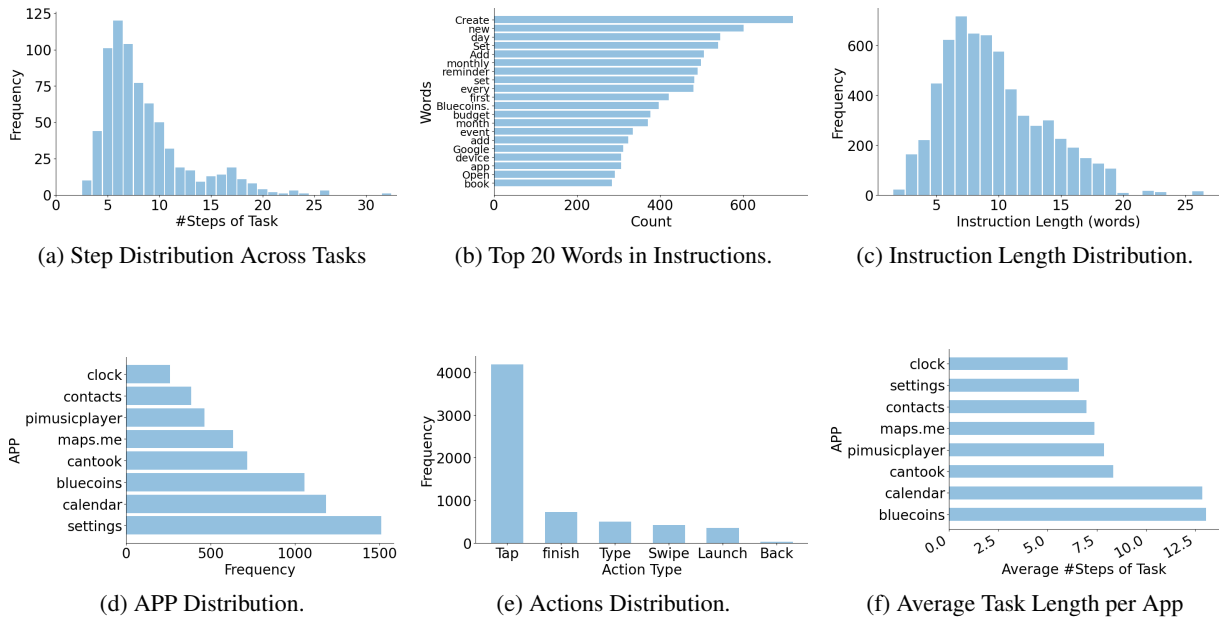


Figure 10: Statistics for Android Instruct dataset. We collect 726 traces and 6208 steps across Apps in ANDROIDLAB benchmark.

not want to be used for training, you can:

1. Skip the screenshot or specify that parts of the screenshot be hidden.
2. Skip the entire target task.
3. Skip all tasks involving the currently annotated app.

Your data will not be used for purposes other than training the model.

After completing the annotation, you must upload all the tasks you were responsible for in one go. We have designed a plugin to store all the content in a unified folder.

A complete annotation consists of multiple operations called a sequence (trace). Each single-step operation is recorded once, and the definition of a single-step operation is detailed in the annotation documentation.

Please follow the steps below for plugin usage to install the annotation plugin.

Plugin Usage Instructions

Installing ADB and Connecting Phone to Computer

For your Android phone, you need to perform the following settings:

1. Connect the phone to the computer via a USB cable.

2. Ensure that the **Developer Options** and **USB Debugging Mode** are enabled on the Android phone:

- Go to *Settings - Developer Options - Android Debugging*. Check the box for *Allow USB debugging*. If unavailable, go to *Settings - System Updates - Developer Options - USB Debugging*.
- If you can't find the developer options, go to *Settings - About Phone* and tap the *Build Number* seven times.
- If these methods don't work, search for how to enable developer options and USB debugging specific to your phone model.
- If you still encounter issues, seek help in the group chat.

3. Reconnect the phone to the computer, and on the phone, click *Allow file transfer/USB debugging/higher permissions*. Also, allow the connection on the computer (if prompted).
4. After entering Developer Mode, turn off the following animations under *Developer Options* to increase the success rate of retrieving XML information via ADB commands:

- Window Animation Scale.
- Transition Animation Scale.
- Animator Duration Scale.

Follow the steps above until the following result is displayed using the command *adb devices*:

```
adb devices
List of devices attached
1a0d5d59 device
```

The number before *device* is randomly generated. You should see only one device. If there is more than one, try disconnecting other devices or closing virtual machines.

