Understanding the Weakness of Large Language Model Agents within a Complex Android Environment

Mingzhe Xing*
mzxing@stu.pku.edu.cn
Peking University
Beijing, China

Rongkai Zhang
rkzhang@stu.pku.edu.cn
Peking University
Beijing, China

Hui Xue
xuehui@microsoft.com
Microsoft Research
Beijing, China

Qi Chen
cheqi@microsoft.com
Microsoft Research
Beijing, China

Fan Yang
fanyang@microsoft.com
Microsoft Research
Beijing, China

Zhen Xiao†
xiaozhen@pku.edu.cn
Peking University
Beijing, China

ABSTRACT

Large language models (LLMs) have empowered intelligent agents to execute intricate tasks within *domain-specific software* such as browsers and games. However, when applied to *general-purpose software systems* like operating systems, LLM agents face three primary challenges. Firstly, the *action space is vast and dynamic*, posing difficulties for LLM agents to maintain an up-to-date understanding and deliver accurate responses. Secondly, real-world tasks often require *inter-application cooperation*, demanding farsighted planning from LLM agents. Thirdly, agents need to identify optimal solutions *aligning with user constraints*, such as security concerns and preferences. These challenges motivate `AndroidArena`, an environment and benchmark designed to evaluate LLM agents on a modern operating system. To address high-cost of manpower, we design a scalable and semi-automated method to construct the benchmark. In the task evaluation, `AndroidArena` incorporates accurate and adaptive metrics to address the issue of non-unique solutions. Our findings reveal that even state-of-the-art LLM agents struggle in cross-APP scenarios and adhering to specific constraints. Additionally, we identify a lack of four key capabilities, *i.e.,* understanding, reasoning, exploration, and reflection, as primary reasons for the failure of LLM agents. Furthermore, we provide empirical analysis on the failure of reflection, and improve the success rate by 27% with our proposed exploration strategy. This work is the first to present valuable insights in understanding fine-grained weakness of LLM agents, and offers a path forward for future research in this area. Environment, benchmark, prompt, and evaluation code for `AndroidArena` are released at https://github.com/AndroidArenaAgent/AndroidArena.

CCS CONCEPTS

• **Computing methodologies → Planning and scheduling**; **Robotic planning**; • **Human-centered computing →** *Human computer interaction (HCI).*

KEYWORDS

Large Language Model; AI Agent; Task Planning

ACM Reference Format:
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. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671650

1 INTRODUCTION

Large language models (LLMs) have shown great potentials in understanding hidden intent from human and commonsense reasoning [28]. This makes it possible to utilize *LLM as agent* [30, 33], an intelligent entity capable of making decisions and executing actions based upon the perceived state of environment. An example is for LLMs to interact with *domain-specific software*, such as databases [13], games [29] and browsers [44], for task completion.

More recently, new LLM-based agents have emerged to interact with *general-purpose software systems*, such as operating systems along with their installed APPs, to accomplish more complex *open-domain tasks* [38, 40]. These tasks range from simple actions like setting reminders to more intricate activities like financial management and staying connected with loved ones. Complex scenarios in operating systems typically manifest the following characteristics: 1) a **vast and ever-changing action space** due to real-time internet data exchange, APP installations, and upgrades; 2) an increasing demand for **cross-APP collaboration** as user tasks become more interconnected and multifaceted; and 3) heightened consideration for **personal interests and security concerns**.

These characteristics motivate us to establish a new environment and comprehensive benchmark to study the boundaries of LLM agent's capability within a complex software system. In this paper, we introduce `AndroidArena`, an environment built on the Android operating system, accompanied by an evaluation benchmark containing annotated ground truth action sequences. `AndroidArena`

*This work is done during the internship at Microsoft Research.
†Corresponding author.

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671650

supports real-time internet data exchange and dynamic APP management, and enables seamless operations across various APPs. These features facilitate the evaluation of LLM agents in a *vast and dynamic action space* and *cross-APP scenarios*. Additionally, we propose a scalable method for semi-automatically constructing an instruction benchmark, ensuring comprehensive coverage of APP functionalities. Our open-source benchmark, informed by the aforementioned characteristics, evaluates tasks not only within a single APP but also complex tasks requiring collaboration across multiple APPs. It further considers tasks subject to constraints such as *user preferences and security considerations*.

Evaluating tasks within a complex operating system is non-trivial [15], primarily due to the fact that the feasible action sequence for a task is often non-unique. This presents a significant challenge to precisely evaluating agents in multi-step decision-making scenarios. To address this issue, we devise **adaptive metrics to evaluate task completion accurately**. The evaluation results reveal that all state-of-the-art (SOTA) LLM agents fall short in cross-APP scenarios, with a success rate of less than 60%, and struggle to fully adhere to specific constraints. Notably, GPT-3.5 [19] achieves a 6x higher success rate than LLaMA2-70B [26]. Through meticulous case analysis to understand the causes of failure, we identify and abstract **four key planning capabilities of LLM agents**, inspired by reinforcement learning (RL) [14, 25, 34, 35]: **understanding**, **reasoning**, **exploration**, and **reflection**. We design metrics to **measure these fine-grained capabilities, showing improvement directions for LLM agents**. LLaMA2 exhibits weaknesses across all dimensions, and even advanced models like GPT-4 [19] are no exemptions, exhibiting weak reflection and exploration abilities. Empirical analysis predominantly attributes the weakness in reflection to low-quality trajectories and sparsity in environment feedback. Moreover, we find that by integrating historical visited information into the prompt and balancing exploration and exploitation by the agent, the success rate of specific APPs can improve by 27%, and the exploration performance is enhanced.

In summary, we make the following contributions.

- We open-source AndroidArena, a benchmark based on the Android operating system, to evaluate daily tasks requiring cross-APP collaboration, as well as considerations for constraints such as security. Additionally, our scalable and semi-automated approach reduces the cost of benchmark construction.
- Our findings indicate that STOA models underperform in daily tasks and are not ready for direct product integration. We propose fine-grained metrics that reveal failure causes and highlight four areas for future research: understanding, reasoning, reflection, and exploration. Initial analysis show the failure reasons of reflection, and 27% of improvement when enhancing exploration.

2 BACKGROUND

2.1 Frameworks of LLM Agent

With the emergence of LLMs, the study of LLM agents has begun to thrive. Early research work [1, 11, 20] prompt LLMs to directly generate actions based on environment observations. ReAct [42] is a pioneer work to integrate reasoning and acting in LLM for general task solving. It first generates reasoning traces based on history context, subsequently producing actions to interact with the environment. Building upon this task-solving paradigm, subsequent agents have been proposed to enhance capabilities in various dimensions. Reflexion [23] summarizes textual feedback from the environment and then incorporates it as additional context for the LLM agent in the subsequent episode. The self-reflective context acts as a semantic gradient signal, offering the agent a concrete direction to improve upon, and facilitates the learning process from prior mistakes for enhancing task performance. This paper focuses on evaluating the abilities of LLM agents and understanding their weaknesses. We adopt ReAct as the basic agent strategy and Reflexion as an approach to assess the agent's ability of self-reflection.

2.2 Existing Operating System Task Benchmark

Operating Systems (OS) serve as crucial environments with which humans interact daily, and numerous benchmarks have emerged to evaluate the performances of agents within OS. AITW [21] stands out as a static image dataset that offers human demonstrations of device interactions. However, the static nature of AITW prevents agents from obtaining a reproducible environment. On the other hand, AndroidEnv [27] provides support for dynamic interactions with APP. Despite this, it only supports single APP's interaction in each environment instance, which limits its capability to evaluate complex and realistic tasks. WebArena [44] creates tasks simulating human behavior on web browsers. However, it is also limited to automate tasks on a single website, which poses a constraint on its applicability. While these works have been a source of inspiration, they also highlight the significant challenges of evaluating tasks on OS. Tasks performed by real-world users are often more complex and demanding, requiring the collaboration of multiple APPs. Additionally, agents need to consider various constraints such as security and user preference. Therefore we propose AndroidArena, a reproducible mobile environment that allows for cross-APP access. Alongside this, we introduce a new dataset that encapsulates the richness, difficulty, and constraints of instructions. The detail comparison with other works is listed in Table 1.

