

---

# ANDROIDWORLD: A Dynamic Benchmarking Environment for Autonomous Agents

---

Christopher Rawles<sup>\*1</sup>, Sarah Clinckemaillie<sup>†2</sup>, Yifan Chang<sup>†2</sup>, Jonathan Waltz<sup>2</sup>, Gabrielle Lau<sup>2</sup>, Marybeth Fair<sup>2</sup>, Alice Li<sup>1</sup>, William Bishop<sup>1</sup>, Wei Li<sup>1</sup>, Folawiyo Campbell-Ajala<sup>1</sup>, Daniel Toyama<sup>1</sup>, Robert Berry<sup>1</sup>, Divya Tyamagundlu<sup>2</sup>, Timothy Lillicrap<sup>1</sup>, and Oriana Riva<sup>1</sup>

<sup>1</sup>*Google DeepMind*

<sup>2</sup>*Google*

## Abstract

Autonomous agents that execute human tasks by controlling computers can enhance human productivity and application accessibility. Yet, progress in this field will be driven by realistic and reproducible benchmarks. We present ANDROIDWORLD, a fully-functioning Android environment that provides reward signals for 116 programmatic task workflows across 20 real-world Android apps. Unlike existing interactive environments, which provide a static test set, ANDROIDWORLD dynamically constructs tasks that are parameterized and expressed in natural language in unlimited ways, thus enabling testing on a much larger and realistic suite of tasks. Reward signals are derived from the computer’s system state, making them durable across task variations and extensible across different apps.

To demonstrate ANDROIDWORLD’s benefits and mode of operation, we introduce a new computer control agent, M3A. M3A can complete 30.6% of the ANDROIDWORLD’s tasks, leaving ample room for future work. Furthermore, we adapt a popular desktop web agent to work on Android, which we find to be less effective on mobile, suggesting future research is needed to achieve universal, cross-domain agents. Finally, we conduct a robustness analysis by testing M3A against a range of task variations on a representative subset of tasks, demonstrating that variations in task parameters can significantly alter the complexity of a task and therefore an agent’s performance, highlighting the importance of testing agents under diverse conditions. ANDROIDWORLD and the experiments in this paper are available at [https://github.com/google-research/android\\_world](https://github.com/google-research/android_world).

## 1 Introduction

Autonomous agents that interpret human natural language instructions and operate computing devices can provide enormous value by automating repetitive tasks, augmenting human intelligence, and accomplishing complex workflows. The enthusiasm for building such agents is evident by the growing number of computer control agent systems [41, 9, 70, 24, 22, 16] and code repositories [13, 1, 2, 57]. Yet, most existing approaches evaluate agents with proxy metrics, by comparing an agent’s trajectory to a previously collected human demonstration [28, 5, 9, 41, 63, 68, 33, 67, 61]. This kind of measurement is not representative of real-world performance [41, 70] because it does not reflect that there are usually multiple correct paths to solve a task, environments may behave non-deterministically (e.g., the environment freezes), and agents can dynamically learn from mistakes to

---

<sup>\*</sup>Lead contributor.

<sup>†</sup>Equal contribution.

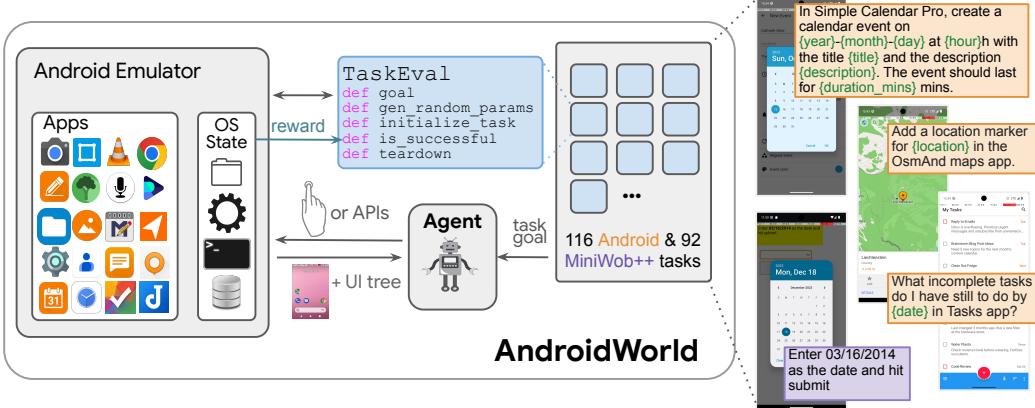


Figure 1: ANDROIDWORLD is an environment for building and testing autonomous agents.

correct their actions [46, 31, 27, 39]. Towards more accurate evaluation, emerging testing environments [73, 24, 64, 59] place agents in interactive environments and measure their task completion rates across a suite of tasks.