Installing ADB Keyboard

Download the ADB Keyboard APK.

Run: `adb install <APK full path>`

Enable permissions on the phone and agree to the installation.

Once the installation is complete, set ADB Keyboard as the default input method in the phone settings. You can try the following two lines of code:

```
ime enable com.android.adbkeyboard/.AdbIME
ime set com.android.adbkeyboard/.AdbIME
```

If successful, when you open any text box, you'll see the message *ADB Keyboard ON* at the bottom of the screen. If unsuccessful, manually change the input method in the settings.

Running Test Script

1. Open the command line, run *adb devices*, and ensure correct output.
2. Run the following commands in adb shell:

```
input keyevent KEYCODE_BACK
input keyevent KEYCODE_HOME
input keyevent KEYCODE_ENTER
```

If there's no error or response, it's fine. If you see *Command execution failed*, ensure you're using the correct method sequence, not *Press xxx* commands like *adb shell input keyevent KEYCODE_A*.

3. Open any text input field and run the following commands in adb shell:

```
input keyevent KEYCODE_A
```

The setup succeeds if the letter "a" appears on the screen.

Annotation Plugin Usage Instructions

You can perform the following operations on the phone. After completing any one of these operations, do not proceed until the command line shows *Operation completed*. If the phone has not responded yet (such as loading a new page), wait until the page is fully loaded before clicking the next *Begin*.

1. **Click or Swipe:** Perform this directly on the phone. Click slowly, holding for 0.2 to 0.5 seconds.
2. **Text Input:** If the ADB Keyboard was successfully installed, you can input text. Before entering text, click on the text box in the previous step and ensure that the *ADB Keyboard ON* symbol appears at the bottom of the screen. Click the *Type* button on the GUI interface, enter the desired text in the computer's input box (Chinese/English), then click *OK*. You will observe the input on the phone, and the command line will display *Simulating typing xxx*.
3. **Press xxx:** Three preset buttons are defined: *Press Home* (Home key), *Press Back* (Back key), and *Press Enter* (keyboard Enter key). The command line will show *Simulating press xxx*.
4. **Finish Task:** If you believe the task is complete, click the *Finish* button on the GUI. If the task requires an answer, fill in the response in the popup text box. If not, click *OK*.

After finishing a task, you can close the command line and GUI windows. If there are no issues with the annotation, you can return to Step 2 to start the next annotation. Otherwise, follow these steps:

1. The command line will output the *Save Path*, which contains all saved information for the annotation. You may delete the folder if you believe an error occurred or sensitive information was recorded.
2. Each task has a prefix consisting of the first 32 characters of the task name. Ensure that the final submission includes one and only one instance of each non-skipped task.
3. If certain operations were recorded incorrectly without affecting the phone's state, you may

delete those steps. The step sequence is stored in *Save Path/traces/trace.jsonl*. Record the steps you need to delete.

4. If a screen contains sensitive information that can be removed while still being used for training, record the steps and describe the sensitive information in detail.

Summary of Key Points

1. Always use *adb devices* before starting the annotation to ensure a successful connection.
2. Reopen the *app_for_XXX/dist/label(.exe)* for each annotation instruction.
3. The storage path must not contain Chinese characters.
4. Click *Begin* before each operation and wait for the message *Begin your operation...* to appear before proceeding. If you proceed without waiting, the operation will be invalid. If the state cannot be recovered, you must restart the task. Make sure to click *Begin* before finishing as well.
5. After each operation is completed, wait until the corresponding success message appears in the command line and you see the output *Operation completed* before clicking *Begin* for the next action. Failure to follow these two key rules may result in invalid data. It's better to proceed slowly and carefully than rush and make mistakes.

C.3 Reward Model

Our approach involves the following steps. Firstly, in the Task Derivation and Expansion phase, we generate the task content. Then, we have the existing large models perform the tasks on Android devices using the operation framework defined by ANDROIDLAB.

In the initial rounds, the criteria for determining task completion are based on direct observation of the operation sequence by annotators, who judge whether the task has been completed. After accumulating a sufficient number of annotated trajectories, we construct a reward model to assist in the judgment through the following method:

We concatenate all operation screenshots and overlay the specific actions on the images (e.g., click positions, swipe paths, and input contents).

The model is then required to output the task completion status. We utilize approximately 500 manually annotated trajectories (including both positive and negative samples) to train this reward model. Our observations indicated that the reward model trained using this method achieved an accuracy rate of over 98% in determining task completion.