2.3 Existing Metrics for Multi-step Decision

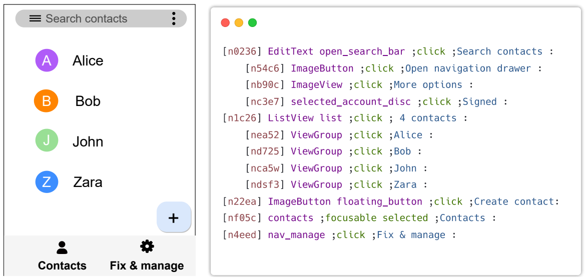
To assess the performances of LLM agents, a range of metrics are proposed. SmartPlay [32] and LASER [17] employ success rate and reward to evaluate task completion. However, these metrics cannot reflect the detailed completion within individual tasks. TPTU [22] and Mind2Web [7] incorporate a step-wise action alignment method, offering a more nuanced analysis of task completion. Nevertheless, when applied to multi-step decision-making scenarios where the feasible action sequence for completing a task is not unique, this kind of step-wise matching method may introduce inaccuracies. In this paper, we propose an adaptive way to accurately assess the task completion, and a set of fine-grained ability evaluation metrics to understand the weaknesses of agents, providing valuable insights into improvement directions for LLM agents.

3 ANDROIDARENA ENVIRONMENT

In this section, we introduce the AndroidArena environment, distinguished by its vast and dynamic action space, along with its

Table 1: The comparison between our `AndroidArena` benchmark and existing benchmarks.

Benchmark	Online Evaluation	Realistic Environment	Scalably Generated	Collaborative Tasks between APPs	Tasks with Constraints
MineCraft [29]	✓	✗	✗	✗	✗
Mind2Web [7]	✗	✓	✗	✗	✗
AITW [21]	✗	✓	✗	✗	✗
AndroidEnv [27]	✓	✓	✗	✗	✗
WebArena [44]	✓	✓	✗	✗	✗
`AndroidArena`	✓	✓	✓	✓	✓



```
[n0236] EditText open_search_bar ;click ;Search contacts :
    [n54c6] ImageButton ;click ;Open navigation drawer :
    [nb90c] ImageView ;click ;More options :
    [nc3e7] selected_account_disc ;click ;Signed :
[n1c26] ListView list ;click ; 4 contacts :
    [nea52] ViewGroup ;click ;Alice :
    [nd725] ViewGroup ;click ;Bob :
    [nca5w] ViewGroup ;click ;John :
    [ndsf3] ViewGroup ;click ;Zara :
[n22ea] ImageButton floating_button ;click ;Create contact:
[nf05c] contacts ;focusable selected ;Contacts :
[n4eed] nav_manage ;click ;Fix & manage :
```

(a) APP screenshot.

(b) Compressed observation.

Figure 1: An example of the Contacts APP page and its corresponding compressed observation.

capability to facilitate cross-APP and constrained task execution. We begin by offering a formal definition of the mobile task automation process, followed by an overview of the system implementation. Subsequently, we explore the intricacies of the action space, highlighting its dynamic and expansive nature.

3.1 LLM Agent for Mobile Task Automation

Given a task presented with a user instruction in natural language, the agent is responsible for making action decisions to complete this instruction on the phone. This process can be formulated as a Contextual Markov Decision Process (CMDP) [9] $\langle C, \mathcal{S}, \mathcal{A}, \mathcal{T}, r \rangle$. Context $c \in C$ is the mobile task explicitly expressed as a textual instruction. State $s \in \mathcal{S}$ is the current observed phone state, *i.e.,* the displayed content on the screen. Action $a \in \mathcal{A}$ can be performed on the current phone screen, *e.g.,* clicks or typing. Transition function $\mathcal{T}(s'|s, a)$ represents the change in the phone on performing an action. Reward r is awarded for successful completion of the task.

Implementation. Our implementation is based on UIAutomator [8], a UI testing framework that enables direct operations on UI components. With UIAutomator, we offer flexible configurations to render APP page content (*i.e.,* the observation space) in two modes: 1) the phone screenshot, a pixel-based representation as perceived by humans, and 2) the textual XML description of the phone screen (depicted in Fig. 8 in §A). It is important to note that, given the focus of this work on LLM agents, we exclusively utilize the text modality, while acknowledging that our implementation is capable of supporting multi-modal models. Each UI component in the screen corresponds to an XML entry, containing its role (*e.g.,* a button), text content, and properties (*e.g.,* if clickable) information. A statistic conducted on eight popular APPs indicates an average token count for the XML exceeding 10,000. Consequently, directly feeding the XML into the LLM is impractical due to context length limitations. To address this challenge, we propose a two-stage heuristic compression method, involving the removal of decision-irrelevant XML tags and the merging of non-visible or non-functional nodes (the detailed algorithm is provided in §A) to compress the XML. As illustrated in Fig. 1(b), the compressed observation maintains the hierarchical structure of the original XML, enabling the LLM agent to comprehend the UI layout via text. Subsequent to compression, each entry is assigned a unique ID (*e.g.,*[nd725]), facilitating agents in locating the UI element. Our proposed method achieves a compression ratio of 86.6% across several tested APPs (please see Table 6 in §A). Motivated by previous research [12] showing superior performance by regarding LLM as reward functions, we employ GPT-4 to quantify the reward r, and validate its effectiveness through experiments in §6.2.

3.2 Vast and Dynamic Action Space

Unlike prior environments [27, 44] focusing on a single APP and only supporting specific actions, our action space is vast and dynamic. It is attributed to the fact that a typical APP may feature hundreds of UI elements available for manipulation, and these UI components exhibit variability owing to real-time internet data exchange. The vast and dynamic natures are further amplified when considering all the APPs within `AndroidArena`. Our designed actions can be categorized into four groups: 1) APP-level actions are responsible for installing, launching, and stopping APPs; 2) Component-level actions directly operate the UI components such as clicking, typing, and swiping etc; 3) System-level actions include turning the screen on and off, adjusting the volume, and taking screenshots etc; and 4) Task-level action is issued when the agent deems the task should finish. The complete action space is in §B.

4 SCALABLE MOBILE TASK GENERATOR

The tasks executed in `AndroidArena` environment are distinguished from other benchmarks by incorporating cross-APP collaboration and constrained tasks scenarios, commonly encountered in real-life but ignored in existing benchmarks. Even for single-APP tasks, existing benchmarks are either small-scale [40] or derived from the PixelHelp forum [21, 39], a platform dedicated to discussing

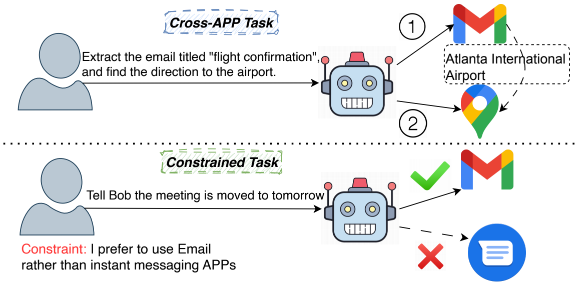

Cross-APP Task
Extract the email titled "flight confirmation", and find the direction to the airport.
①
Atlanta International Airport
②
Constrained Task
Tell Bob the meeting is moved to tomorrow
Constraint: I prefer to use Email rather than instant messaging APPs

Figure 2: Examples of cross-APP and constrained tasks.

Table 2: The statistics of our benchmark.

Task Type	#Tasks	Avg. Len. of Action Sequence
single-APP tasks	164	6.13
cross-APP tasks	22	11.14
constrained tasks	35	6.03

phone-related issues, thus deviating from routine tasks. These shortcomings underscore the necessity for a benchmark that exhibits higher **scalability** and **aligns closely with human experiences**, while accounting for **cross-APP** and **constrained tasks**.