Reward signals are quantitative metrics that indicate functional correctness of a task, i.e. is the stated goal achieved? For example, for the task “Send a text message to Jane confirming I’ll be there”, a positive reward indicates the relevant message has been sent. Yet, real-world environments (apps and websites), unlike simulations [49, 47] and games [38, 48, 55], do not naturally provide explicit reward signals. The internals of most apps and websites are hidden from developers making it difficult to program rewards. Additionally, such systems are seemingly infinitely expansive, spanning millions of continuously-changing apps and websites. All these factors make it hard to establish reliable and durable reward functions.

One way to obtain reliable reward signals is to leverage human judgment [41, 70, 39, 23]. This allows one to conduct one-off evaluations on a diverse suite of tasks, however each evaluation run requires significant time and costly manual review. LLM-based “judges” used to evaluate natural language generation on open-ended questions [7, 71, 32] are emerging as a solution to dynamic real-world environments [10, 35] and while showing promise also for computer control agents [39, 16] they remain imperfect evaluators. As a result, existing testing environments for computer control agents that provide *automated* reward signals are limited in their real-world diversity and scale. They cover mostly single-app tasks on a small set of domains (1–6 websites [64, 73, 24]) and evaluate agents on static test sets [59, 37], instead of across varied conditions and inputs which are found in real-world scenarios.

To tackle these challenges, we introduce and release ANDROIDWORLD, a robust agent environment for Android that offers realism and scalability for developing and evaluating computer control agents. ANDROIDWORLD is an open environment consisting of a fully-functioning Android OS, allowing access to millions of Android apps and the Web. Crucially, it provides a highly reproducible task suite for developing and evaluating computer control agents, consisting of 116 tasks across 20 apps. Similarly to the popular MiniWoB++ [45] benchmark, each task is dynamically instantiated using randomly-generated parameters, challenging agents with millions of unique task goals and conditions. While MiniWoB++ consists of simple, synthetic websites, ANDROIDWORLD supports real-world Android applications. To make reward signals durable using real applications, ANDROIDWORLD’s key insight is to leverage the extensive and consistent state management capabilities of the Android operating system, using the same mechanisms that the apps themselves utilize. The dynamic nature of ANDROIDWORLD’s tasks is not only a test of adaptability for evaluation purposes, but it can also spawn new research on online learning algorithms for computer control agents.

ANDROIDWORLD is a lightweight environment, requiring only 2 GB of memory and 8 GB of disk space, and is designed with convenience in mind. It connects agents to Android OS by leveraging the

Python library AndroidEnv [53] to connect to the freely available Android Emulator.<sup>1</sup> ANDROIDWORLD is highly extensible, making it easy to add new tasks and even new benchmarks, which we demonstrate by integrating the MiniWoB++ [45, 29] benchmark.

To demonstrate ANDROIDWORLD’s usefulness as a benchmark, we build and release a new multimodal agent, M3A (Multimodal Autonomous Agent for Android), and establish state-of-the-art results on ANDROIDWORLD. We analyze M3A’s performance using both multimodal and text-only input, and we show that multimodal perception significantly improves agent performance. On ANDROIDWORLD, M3A achieves a 30.6% success rate. This performance surpasses that of a web agent adapted for Android but remains significantly lower than the human success rate of 80.0%. In pursuit of building robust computer control agents, our study extends to a series of real-world condition tests, where we show performance varies significantly across variations in intent parameters, phrasing, and environmental configurations.

We make the following contributions: (i) the creation of a new, highly diverse and realistic computer control agent environment; (ii) establishment of benchmark performance with a state-of-the-art multimodal agent, and (iii) a careful analysis demonstrating the need to evaluate agents across multiple test sets and conditions due to the inherent stochasticity in both models and environments.