D Additional Results

D.1 Detail results across different APPs

Table 5 shows the number of tasks correctly completed by various models across different apps, without employing the ReAct and SeeAct frameworks. This table shows that GPT-4o and GPT-4-1106-Preview perform relatively well, completing 78 and 79 tasks, respectively. In the XML mode, GPT-4-1106-Preview stands out as the top performer with 43 tasks completed. Comparatively, in the SoM mode, GPT-4o excels, completing a significantly higher number of tasks than the other models. Most models exhibit high success rates in tasks like "Contacts" and "Setting". Overall, GPT-4o and GPT-4-1106-Preview outperform the other models significantly in both XML and SoM modes, while Gemini-1.5-Pro shows a reasonable number of task completions across various apps.

Table 6 shows the performance improvements observed after implementing the ReAct and SeeAct frameworks on different models across various apps. Notably, GPT-4o shows significant enhancement, with the number of completed tasks increasing from 35 to 46 in XML+ReAct mode and from 43 to 44 in SoM+ReAct mode. Gemini-1.5-Pro also benefits, increasing from 26 to 43 tasks. The improvements are evident in specific apps like "Bluecoins" and are especially notable in high-complexity, multi-step tasks. GPT-4o leads in performance across all frameworks, showing how ReAct and SeeAct improve the model.

D.2 Detail results across different multi-modal training mode

We compare different multimodal training modes in Table 7. Under the same training data and base model settings, BBOX mode removes specified sets-of-masks from the screen. It is worth mentioning that datasets like AITW only provide click positions rather than bounding boxes (BBOX) and do not offer a way to reconstruct the click-box from XML. Therefore, theoretically, data from AITW and similar datasets are more challenging to learn

Table 5: The number of tasks completed by all models across all apps in different modes.

| Mode | Model | Bluecoins 15 | Calendar 14 | Cantook 12 | Clock 27 | Contacts 15 | Maps.me 15 | PiMusic 12 | Setting 23 | Zoom 5 | Total 138 |
|------|------------------------------|-----------------|----------------|---------------|-------------|----------------|---------------|---------------|---------------|-----------|--------------|
| XML | GPT-4o | 1 | 0 | 3 | 8 | 5 | 5 | 2 | 10 | 1 | 35 |
| | GPT-4-1106-Preview | 1 | 4 | 6 | 4 | 6 | 6 | 4 | 9 | 3 | 43 |
| | Gemini-1.5-Pro | 1 | 1 | 3 | 6 | 3 | 4 | 3 | 4 | 1 | 26 |
| | Gemini-1.0 | 0 | 1 | 1 | 4 | 2 | 0 | 1 | 2 | 1 | 12 |
| | GLM4-PLUS | 2 | 0 | 4 | 9 | 6 | 3 | 2 | 10 | 2 | 38 |
| | LLaMA3.1-8B-Instruct | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 3 |
| | Qwen2.5-7B-Instruct | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 2 | 0 | 6 |
| | GLM4-9B-Chat | 0 | 1 | 0 | 2 | 1 | 1 | 0 | 3 | 2 | 10 |
| | LLaMA3.1-8B-ft | 3 | 1 | 6 | 7 | 6 | 5 | 0 | 4 | 1 | 33 |
| | Qwen2.5-7B-ft | 1 | 1 | 3 | 4 | 7 | 4 | 1 | 6 | 0 | 27 |
| SoM | GLM4-9B-ft | 0 | 1 | 5 | 7 | 5 | 2 | 0 | 8 | 1 | 29 |
| | GPT-4o | 1 | 1 | 5 | 7 | 8 | 2 | 2 | 13 | 4 | 43 |
| | GPT-4-Vision-Preview | 1 | 1 | 5 | 8 | 6 | 2 | 2 | 8 | 3 | 36 |
| | Gemini-1.5-Pro | 0 | 0 | 5 | 2 | 5 | 0 | 1 | 7 | 3 | 23 |
| | Gemini-1.0 | 0 | 0 | 2 | 3 | 3 | 0 | 1 | 5 | 1 | 15 |
| | Claude-3.5-Sonnet | 4 | 2 | 4 | 9 | 7 | 0 | 3 | 10 | 1 | 40 |
| | Claude-3-Opus | 1 | 0 | 1 | 2 | 4 | 0 | 3 | 7 | 0 | 18 |
| | CogVLM2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| | LLaMA3.2-11B-Vision-Instruct | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 |
| | Qwen2-VL-7B-Instruct | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 1 | 1 | 5 |
| | CogVLM2-ft | 0 | 0 | 2 | 3 | 4 | 1 | 1 | 4 | 1 | 16 |
| | LLaMA3.2-11B-Vision-ft | 1 | 1 | 1 | 3 | 0 | 6 | 0 | 2 | 0 | 14 |
| | Qwen2-VL-7B-Instruct-ft | 1 | 0 | 1 | 4 | 5 | 3 | 2 | 7 | 2 | 25 |