This section outlines our proposed Mobile Task Generator (MTG in short), a framework for scalable task construction. MTG not only aligns with typical human interaction patterns, but also encompasses a diverse array of APP functions, enabling to evaluate the agents across a broader spectrum. The constructed benchmark comprises three task categories: *single-APP tasks*, *cross-APP tasks*, and *constrained tasks*. The single-APP and cross-APP tasks are crafted to assess the agents' proficiency in solving general tasks, and more complex tasks requiring cooperation between two APPs. In contrast to the former two categories focusing on task completion, the constrained tasks are designed to evaluate agents' proficiency in comprehending predefined constraints. In our benchmark, each task consists of a natural language instruction and a sequence of labeled actions for task completion. Constrained tasks additionally include a field of constraints represented in natural language. Examples of cross-APP and constrained tasks are shown in Fig. 2. The statistical information of our benchmark is presented in Table 2.

4.1 Single- and Cross-APP Tasks Construction

APP Functionalities Extraction. We incorporate 13 testing APPs from pre-installed Google suite, including Calendar, Camera, Clock, Contacts, YouTube, Weather, Settings, Photos, Messages, Google Maps, Google Drive, Gmail and Firefox. They are designed to work seamlessly with the Android OS and provide essential services. Our objective is to formulate the task instructions that cover rich and diverse functionalities of APPs while aligning with the authentic usage behavior of humans. To achieve this goal, we propose leveraging insights gleaned from human discussions and shared experiences regarding APPs available on the internet. Concretely, we first formulate queries centered on the usage of specific APPs and employ search engines to retrieve related webpages. As depicted in Fig. 3, exemplified constructed queries are *"Gmail and Calendar collaboration features"* and *"How to use Gmail and Calendar together for tasks"*. We then build a vector database to store these high volume of webpages containing rich functionalities that genuinely engage and concern users. By retrieving from the database with LLM and a specific prompt, we can extract confined APP functionalities.

Instruction Generation and Evolution Our next step involves utilizing a LLM with a functionality-to-instruction prompt to generate initial task instructions grounded in the identified APP functionalities. To automatically mass-produce more instructions, we employ the Evol-Instruct [36] strategy to expand the original instructions. In the application of this strategy, each evolutionary iteration involves using LLM along with two prompts, namely *in-depth evolving* and *in-breadth evolving*. The in-depth evolving prompt encourages LLM to rewrite instructions by making them more complex and challenging, while in-breadth evolving prompt aims to enhance the feature coverage and overall dataset diversity. Through the iterative execution of multiple evolutions, we sequentially derive evolution datasets, thereby expanding and refining the pool of task instructions.

Human Verification and Annotation To construct the benchmark, we engage annotators proficient in operating the testing APPs. They are first instructed to discern and filter tasks exhibiting repetitiveness, ambiguity, or impossibility to complete. Subsequently, they document their interactions with the phone. Given that there might be multiple feasible action sequences for completing a task, they are encouraged to opt for the most concise action plan with the shortest action sequence. After completing a task, annotators re-execute the annotated action sequence with a replay script, enabling them easily to verify the accuracy of annotated action sequence. Subsequently, the compiled task instructions and action demonstrations are collected into the benchmark dataset.

4.2 Constrained Tasks Construction

In the context of real-world mobile tasks, often confined by specific user preferences or security considerations, we introduce a constrained task set to assess the agents' capability to comprehend user-defined constraints and make decisions adeptly to avoid violations. Specifically, we consider three types of constraints: *APP-level*, *page-level* and *component-level* constraints. APP-level constraints involve the preferences of using specific APPs, exemplified by constraints like *"preferring not to use instant message for communication"*. Page-level constraints restrict access to a specific page, as seen in scenarios such as *"refraining from entering the label list page in Gmail due to the presence of sensitive information"*. Component-level constraints identify specific UI components as sensitive actions, *e.g., "do not click the payment button"*. It is noteworthy that the constrained tasks are meticulously selected from the single-APP task set and manually labeled with natural language constraints along with the corresponding correct action sequences.

4.3 Benchmark Construction Cost

The main cost in constructing our benchmark lies in filtering low-quality generated tasks and annotating action sequences. During the task filtering phase, we found that most generated single-APP tasks were reasonable and could be readily incorporated into the

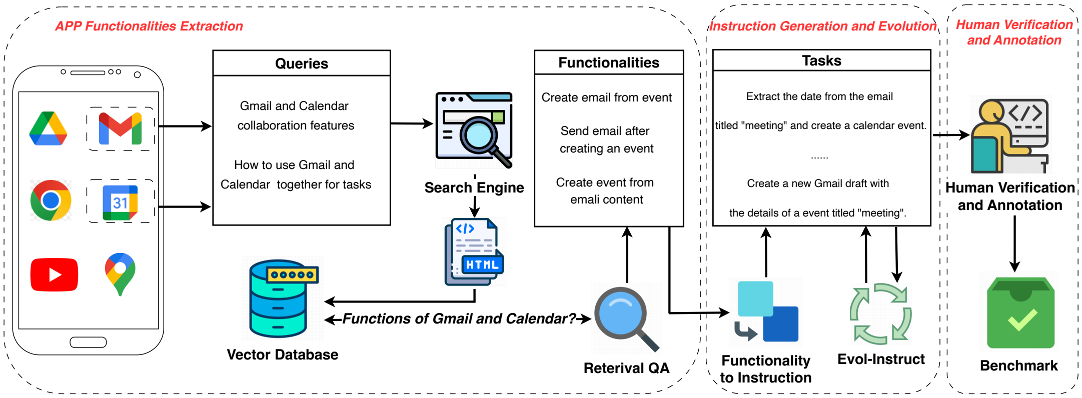

APP Functionalities Extraction
31
Queries
Gmail and Calendar collaboration features
How to use Gmail and Calendar together for tasks
Search Engine
HTML
Functionalities
Create email from event
Send email after creating an event
Create event from emali content
Vector Database
Functions of Gmail and Calendar?
Reterival QA
Instruction Generation and Evolution
Tasks
Extract the date from the email titled "meeting" and create a calendar event.
......
Create a new Gmail draft with the details of a event titled "meeting".
Functionality to Instruction
Evol-Instruct
Human Verification and Annotation
Human Verification and Annotation
Benchmark

Figure 3: Illustrative example of the MTG workflow for cross-APP (*i.e.,* Gmail and Contacts) tasks construction procedure. The single-APP tasks are generated with the same process but with different query templates and LLM prompts.

benchmark. However, the cost of filtering cross-APP tasks was considerably higher. This is due to the exponential increase in task combinations in the two-APP collaboration scenario, resulting in a total of $N * (N - 1)/2$ task sets. Many of these combinations, such as the collaboration between Google Contacts and Camera, are not practically applicable in real-life scenarios. The task filtering process required a total of four person-days to complete. Regarding the action sequence annotation, it was easy to recruit annotators who are proficient in operating the Android phone and APPs with several hours of standardized training on the annotation process. The annotation and verification process required twelve person-days to complete.

5 EVALUATION METRICS

Designing precise metrics is essential for accurately and comprehensively evaluating agent's performance. However, existing metrics employed in multi-step decision-making scenarios [7, 22] exhibit **imprecise** and **surface-level** evaluation issues, which hinder them to fully understand the performance and weakness of LLM agent. To address these limitations, we propose a novel set of metrics to evaluate agent performance in a more **adaptive and precise manner**, and to assess **fine-grained agent planning abilities**.