## 2 Related Work

Table 1 compares existing evaluation environments for autonomous agents.

### 2.1 Interactive evaluation environments

Effective evaluation of autonomous agents requires environments that not only mimic real-world scenarios but also provide immediate reward signals upon successful task completion [41, 9, 3, 42, 6]. MiniWoB++ [45, 31] offers a lightweight framework of small, synthetic HTML pages with parameterized tasks which allow for unlimited task variability. WebShop [64] provides a simulated e-commerce environment, whereas WebArena [73] and VisualWebArena [24] consist of simulated websites across up to four domains. OSWorld [59] provides an interface and programmatic rewards across nine apps for desktop OSes. GAIA [37] is a static dataset that tests an agent’s ability to interact with live web environments. MMInA [69] is a multihop and multimodal benchmark designed to evaluate agents for compositional Internet tasks. B-MoCA [25] is the only interactive testing environment that exists today for mobile. It supports 31 tasks whose success is determined by comparing an agent’s execution with logs from human demonstrations using simple regex expressions. Unlike MiniWoB++, all these recent frameworks use a fixed set of evaluation tasks where initial conditions and goals are statically defined, limiting adaptability. As has been observed in other domains such as reinforcement learning [17, 40, 8], the current output of foundation models is stochastic and highly sensitive to their inputs [20, 34, 30, 43], motivating the need to test them under varying conditions.

Reward signal construction is crucial when comparing interactive benchmarking environments. MiniWoB++ [45] dynamically constructs tasks, embedding rewards directly in custom HTML and JS code, a method specific to web-based tasks and less adaptable to other environments. WebArena [73], VisualWebArena [24] and MMInA [69] check trajectories for specific URLs and examine result pages for expected strings or images. Information retrieval tasks compare returned answers against a ground truth. WebShop [64] identifies specific products by matching their attributes against pre-defined gold attributes. B-MoCA [25] uses regular expressions to match log outputs. These types of reward are application specific and time dependent. OSWorld [59] scales better by inspecting device states or querying the cloud to compute rewards, thus offering robust functions adaptable to content and app updates, and shareable across multiple tasks. ANDROIDWORLD enhances OSWorld’s approach by dynamically constructing the start states and varying the task goals in unlimited ways.

Additionally, other studies leverage human evaluation [41, 70, 4] for tasks where automatic evaluation is not available. Lastly, emerging research [39, 16] explores the potential of multimodal models to generalize agent evaluations to new settings, though this area requires further research to achieve accuracy comparable to manually coded rewards.

---

<sup>1</sup>The Android Emulator is packaged as part of Android Studio, which can be downloaded from <https://developer.android.com/studio>

Table 1: Comparison of different datasets and environments for benchmarking computer agents.

	Env?	# of apps or websites	# task templates	Avg # task instances	Reward method	Platform
GAIA	✗	n/a	466	1	text-match	None
MIND2WEB	✗	137	2350	1	None	Desktop Web
WEBLINX	✗	155	2337	1	None	Desktop Web
PIXELHELP	✗	4	187	1	None	Android
METAGUI	✗	6	1125	1	None	Android
MoTIF	✗	125	4707	1	None	Android (Apps+Web)
AITW	✗	357+	30378	1	None	Android (Apps+Web)
OMNIACT	✗	60+	9802	1	None	Desktop (Apps+Web)
MINIWOB++	✓	1	114	∞	HTML/Javascript state	Web (synthetic)
WEBSHOP	✓	1	12k	1	product attrs match	Desktop Web
WEBARENA	✓	6	241	3.3	url/text-match	Desktop Web
VISUALWEBARENA	✓	4	314	2.9	url/text/image-match	Desktop Web
B-MOCA	✓	12	31	1.9	regex	Android (Apps+Web)
MMINA	✓	14	1050	1	text-match	Desktop web
OSWORLD	✓	9	369	1	device/cloud state	Desktop (Apps+Web)
<b>ANDROIDWORLD</b>	✓	20	116	∞	device state	Android (Apps+Web)

## 2.2 Static datasets for automation

Datasets derived from human interactions provide proxy metrics that correlate with real-world agent performance [28, 5, 9, 41]. On mobile platforms, AitW [41], PixelHelp [28], UGIF [54], and MoTIF [5] consist of demonstrations across Android apps and mobile websites, with screens often represented via accessibility trees. In contrast, desktop web environments typically utilize the DOM for representing website content, with Mind2Web [9], OmniAct [21] and others, across various desktop websites. Mobile-based datasets frequently involve more complex actions, such as scrolling, which are not as useful in DOM-based desktop interactions where the entire action space is readily accessible. Additionally, API-centric datasets like API-Bank [26], ToolTalk [11], and ToolBench [60] assess agents’ capabilities to manipulate computer systems via APIs.