Table 6: The improvement in model performance after employing the ReAct and SeeAct frameworks, is reflected in the increased number of successfully completed tasks across various apps.

| Mode | Model | Bluecoins 15 | Calendar 14 | Cantook 12 | Clock 27 | Contacts 15 | Maps.me 15 | PiMusic 12 | Settings 23 | Zoom 5 | Total 138 |
|------------|----------------|-----------------|----------------|---------------|-------------|----------------|---------------|---------------|----------------|-----------|--------------|
| XML | GPT-4o | 1 | 0 | 3 | 8 | 5 | 5 | 2 | 10 | 1 | 35 |
| | Gemini-1.5-Pro | 1 | 1 | 3 | 6 | 3 | 4 | 3 | 4 | 1 | 26 |
| XML+ReAct | GPT-4o | 2 | 0 | 4 | 12 | 7 | 6 | 2 | 11 | 2 | 46 |
| | Gemini-1.5-Pro | 4 | 0 | 4 | 6 | 6 | 6 | 3 | 11 | 3 | 43 |
| XML+SeeAct | GPT-4o | 1 | 2 | 4 | 8 | 5 | 3 | 2 | 7 | 2 | 34 |
| | Gemini-1.5-Pro | 1 | 0 | 6 | 6 | 5 | 0 | 2 | 8 | 1 | 29 |
| SoM | GPT-4o | 1 | 1 | 5 | 7 | 8 | 2 | 2 | 13 | 4 | 43 |
| | Gemini-1.5-Pro | 0 | 0 | 5 | 2 | 5 | 0 | 1 | 7 | 3 | 23 |
| SoM+ReAct | GPT-4o | 3 | 1 | 5 | 7 | 7 | 3 | 0 | 15 | 3 | 44 |
| | Gemini-1.5-Pro | 1 | 1 | 3 | 2 | 4 | 1 | 2 | 7 | 1 | 22 |
| SoM+SeeAct | GPT-4o | 6 | 1 | 4 | 11 | 6 | 0 | 2 | 9 | 3 | 42 |
| | Gemini-1.5-Pro | 1 | 0 | 6 | 6 | 5 | 0 | 2 | 8 | 1 | 29 |

from.

D.3 Detail results of SeeAct and ReAct methods

We have provided detailed results on the impact of the SeeAct and ReAct frameworks on model performance in Fig 8, including all four metrics.

Table 7: Different multi-modal modes of instruction tuning. We use the same set of training data but only add a set-of-mask index on SoM mode. Note that AITW dataset even could not provide accurate bbox, but only point. We use CogVLM2 as base model.

| Operation Mode | SR | Sub-SR | RRR | ROR |
|-----------------------|-----------|---------------|------------|------------|
| BBOX | 5.79 | 6.03 | 47.95 | 55.05 |
| SoM | 11.59 | 16.06 | 57.37 | 85.58 |

Table 8: The impact of the ReAct and SeeAct frameworks. Notably, model performance is significantly improved in XML+ReAct mode.

| Mode | Model | SR | Sub-SR | RRR | ROR |
|-------------|----------------|-----------|---------------|------------|------------|
| XML | GPT-4o | 25.36 | 30.56 | 107.45 | 86.56 |
| | Gemini-1.5-Pro | 18.84 | 22.40 | 57.72 | 83.99 |
| XML+ReAct | GPT-4o | 33.33 | 38.22 | 97.93 | 90.74 |
| | Gemini-1.5-Pro | 31.16 | 34.54 | 92.08 | 90.31 |
| XML+SeeAct | GPT-4o | 24.64 | 27.31 | 93.78 | 79.62 |
| | Gemini-1.5-Pro | 21.01 | 25.53 | 75.97 | 89.06 |
| SoM | GPT-4o | 31.16 | 35.02 | 87.32 | 85.36 |
| | Gemini-1.5-Pro | 16.67 | 18.48 | 105.95 | 91.52 |
| SoM+ReAct | GPT-4o | 31.88 | 39.19 | 104.69 | 89.80 |
| | Gemini-1.5-Pro | 15.94 | 21.38 | 109.81 | 84.16 |
| SoM+SeeAct | GPT-4o | 30.43 | 36.24 | 97.45 | 88.56 |
| | Gemini-1.5-Pro | 21.01 | 25.53 | 75.97 | 89.06 |