5.1 Adaptive and Precise Task Completion Evaluation

To begin with, we introduce the notations of action sequences. Given a task, its annotated action sequence can be represented as $\mathbf{a}$ of length L, and the actual executed actions is $\hat{\mathbf{a}}$ of length $\hat{L}$. Exemplary instances of $\mathbf{a}$ and $\hat{\mathbf{a}}$ are illustrated as follows:

$$\mathbf{a} = ABCDEFG \tag{1}$$

$$\hat{\mathbf{a}} = AXYBUVWEFFFGZ, \tag{2}$$

where each uppercase character denotes a distinct type of action. Many existing metrics [7, 22] adopt the *step-wise matching* method, which is imprecise in this scenario. In Eq. 2, the agent identifies the correct action B after two steps of exploration (*i.e.,* X and Y). Despite this action sequence not aligning with the ground truth (*i.e.,* Eq. 1) in the step-wise manner, it leads to the correct subsequent step and constitutes a valid action sequence for completing the task. Therefore, previous metrics exhibit inaccuracies in the multi-step decision-making environments where multiple feasible action sequences exist. In contrast to previous greedy step-wise matching, we propose to align the two sequences in an adaptive way, *i.e.,* calculating their **longest common subsequence** (LCS) $\mathbf{a}_{lcs}$ (marked in red in Eq. 1 and 2). The LCS **accurately** and **adaptively** reflects task completion in the multi-step decision-making scenario. Based on the accurate LCS, we propose our metrics to evaluate the task completion as follows:

- **Task Reward (TR).** $TR = \sum_{i=0}^{L} \gamma^{(L-i)} \mathbb{1}_i$, where $\gamma \in [0, 1]$ is the reward discount factor, $\gamma^{(L-i)}$ assigns higher rewards to the actions that are closer to the final action (*e.g.,* G in Eq. 1), and $\mathbb{1}_i$ equals 1 when the i-th action is in the LCS. This metric considers both the action matching and the distance towards task success.
- **Task Completion Ratio (TCR).** $TCR = k/L$, where k is the index of the last matched action in the LCS. This metrics measure the progress of task completion.
- **Reversed Redundancy Ratio (RRR).** $RRR = L/\hat{L}$. It can be used to evaluate the efficiency of the agent completing a task. We inverse it for the convenience of comparison, *i.e.,* the higher this metric, the greater the efficiency of the agent.
- **Success Rate (SR).** Unlike the above three metrics relying on ground truth action sequence, the SR is judged by the GPT-4 solely given the trajectory including historical actions and observations. SR equals 1 when GPT-4 perceives that the task has been successfully completed, and 0 when the task is deemed unsuccessful. This metric is devised for the unsupervised scenario, enhancing the scalability of the evaluation. In §6.2, we provide statistical evidence to demonstrate the accuracy of SR.

5.2 Understand Root Cause with Fine-grained Abilities Evaluation

In addition to providing adaptive and accurate metrics for evaluating task completion in complex decision-making scenarios, another

Algorithm 1 Deep Q-learning

```
Initialize replay memory $\mathcal{D}$ and action-value function $Q$
for episode = $1, M$ do
    Initialise state $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$    ▷ Explore the environment
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$    ▷ Reason the next action
        Execute action $a_t$ in emulator and observe reward $r_t$ and state $s_{t+1}$
        Preprocess observation $\phi_{t+1} = \phi(s_{t+1})$    ▷ Understand the environment and observation
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Optimize Q based on a minibatch sampled from $\mathcal{D}$    ▷ Reflection from experience
    end for
end for
```

primary objective of our study is to investigate the underlying root cause contributing to the success or failure of agents planning. Recognizing that RL serves as a classical and effective approach to address the CMDP problem [9], we abstract the fundamental elements and mechanisms in RL agents, and propose fine-grained capabilities tailored to assess LLM agents. Here we use the DQN [18] (Algorithm 1), one of the most classical RL algorithms, as an example. We decompose it into four key dimensions, *i.e.,***understanding**, **reasoning**, **exploration**, and **reflection**.

Understanding. The aspect of understanding encompasses the agent's proficiency in comprehending observation and adhering to the action format and space specified in the prompt. Unlike RL agents confining their output actions strictly within a predefined action space, the output space of LLM spans the entire vocabulary. This imposes a great demand on LLM agents to fully understand and adhere to the specified action format and space. Additionally, constrained by phone screen size limit, vital information such as the succinct status description of a checkbox, poses challenges for LLM agents in understanding crucial but brief observed details. Consequently, to comprehensively gauge the agent's understanding ability, we formulate three metrics, where a smaller Invalid Format and Invalid Action indicates a better understanding of specified action rules, and a smaller Nuggets Mining denotes the agent can better grasp important pieces of information from observation.

- **Invalid Format.** The ratio of outputting actions that deviate from the format predefined in prompt.
- **Invalid Action.** The ratio of outputting actions outside the action space specified in prompt.
- **Nuggets Mining.** The ratio of the target element length to the entire observation, assessing the agent's capacity to understand the task context and identify pivotal pieces of information. For example, when the agent correctly selects the *Bob* as shown in Fig. 1, the Nuggets Mining can be computed as the division of the length of the *[nd725]* entry by the total length of the observation.

Reasoning. This dimension indicates the agent's capacity to deduce the most suitable action based on the current observation. To assess it, two metrics are employed:

- **Operation Logic.** The inverse number of incorrect actions attempted before successfully finding the correct action. Consider Eq. 1 and 2 as an example. The agent correctly executes action B after two erroneous attempts, *i.e.,* X and Y. Therefore, the Operation Logic for this subsequence is calculated as $1/2$.
- **Awareness of Completion.** The ratio of cases that the agent correctly finds the task completed and issues a finish action. A higher value indicates the agent's better ability to reason the task-complete action.

Exploration. LLM agents make decisions from pretraining-derived prior knowledge. Due to the static nature of their prior knowledge, certain LLM agents exhibit a proclivity to iteratively execute the same erroneous action [44]. It precludes them from exploring alternative action pathways to ascertain the correct execution path. This phenomenon reflects the agent's exploration ability, which we quantify by counting the instances of action repetition.

- **Repeat Actions.** The ratio of actions resulting in repetitive or cyclical patterns. A higher value denotes the agent tends to be stuck in a specific state and cannot explore the environment.

Reflection. Similar to RL agents, the LLM agents are proven to have the capability to extract insights from previous trials and leverage the insights for subsequent executions [23]. We utilize the Reflexion mechanism to gauge the agent's proficiency in extracting pertinent experiences and applying them judiciously.

- **Reflexion@K.** $Reflexion@K = \sum_{i=1}^{K}(SR_i - SR_{i-1})$, where K is the number of Reflexion iterations. It measures the differences between the original trail and the trail after Reflexion, and a higher value indicates the agent's stronger ability to learn from experience and reflection from previous trials.

Following AgentBench [16], we normalize these metrics to $[0,\ 1]$ across all agent models. It is important to note that smaller metrics in understanding and exploration indicate better performance in these dimensions, while larger values for the metrics in reasoning and reflection denote superior performance in those aspects. The calculations for the four dimensions are specified as follow:

$$\begin{aligned}
\text{Understanding} &= (1 - \text{Invalid Format Ratio}) + \\
&\qquad (1 - \text{Invalid Action Ratio}) + (1 - \text{Nuggets Mining}) \\
\text{Reasoning} &= \text{Operation Logic} + \text{Awareness of Completion} \\
\text{Exploration} &= 1 - \text{Repeat Action Ratio} \\
\text{Reflection} &= \text{Reflexion@K}
\end{aligned}$$

Remark: The four dimensions are not mutually independent. For instance, a prerequisite for reasoning the optimal action is a thorough understanding of the environment and observation. Our objective is to assess agent abilities from diverse perspectives rather than segregating them into independent components. It is worth noting that our proposed dimensions and metrics can be generalized to other LLM agents, enabling the evaluation of their capabilities in different environments.

Table 3: Performances evaluated on single-APP and cross-APP tasks. Cross-APP tasks pose a significant challenge for SOTA agents, and highlight a substantial disparity between GPT-4 and other agents.