## 2.3 Interactive agents

Prior to today’s foundation models, traditional approaches to developing user interface-operating agents primarily used reinforcement learning and behavioral cloning to simulate interactions like mouse clicks and keyboard typing [28, 31, 14, 19]. More recent work tends to leverage off-the-shelf foundational models [50, 51, 52] with in-context learning (ICL) and fine-tuning applied to mobile [41, 18, 56, 61, 68, 4], desktop web [70, 9, 73, 24], and desktop OS [58, 66, 59]. Recent work explores agents that reflect on system state [46, 65, 36] by leveraging exploration, self-evaluation, and retry-capabilities enabling continual learning and adaptation [27, 63, 39, 58].

# 3 ANDROIDWORLD

## 3.1 Android for autonomous agents

Android is an ideal environment for developing autonomous agents. It is the most widely used operating system globally<sup>2</sup> and is highly flexible for research, while providing an open world of the Web<sup>3</sup> and over 2M applications for agents to operate in. Using emulation, the Android environment is easy to deploy, does not require specialized hardware, and can be run on a laptop. Android Virtual Devices (AVDs), or emulator images, are well suited for research because they are self-contained, easily distributable, and configurable.

Compared to desktop environments, mobile environments, like Android, pose unique research challenges for computer control agents. On one hand, mobile UIs tend to be simpler than their desktop

<sup>2</sup><https://gs.statcounter.com/os-market-share>.

<sup>3</sup>Mobile is the most popular platform for accessing the web; <https://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/>

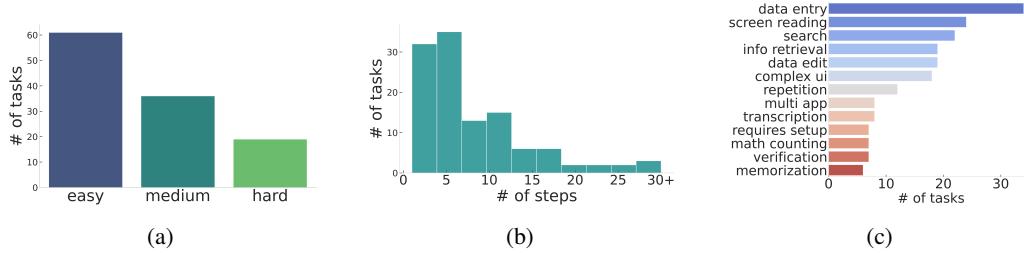


Figure 2: Human raters assessed task difficulty (2a), duration (2b), and category (2c). Each rater performed the tasks assigned to them, then assigned a difficulty level and selected relevant tags from a predefined list. They also estimated the number of steps required to complete each task, using the action space available to an agent.

counterparts because of their smaller screen size. On the other hand, the action space on mobile devices is more complicated and more actions can be required to complete tasks. Precise gestures are needed to fully operate the UI, such as when navigating a carousel widget, long-pressing on a widget, or performing multi-finger gestures to zoom in. Since it is an OS, Android is a fully open environment compared to web-browser-only environments. Android’s flexibility is also reflected in its action space; in addition to UI actions (click, scroll, type, etc.), Android provides function-calling APIs, such as sending a text message, for example, which allow computer control agents to utilize a broader action space.

### 3.2 The observation and action space

ANDROIDWORLD provides an interface for agents to receive observations and execute actions on Android. It uses AndroidEnv [53] and the Android Device Bridge to facilitate interaction between Android and the agent. The observation space consists of a full-resolution screenshot and a UI tree representation developed for accessibility purposes. The action space is similar to that which humans use, consisting of gestures (i.e., tapping, long-press, and swiping), typing, and navigation buttons (i.e., go home and go back). In addition to these naturalistic actions, ANDROIDWORLD exposes a limited set of function calling APIs, such as `send_text_message`, to help agents accomplish goals. Appendix A provides more details on the observation format and action space.