	Single-APP Tasks				Cross-APP Tasks			
Model	TR	TCR	RRR	SR	TR	TCR	RRR	SR
LLaMA2-13B	0.025	0.038	0.007	0.023	0.027	0.084	0.000	0.000
LLaMA2-70B	0.237	0.301	0.047	0.127	0.062	0.089	0.000	0.000
GPT-3.5	0.413	0.555	0.262	0.449	0.214	0.390	0.021	0.048
GPT-4	**0.502**	**0.689**	**0.755**	**0.759**	**0.421**	**0.746**	**0.685**	**0.571**

Table 4: The Pearson Correlation Coefficient of SR with information richness (IR), and with Operation Simplicity (OpS), and with the multiplication of IR and OpS.

Metrics	GPT-3.5	GPT-4
IR	0.37	0.62
OpS	0.61	0.28
IR × OpS	0.68	0.57

6 EXPERIMENTS AND FINDINGS

In this section, we setup the experiments, and present the experimental results. We summarize noteworthy findings as follows. First, existing SOTA agents still exhibit **substantial room for improvement** (§6.2). Second, in contrast to the results observed in prior benchmarks [6, 16], **LLaMA2-70B exhibits inferior planning abilities across various dimensions. GPT-4, while advanced, requires further improvement in the exploration and reflection dimensions.** (§6.3).

6.1 Evaluation Setting

We conduct experiments on SOTA open-source and closed-source LLMs. The detailed experiment settings are introduced as follows.
Agent Models. The selected LLMs encompass GPT-{3.5-turbo, 4} [19], LLaMA2-{13B-chat, 70B-chat} [26], representing two powerful closed-source and open-source LLM model families, respectively. The open-source LLM agents are deployed on a server equipped with 8*A100 GPUs, and GPT-3.5 and GPT-4 are accessed via through Azure OpenAI API. Regarding the prompt settings for LLM agents, please refer to §D.
Max Step. We set maximum step limits for agents to evaluate their capabilities of completing tasks within reasonable timeframe. According to the length of action sequences as shown in Table 2, we empirically set the maximum step limit as 15 for single-APP and constrained tasks, while for cross-APP tasks, it is set as 30.

6.2 Poor Performance in Mobile Tasks

In this section, we integrate the metrics introduced in §5.1 to assess the task completion of LLM agents across various task types. We report the results across single-APP, cross-APP, and constrained tasks in Table 3 and Table 5. Recall that the Success Rate (SR) is assessed by GPT-4. To validate its reliability, we perform cross-validation between it with TR and TCR, where TR and TCR represent alternative perspectives on task completion. Specifically, we compute the Pearson Correlation Coefficient (PCC) [5] between SR and TR, resulting in a correlation of 0.87, and between SR and TCR, yielding a correlation of 0.91. These high coefficients indicate a substantial correlation between SR and both TR and TCR, **validating the rationale of adopting the GPT-4 judgment mechanism**.

Table 3 reveals a **significant deficiency of SOTA agents in the real-world mobile tasks**. While GPT-4 achieves a 75.9% SR on single-APP tasks, all agents exhibit an inability to make effective decisions across other task settings. Noteworthy the performance gap between GPT-4 and GPT-3.5 is much larger for cross-APP tasks than single-APP tasks. It indicates that the cross-APP tasks are more complex and difficult, and can well reveal the **significant disparity in planning abilities between the two agents**. In contrast to prior benchmark studies, LLaMA2-70B demonstrates inferior performance relative to GPT-3.5 and GPT-4.

Table 5: Constraints violation ratios for Constrained Tasks.

Model	APP-level	Page-level	Component-level
GPT-3.5	0.207	0.072	0.33
GPT-4	0.000	0.050	0.00

We conduct a detailed examination of the APPs where the SOTA agents, including GPT-3.5 and GPT-4, do not perform well. Our investigation reveals **a vulnerability in handling APPs characterized by deficient textual information and intricate operational logics**. To further substantiate this observation, we calculate the PCC between SR and the information richness (IR) and Operation Simplicity (OpS). Specifically, we utilize the average length of APP observation and the inverse length of ground truth actions to quantify IR and OpS, respectively. The results in Table 4 demonstrate that OpS poses a more substantial challenge for GPT-3.5 in achieving a higher SR, while GPT-4 exhibits a greater sensitivity to IR. The high values of IR × OpS further prove our findings.

Beyond basic task completion, we assess the agents' capacity to comprehend constraints and adeptly make decisions to avoid violations. Table 5 presents the constraint violation ratios of GPT-3.5 and GPT-4. LLaMA2 models are excluded as they face challenges in completing basic tasks, rendering this assessment impractical. Table 5 reveals that even for straightforward constraints, GPT-3.5 still may violate them. By reading its intermediate reasoning processes, we discern that GPT-3.5 lacks awareness and understanding of constraints. For instance, in the case of *"Find the current weather forecast"* with the constraint *"do not use the Weather APP"*, GPT-3.5 directly opens the Weather APP, while GPT-4 comprehends the constraint and devises an alternative way using a web browser to search for weather forecast. It highlights the **considerable distance yet to be covered before GPT-3.5 can be applied effectively in permission-sensitive environments**.

6.3 Four Weakness Leading to Failure

In this section, we employ the metrics introduced in §5.2 to quantify the fine-grained planning abilities of these agents, so as to understand their weaknesses that lead to failure. As shown in Fig. 4, GPT-4 shows superior performances across various dimensions,

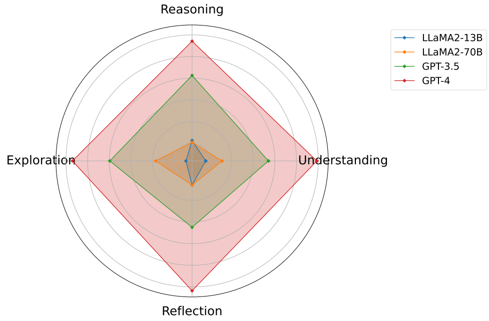
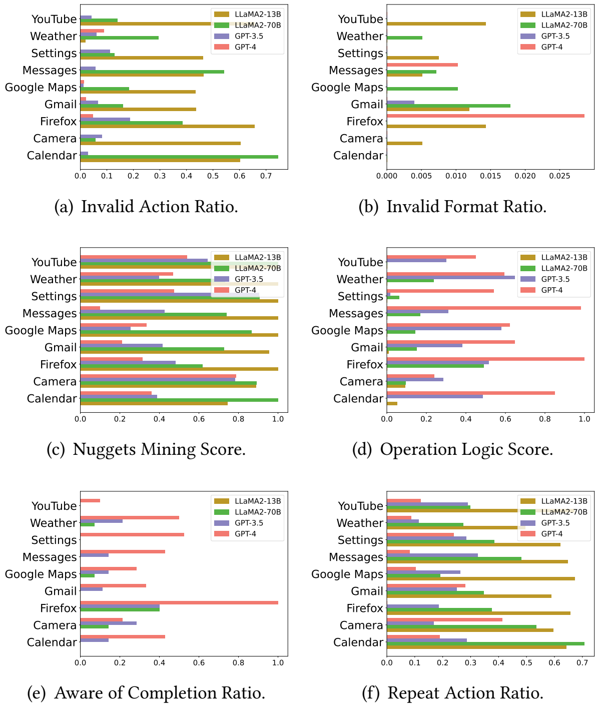


Figure 4: Agent abilities evaluation on cross-APP tasks.



(a) Invalid Action Ratio.

(b) Invalid Format Ratio.

(c) Nuggets Mining Score.

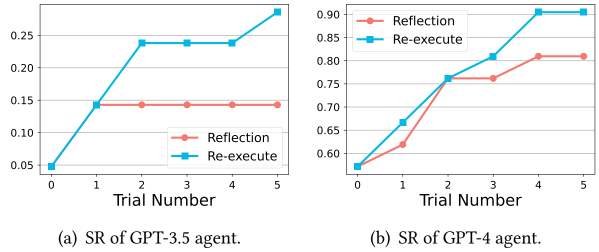
(d) Operation Logic Score.

(e) Aware of Completion Ratio.

(f) Repeat Action Ratio.