### 3.3 Reproducible and parameterized tasks

ANDROIDWORLD consists of a suite of 116 tasks, spread across 20 diverse applications (see Appendix B for more details). These tasks simulate practical, everyday activities, including note-taking, scheduling appointments, communicating through messaging, and interacting with system utilities. The suite consists of open-source apps and built-in Android system apps, such as Settings and Contacts. As rated by humans, the tasks vary in difficulty, duration, and categories (Figure 2).

To achieve a high degree of reproducibility in real-world scenarios, ANDROIDWORLD precisely controls the OS and app state in several ways. The Android OS is fixed, consisting of a Pixel 6 emulator running Android 13 with a fixed time on the date October 15th, 2023. All applications in ANDROIDWORLD are fully-functional, open-source apps, besides the OS-level apps included with Android. For the open-source apps, ANDROIDWORLD maintains a constant environment by installing a fixed version of each app, acquired from F-Droid.<sup>4</sup> OS-level apps’ versions are determined by the Android OS, which is also fixed. ANDROIDWORLD only utilizes apps that do not require login/authentication, can function offline, and store their application data on device.

In addition to managing the states of apps and operating systems, ANDROIDWORLD precisely defines and controls the state during task execution. Each task has its own unique setup, reward determination logic, and teardown procedures (see Appendix B.2 for a detailed example), ensuring a fully reproducible suite of tasks. This careful state management not only ensures reproducibility but also allows for easy customization of tasks through *parametrization*. Task parameters, initialized randomly at the start of each task based on a controlled random seed, dictate the initial state

<sup>4</sup><https://f-droid.org/>

Table 2: Selected tasks with code describing validation logic.

Task	Validation code
In Simple Calendar Pro, create a calendar event on $\{\text{event.year}\}-\{\text{event.month}\}-\{\text{event.day}\}$ at $\{\text{event.hour}\}h$ with the title ' $\{\text{event.title}\}$ ' and the description ' $\{\text{event.description}\}$ '. The event should last for $\{\text{event.duration}\}$ mins.	<code>event_exists(event)</code>
Send a text message to $\{\text{phone.number}\}$ with message: $\{\text{message}\}$ .	<code>message_exists(phone_number, message, messaging_db)</code>
Create a new drawing in Simple Draw Pro. Name it $\{\text{file.name}\}$ . Save it in the Pictures folder.	<code>file_exists(file_path)</code>
Create a timer with $\{\text{hours}\}$ hours, $\{\text{minutes}\}$ minutes, and $\{\text{seconds}\}$ seconds. Do not start the timer.	<code>timer_displays(time, ui_hierarchy)</code>
Create a new note in Markor named $\{\text{file.name}\}$ with the following text: $\{\text{text}\}$ . Share the entire content of the note with the phone number $\{\text{number}\}$ via SMS.	<code>(note_created(file_name, text) + message_exists(phone_number, message)) / 2.0</code>
Turn on WiFi and open $\{\text{app.name}\}$ .	<code>(wifi_enabled() + app_launched(app_name)) / 2.0</code>

and influence reward outcomes. Similarly to MiniWoB++ [45, 29], ANDROIDWORLD consists of a practically infinite set of varying initial conditions and success criteria. This approach provides more granular analyses of agents’ adaptability — a vital attribute for real-world deployment. Beyond testing agent robustness, the dynamic construction of tasks supports the use of online learning methodologies, particularly reinforcement learning [45, 29, 19, 14]. It also simplifies the generation of distinct train/test datasets, facilitating supervised learning experiments [19, 44, 12].

### 3.4 Durable rewards from system state

ANDROIDWORLD provides reward signals by managing application state using the Android Debug Bridge (adb). With the adb tool ANDROIDWORLD has complete access to system resources including the file system, application databases, and system settings. Determining reward signals from system state has several benefits. It is highly accurate because an application’s state can be quickly inspected and manipulated using the same mechanisms that the app itself utilizes. Using the underlying system state is much more durable than matching superficial UI changes. Additionally, it facilitates easy re-use across disparate apps, which tend to use the same underlying caching mechanisms. For instance, logic for checking existence of a specific file is used across many unrelated applications, including those for file management, note-taking, and media playback. For applications leveraging SQLite databases, a common pattern, ANDROIDWORLD implements evaluators that verify the existence of new and deleted rows. Table 2 shows examples of the validators in ANDROIDWORLD. For more examples see Table 5 in the Appendix.