Figure 5: Metrics for understanding, reasoning and exploration dimensions.

further substantiating its excellence in task completion, as indicated in Table 3. In contrast, **LLaMA2 models exhibit significant weaknesses across all four dimensions**. In Fig. 5, we present the composed metrics of these dimensions. Due to space limit, we only present part of the testing APPs, and the complete APP metrics can be found in §C. Fig. 5(a) and 5(b) present the ratios of outputting invalid format and out-of-space actions. The notably higher ratios of LLaMA2 show its challenges in understanding and adhering to prescribed action rules. Fig. 5(c) demonstrates the superior capacity of GPT-3.5 and GPT-4 to apprehend more nuanced information compared to LLaMA2. In Fig. 5(d), LLaMA2 agents exhibit challenges in identifying the correct subsequent actions even after multiple attempts. Moreover, LLaMA2-13B lacks the awareness that the task has been successfully completed, as depicted in Fig. 5(e). Fig. 5(f) indicates a high repeat action ratio of LLaMA2, underscoring its limited ability to explore the environment. **GPT-4 also demonstrates a notable proclivity for repeating erroneous actions for several APPs**. To improve the exploration ability of GPT-4, we introduce an exploration strategy and examine its impact on performance in §7.2. While GPT-4 shows certain improvement through Reflexion, our analysis suggests that **it stems from inherent opportunities for additional attempts to complete the task rather than an enhancement in the agent's policy**. Detailed experiments and analysis can be found in §7.1.



(a) SR of GPT-3.5 agent.

(b) SR of GPT-4 agent.

Figure 6: Performances evaluated on cross-APP tasks by increasing the reflection times.

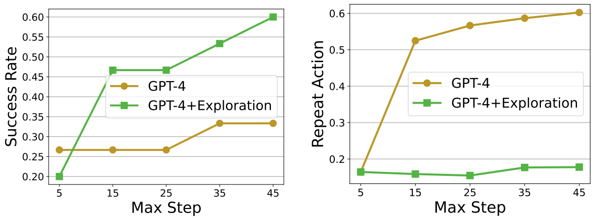
7 FUTURE DIRECTIONS FOR ENHANCING LLM AGENT

Through experiments in §6.3, we observe that LLaMA2 models display weaknesses across all four dimensions. Even for the leading model, GPT-4, still exhibits shortcomings in exploration and reflection. In this section, we first analyze the ineffectiveness of Reflexion and provide an empirical analysis of the factors contributing to this phenomenon. Second, we propose a novel prompt-based exploration method, revealing that explicitly encouraging the agent to explore unknown actions can enhance performance.

7.1 Analysis of Reflection's Failure

Recall that Reflexion summarizes experience and then re-executes failed tasks, it inherently offers opportunities for additional attempts and possesses potential for performance improvement. Accordingly, we conduct a comparative evaluation with re-executing failed tasks without the Reflexion process, namely *Re-execute*. In Fig. 6, we present the SR of Reflexion@5 and Re-execute@5. Contrary to expectations, we observe that **Reflexion does not yield positive outcomes compared to Re-execute**. This unexpected phenomenon motivates an investigation of the underlying mechanisms of Reflexion and the challenges of applying it in our scenario. To initiate our investigation, we provide a formal definition of the Reflexion process, specified as follows:

$$\underbrace{P(\text{new trajectory} \mid \text{reflection})}_{\text{③}} \cdot \underbrace{P(\text{reflection} \mid \overbrace{\text{old trajectory}}^{\text{①}})}_{\text{②}},$$



(a) Success rate.

(b) Repeat action ratio.

Figure 7: GPT-4 and GPT-4+Exploration comparisons by varying the maximum step limit on the Camera APP.

where P denotes the LLM agents. This equation describes the Reflexion process, *i.e.,* extracting valuable insights from past trajectories and benefiting subsequent trials. Three key steps in this equation may contribute to the degradation of Reflexion performance in our scenario. The first and **the most important reason is that the old trajectory (marked in ①) is less informative** compared with previous scenarios. Unlike benchmarks [3, 41] characterized by one-step decision, and virtual ALFWorld environment [24] with small and static action space, our environment necessitates multi-step planning within a vast and dynamic action space. This challenge makes it hard to explore the entire action space and sparsifies the reward feedback, thus cannot provide sufficient guidance for next trial. Therefore, a potential improvement can be achieved by improving trace quality. In specific, employing exploration strategies to broaden the explored action space for informative experience [17] and devising intrinsic rewards to mitigate the sparse reward issue [43]. Secondly, a constrained ability to distill reflection (*i.e.,* part ②) diminishes the reflection efficacy. Lastly, regarding part ③, the reflection may not be fully leveraged by agent or, conversely, introduces bias [10] and degrades the performance compared to the Re-execute that is without reflection.

7.2 Enhancing Exploration Boosts Performance

Upon reading the trajectories, we observe that even for GPT-4, it still presents a pronounced tendency to repeat erroneous actions, as illustrated in Fig. 5(f), indicating its limited exploration capabilities. Furthermore, the repetition of actions degrades the quality of preceding trajectories, rendering them insufficient for providing informative guidance for reflection, as discussed in §7.1.

In this section, we introduce a novel prompt-based exploration strategy for LLM agents. Diverging from prior approaches [4] that treat the LLM as a RL policy network and employ exploration strategies originating from RL, our strategy guides the exploration of LLM agent by **incorporating a prompt indicating the count of previously visited observations $M(\mathbf{s})$ and issued actions $N(\mathbf{s}, \mathbf{a})$**. Specifically, we embed a hint prompt such as *"You have already been in the current state M times, and taken action A for N times"* at each decision step. This concept is inspired by the Upper Confidence Bound (UCB) [2, 31]. Unlike UCB, we do not design explicit exploration strategies. Instead, we integrate historical information into the prompt, leveraging the powerful decision-making capabilities of the LLM to balance exploration and exploitation. We conduct an experiment on the Camera APP, where GPT-4 exhibits the highest repeat action ratio, to evaluate the effectiveness of the exploration strategy. We vary the maximum step limit in $\{5, 15, 25, 35, 45\}$, and present SR and repeat action ratio in Fig. 7. The results show that, with a simple counting-based prompt, SR can achieve 27% of improvement. Furthermore, as the maximum step limit increases, the exploration ability of GPT-4 degrades. In contrast, for GPT-4+Exploration, effective exploration of the environment persists, leading to continued performance improvement.

8 CONCLUSION

This study introduces `AndroidArena` environment and a scalable benchmark. It would benefit a broad spectrum of audiences, including both developers and users of LLM agents. For developers, we open-sourced an easily accessible environment with a comprehensive action space and an elaborately designed observation compression method, facilitating the practical development of phone intelligent assistants. It supports the evaluation of cross-APP and constrained task scenarios. We propose adaptive and precise metrics to assess task completion, and fine-grained abilities of agents to understand their weaknesses. The results underscore significant room for improvement among SOTA agents. Understanding the capability boundary of SOTA LLM agents is crucial for making informed decisions about their usage for customers, allowing them to maximize the benefits of the phone assistant. We highlight four research directions for enhancing LLM agents. Additionally, we offer empirical insights into the failure of reflection and present a novel method to enhance the exploration capabilities of agents. In the future, we plan to investigate the weaknesses of multi-modal model agents. Given that vision models excel at spatial understanding and reasoning, areas where LLMs struggle [37], we intend to scrutinize their fine-grained abilities and identify promising research directions in this domain. Our `AndroidArena` supports multi-modal evaluation, and the benchmark can be easily extended to this setting.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments on this paper. Zhen Xiao is the corresponding author.

REFERENCES

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691* (2022).

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47 (2002), 235–256.

[3] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[4] Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. 2023. Grounding large language models in interactive environments with online reinforcement learning. *arXiv preprint arXiv:2302.02662* (2023).

[5] Israel Cohen, Yiteng Huang, Jingdong Chen, Jacob Benesty, Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. *Noise reduction in speech processing* (2009), 1–4.

[6] Nicholas Crispino, Kyle Montgomery, Fankun Zeng, Dawn Song, and Chenguang Wang. 2023. Agent Instructs Large Language Models to be General Zero-Shot Reasoners. *arXiv preprint arXiv:2310.03710* (2023).

[7] 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 (2023).

[8] S Gunasekaran and V Bargavi. 2015. Survey on automation testing tools for mobile applications. *International Journal of Advanced Engineering Research and Science* 2, 11 (2015), 2349–6495.

[9] Assaf Hallak, Dotan Di Castro, and Shie Mannor. 2015. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259* (2015).

[10] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798* (2023).

[11] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*. PMLR, 9118–9147.

[12] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. 2023. Reward design with language models. *arXiv preprint arXiv:2303.00001* (2023).

[13] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111* (2023).

[14] Pengze Li, Mingxuan Song, Mingzhe Xing, Zhen Xiao, Qiuyu Ding, Shengjie Guan, and Jieyi Long. 2024. SPRING: Improving the Throughput of Sharding Blockchain via Deep Reinforcement Learning Based State Placement. In *Proceedings of the ACM on Web Conference 2024*. 2836–2846.

[15] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal LLM Agents: Insights and Survey about the Capability, Efficiency and Security. *arXiv preprint arXiv:2401.05459* (2024).

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

[17] Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. 2023. LASER: LLM Agent with State-Space Exploration for Web Navigation. *arXiv preprint arXiv:2309.08172* (2023).

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[19] R OpenAI. 2023. GPT-4 technical report. *arXiv* (2023), 2303–08774.

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

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

[22] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427* (2023).

[23] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[24] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768* (2020).

[25] Richard S Sutton, Andrew G Barto, et al. 1999. Reinforcement learning. *Journal of Cognitive Neuroscience* 11, 1 (1999), 126–134.

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

[27] 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. abs/2105.13231 (2021). arXiv:2105.13231 [cs.LG] http://arxiv.org/abs/2105.13231

[28] Boshi Wang, Xiang Yue, and Huan Sun. 2023. Can ChatGPT Defend its Belief in Truth? Evaluating LLM Reasoning via Debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 11865–11881.

[29] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).

[30] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2023. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432* (2023).

[31] Yingpeng Wen, Qinliang Su, Minghua Shen, and Nong Xiao. 2022. Improving the exploration efficiency of DQNs via the confidence bound methods. *Applied Intelligence* (2022), 1–15.

[32] Yue Wu, Xuan Tang, Tom M Mitchell, and Yuanzhi Li. 2023. SmartPlay: A Benchmark for LLMs as Intelligent Agents. *arXiv preprint arXiv:2310.01557* (2023).

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

[34] Mingzhe Xing, Hangyu Mao, and Zhen Xiao. [n. d.]. Fast and Fine-grained Autoscaler for Streaming Jobs with Reinforcement Learning.

[35] Mingzhe Xing, Hangyu Mao, Shenglin Yin, Lichen Pan, Zhengchao Zhang, Zhen Xiao, and Jieyi Long. 2023. A Dual-Agent Scheduler for Distributed Deep Learning Jobs on Public Cloud via Reinforcement Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2776–2788.

[36] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).

[37] Yutaro Yamada, Yihan Bao, Andrew K Lampinen, Jungo Kasai, and Ilker Yildirim. 2023. Evaluating Spatial Understanding of Large Language Models. *arXiv preprint arXiv:2310.14540* (2023).

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

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

[40] Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771* (2023).

[41] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600* (2018).

[42] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[43] Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. 2023. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563* (2023).

[44] 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* (2023).

APPENDIX

A DETAILED OBSERVATION COMPRESSION METHOD

The textual observation is derived from the XML representation encapsulating comprehensive screen information as shown in Fig. 8. However, directly inputting the entire XML into the LLM, proves to be excessively lengthy as illustrated in Table 6. To mitigate this, we employ a two-phase heuristic approach for compressing the XML to a manageable length for LLM processing. The XML entries are categorized into two groups: one for layout, which does not support actionable operations, and the other for UI components. We eliminate the XML entries related to layout, retaining only those associated with UI components. In the second phase, we merge non-functional and non-visible nodes upwards, incorporating their descriptive information into the parent nodes. This strategy enhances the LLM's ability to understand the semantic of the hierarchical XML tree, and result in a more efficient compression. For components with nuanced state descriptions, we amplify their textual information. For instance, when the switch component is in the off position, we append a description stating *"it is currently unchecked, and you can switch it on."*. To enable the agent to accurately select the UI component for operation, a unique ID is assigned to each component in the compressed observation. In the compressed observation, components are structurally organized, maintaining their ancestral-descendant relationships in the original XML tree, aiding the LLM agent in comprehending the interface's layout through text and thereby enhancing its command efficacy.

Table 6: We randomly select several APPs and compare the token numbers before and after compression. Our compression ratio reach a high of 86.6%, while preserving the semantic information. This approach enhances the utilization of the LLM agent context, allowing for the accommodation of more historical observations in each decision-making process.

App Name	#Token (Original)	#Token (Compressed)
Gmail (email list)	11,707	1,155
Gmail (compose email)	7,273	413
Calendar	8,604	584
Google map	15,725	637
YouTube	12,005	939
Play Store	10,450	620
Google drive	11,060	651
Clock Alarm	9,633	505
Clock	7,980	285

B DETAILED ACTION SPACE

We support four-level's action space, *i.e.,* APP level, component level, system level and task level. App level actions are responsible for installing, launching and stopping APPs. Most actions are component level which are responsible for operating UI components, such as clicking, typing, and swiping. We also support system level actions including turning the screen on and off, adjusting the volume, setting orientation, and taking screenshots. Task-level action is designed for the agent to decide if a task should finish.

In previous work like AndroidEnv, the action is done by successive touches and lifts, each consists a position (x, y) and an $ActionType \in \{TOUCH, LIFT, REPEAT\}$. AndroidEnv divides the screen into a grid and restricts the ActionType to TOUCH, or groups action sequences like [TOUCH, LIFT, TOUCH, LIFT] into a single gesture, such as swiping, scrolling, or drag-and-drop. However, continuous touches and lifts bring additional inference overhead for agents, and cannot accurately simulate the continuity and smoothness of swiping. Instead of interacting with the phone by successive touches and lifts, we directly operate the UI components of APPs through UIAutomator. It is a testing framework for Android, sending a series of events including pressing, dragging, and scrolling. These events are consistent with real finger slides. Operating components by sending action events is not only more accurate and natural in simulating real user operations, but also superior in terms of APP compatibility. We can get the executable actions that each component can perform from the corresponding XML and maintain them in the compressed observation, such as clickable, double clickable, long clickable, etc. At the same time, we also record the type of each component in the compressed observation, such as button, text-editor, which can assist the agent to give appropriate action instructions. As we have set a unique ID for each component in the compressed observation, the agent can operate a component by specifying its ID and the corresponding action type. Since UIAutomator locates and operates on components based on their XPath, our implementation employs a mapping table to convert component IDs into component XPaths, after which we perform the operations.

Table 7: The complete action space, including action type and the corresponding parameters.