### 3.5 Task composability

In addition to facilitating accurate and reusable evaluations, inferring a task’s success from system state makes it easy to create *composite* tasks by combining together existing tasks. For example, the task “Create a calendar event with details and text the details to contact” could be created by combining together two existing tasks for creating a calendar event and for sending a text message, which is possible because each task initialization and success detection logic is hermetic. Composite tasks tend to be more challenging because of their complexity, although they provide partial rewards based on completion of sub tasks, to help facilitate hill climbing. The last two rows of Table 2 show the validation code for composite tasks.

### 3.6 Integrating MiniWoB++

To demonstrate ANDROIDWORLD’s extensibility, we implement the MiniWoB++ in the ANDROIDWORLD framework and term it MobileMiniWoB++. Each MiniWoB++ task is instantiated using the standard ANDROIDWORLD interface, inheriting from TaskEval base class, and contains meth-

ods like `initialize_state` and `is_successful`. Since MiniWoB++ leverages JavaScript for task configuration and success detection, we built a WebView app to communicate between Python and the app. For instance, the `is_successful` method of each task retrieves the reward value from the WebView app via an Android intent.

MobileMiniWoB++ introduces modifications in both observations and actions compared to the original benchmark. For example, HTML5 `<input>` elements are natively rendered with native Android UI widgets like the date-picker (see Figure 4 in the Appendix), enhancing the realism of the tasks. MobileMiniWoB++ uses the same observation space as the Android tasks (accessibility tree and screenshot). Notably, it does not include the DOM as in the original implementation. The action space from ANDROIDWORLD is retained. We manually review and test each task to ensure they are solvable. We excluded twelve of the original tasks that failed to render correctly on Android, presented compatibility issues with the touch interface, or required near real-time interaction, which poses challenges on emulators. Overall, ANDROIDWORLD supports 92 MiniWoB++ tasks.

## 4 ANDROIDWORLD as a computer-control benchmark

To test ANDROIDWORLD applicability for autonomous agents, we develop and test a state of the art agent and its variants across all 20 apps and 116 tasks as well as on MobileMiniWoB++.

### 4.1 Computer control agents

#### 4.1.1 M3A

We develop a multimodal autonomous agent for Android, M3A. It is zero-shot, integrating ReAct-style [65] and Reflexion-style [46] prompting to consume user instructions and screen content, reason, take actions, and update its decision-making based on the outcome of its actions.

In the first stage, M3A generates an action, represented in JSON, and reasoning for that action. To generate this output, the agent is provided with a list of available action types, guidelines for operating the phone, and a list of UI elements derived from the Android accessibility tree’s leaf nodes. The agent receives the current screenshot and a Set-of-Mark (SoM) [62] annotated screenshot, which includes bounding boxes with numeric labels on the top-left corner for each UI element (see screenshot in Figure 5 in the Appendix). The agent attempts to execute outputted action by referencing the specific mark (if applicable). In addition to the multimodal agent, we have developed a text-only variant that consumes the screen represented using the accessibility tree and selects the relevant action in JSON format.

After executing an action, M3A reflects on its effect by observing any state changes that may have occurred. During this stage, the agent is provided with available action types, general operating guidelines, the actual action taken, and its reasoning, as well as before-and-after UI states, represented by UI element representations and screenshots with SoM annotations. We request the LLM to provide a concise summary of this step, including the intended action, success or failure, potential reasons for failure, and recommendations for subsequent actions. This summary will serve as the action history and be used for future action selection.

#### 4.1.2 SeeAct baseline

We implement a baseline agent based on SeeAct [70], which was originally designed for GPT-4V for web navigation tasks. Specifically, we implement the best-performing variant of SeeAct<sub>choice</sub>, which grounds actions via textual choices. We implement SeeAct for the Android environment to evaluate how an existing model that performs well on web tasks [9] can be adapted and applied to Android.