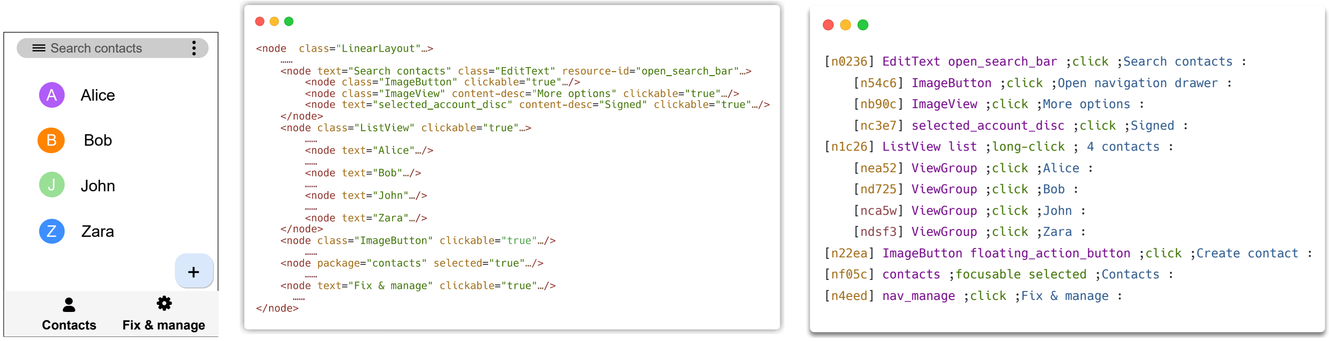
Action level	Action Type	Action Parameters
APP level	Install APP	Download link
	Launch APP	Package name
	Stop APP	Package name
	Stop all APP	
Component level	Click	XPath
	Double click	XPath
	Long click	XPath
	Set text	XPath, Text
	Swipe up/down/left/right	Ratio
	Press back	
	Press home	
System level	Screen on/off	
	Volume up/down/mute	
	Set orientation	Horizontal/vertical
	Screenshot	
Task Level	Finish task	

C COMPLETE RESULTS FOR TESTING APPS

Due to space limit, we present part of the testing APPs in §6.2. In Fig. 9, we show the metrics for all testing APPs.

D PROMPT DESIGN

Following WebArena, our prompt for each decision-making step incorporates environment descriptions, two-shot examples, task

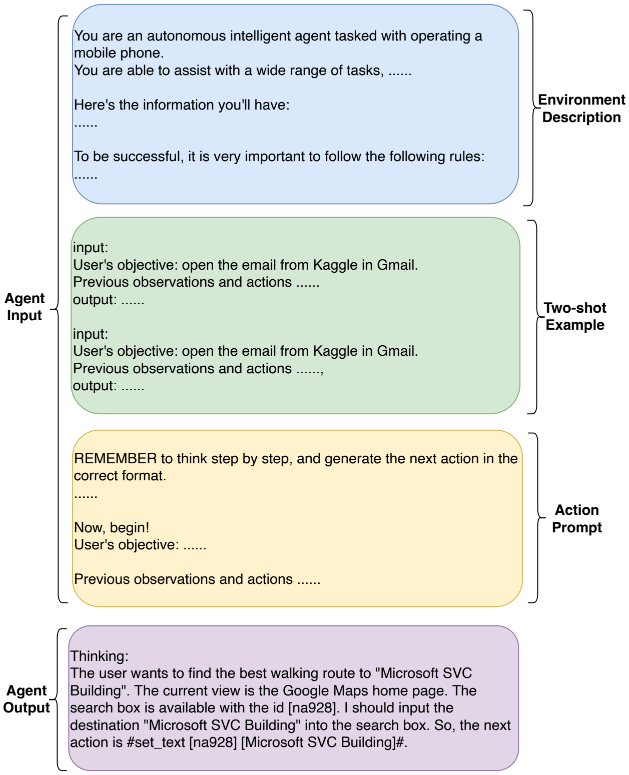


```
<node  class="LinearLayout"…>
    ……
    <node text="Search contacts" class="EditText" resource-id="open_search_bar"…>
        <node class="ImageButton" clickable="true"…/>
        <node class="ImageView" content-desc="More options" clickable="true"…/>
        <node text="selected_account_disc" content-desc="Signed" clickable="true"…/>
    </node>
    <node class="ListView" clickable="true"…>
        ……
        <node text="Alice"…/>
        ……
        <node text="Bob"…/>
        ……
        <node text="John"…/>
        ……
        <node text="Zara"…/>
    </node>
    <node class="ImageButton" clickable="true"…/>
        ……
    <node package="contacts" selected="true"…/>
        ……
    <node text="Fix & manage" clickable="true"…/>
      ……
</node>
```

```
[n0236] EditText open_search_bar ;click ;Search contacts :
    [n54c6] ImageButton ;click ;Open navigation drawer :
    [nb90c] ImageView ;click ;More options :
    [nc3e7] selected_account_disc ;click ;Signed :
[n1c26] ListView list ;long-click ; 4 contacts :
    [nea52] ViewGroup ;click ;Alice :
    [nd725] ViewGroup ;click ;Bob :
    [nca5w] ViewGroup ;click ;John :
    [ndsf3] ViewGroup ;click ;Zara :
[n22ea] ImageButton floating_action_button ;click ;Create contact :
[nf05c] contacts ;focusable selected ;Contacts :
[n4eed] nav_manage ;click ;Fix & manage :
```

(a) Screenshot. (b) XML derived by UIAutomator. (c) Compressed observation.

Figure 8: An example of the screenshot, original XML and compressed observation of Contacts APP page.



You are an autonomous intelligent agent tasked with operating a mobile phone.
You are able to assist with a wide range of tasks, ...

Here's the information you'll have:
......

To be successful, it is very important to follow the following rules:
......

input:
User's objective: open the email from Kaggle in Gmail.
Previous observations and actions ......
output: ......

input:
User's objective: open the email from Kaggle in Gmail.
Previous observations and actions ......,
output: ......

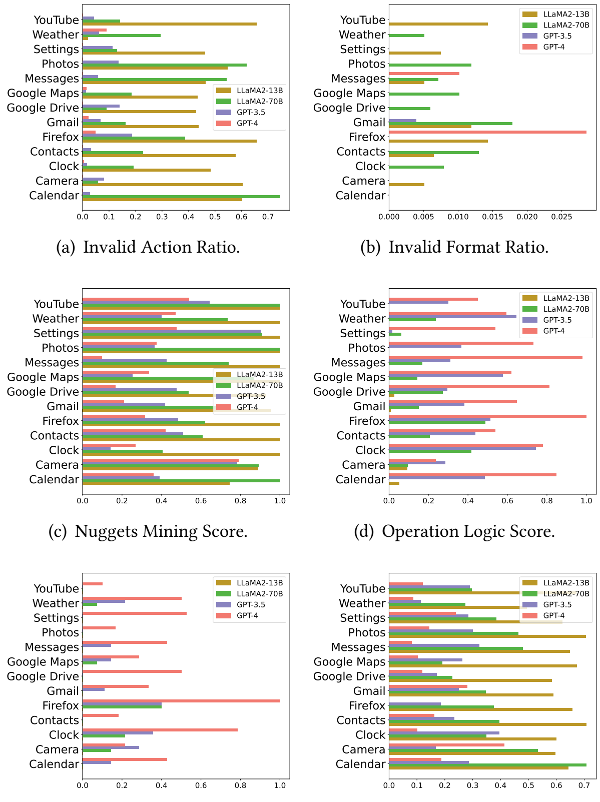
REMEMBER to think step by step, and generate the next action in the correct format.
......

Now, begin!
User's objective: ......

Previous observations and actions ......

Thinking:
The user wants to find the best walking route to "Microsoft SVC Building". The current view is the Google Maps home page. The search box is available with the id [na928]. I should input the destination "Microsoft SVC Building" into the search box. So, the next action is #set_text [na928] [Microsoft SVC Building]#.

Figure 10: Prompt structure.



(a) Invalid Action Ratio. (b) Invalid Format Ratio.

(c) Nuggets Mining Score. (d) Operation Logic Score.

(e) Aware of Completion Ratio. (f) Repeat Action Ratio.

Figure 9: Metrics for understanding, reasoning and exploration dimensions on all testing APPs.

instructions, and historical observations and actions, as illustrated in Fig. 10. We adopt the Reflexion prompt from its official implementation with modifications tailored to our specific scenario. The prompt context limit is 4K for LLaMA2-13B, LLaMA2-70B, and GPT-3.5, and at 8K for GPT-4. Given that historical observations and actions may exceed the context limit, and to ensure a fair comparison, we apply the truncation strategy employed in WebArena across all agents to maintain a prompt within the 4K context limit. The detailed prompt can be found at our released Github project.