To accommodate the Android environment, we adapt SeeAct in several ways. Firstly, we augment the action space from the original SeeAct implementation to support actions needed for mobile, including scroll, long press, navigate home and back, and open app actions. Secondly, in lieu of the DOM, which is not available for Android apps, we utilize the accessibility tree to construct candidate UI actions. Due to the lack of the DOM representation, we do not use the bespoke ranker model from the original implementation. However we observe that after applying a filtering heuristic to remove non-interactable elements, the majority of screens contains less than 50 candidate elements.

Table 3: Success Rates (SR) on ANDROIDWORLD and MobileMiniWoB.

Agent	Input	Base model	SR <sub>ANDROIDWORLD</sub>	SR <sub>MMiniWoB++</sub>
<i>Human</i>	<i>screen</i>	<i>N/A</i>	80.0	100.0
M3A	a11y tree	GPT-4 Turbo	<b>30.6</b>	59.7
M3A	a11y tree	Gemini 1.5 Pro	19.4	57.4
M3A	SoM (screen + a11y tree)	GPT-4 Turbo	25.4	<b>67.7</b>
M3A	SoM (screen + a11y tree)	Gemini 1.5 Pro	22.8	40.3
SeeAct [70]	SoM (screen + a11y tree)	GPT-4 Turbo	15.5	66.1

## 4.2 Experiment results

We evaluate the performance of M3A and SeeAct on the ANDROIDWORLD and MobileMiniWoB++ task suites. For each task, we set the seed to 30 and provide a task-specific step budget that allows for solving the task with additional buffer steps. The agents operate in a zero-shot manner and do not draw from a memory of prior screens, although they maintain a history of prior actions and their effects. We experiment with Gemini 1.5 Pro and GPT-4 Turbo as the underlying base models. For MobileMiniWoB++, similarly to recent work, we evaluate on a subset of 62 tasks [72, 22, 15].

Table 3 presents the success rates (SR) for the agents and human performance on both task suites. Although the agents have far from human performance, they demonstrate out-of-the-box capabilities in operating mobile UIs, exhibiting basic understanding and control capabilities of UIs. They can perform a variety of actions, including long-press, scrolling to search for information, and revising their plan if actions do not work out. The best performance is obtained for M3A when using GPT-4. On ANDROIDWORLD the SoM-based variant is less performant, while on MobileMiniWoB++ it performs best. A similar result was obtained in recent work in the context of computer agents for desktop applications [59]. We posit SoM grounding is less beneficial for ANDROIDWORLD as the accessibility tree for Android apps is generally better populated than for mobile websites and covers the required action space, reducing the value of SoM.

Grounding remains a key challenge for all agents. The agents struggle with precise interactions, such as manipulating text and are often unable to recover from mistyping errors. This is evidenced by agents exhibiting better performance in data editing tasks compared to data entry tasks. As another example, operating sliders proves difficult for agents despite their simplicity for humans.

Screen understanding for mobile UIs remains challenging, with agents failing to detect subtle items that are essential for task completion (see Figure 7a in the Appendix). Additionally, agents struggle with certain UI patterns and affordances, often lacking the capability to sufficiently explore and adapt as humans do. For example, when encountering unfamiliar UI elements or tasks, agents are unable to figure out the correct actions through exploration (see Figure 7c in the Appendix). Moreover, agents sometimes struggle with tasks that simply involve confirming system states, e.g., confirming the WiFi is turned on, suggesting challenges in both task and screen understanding.

Longer tasks are harder for agents, due to increased opportunities to make mistakes. Longer tasks also may require agents to keep track of memory, and perform significant data entry, which can pose difficulties for agents. For these tasks that demand agent memory, such as performing transcriptions across apps or multiplying numbers, or tasks that involve scrolling across entries, the agents struggle as they do have little ability to “remember” what was seen on the prior screens.

Compared to M3A, SeeAct is less performant on the ANDROIDWORLD task suite but performs similarly on MobileMiniWoB++. This stronger performance on the web is expected, as SeeAct was optimized for web environments. SeeAct struggles with operating mobile UIs, particularly with mobile-specific actions such as long-presses and swipes. Additionally, SeeAct sometimes fails to select the correct action based on its generated output, likely because it does not utilize screen elements during action generation.

Beyond difficulties with actions, SeeAct struggles with tasks requiring memory, as it only caches its actions, not their results. For instance, we observed SeeAct getting stuck in loops, issuing ineffective actions like continually scrolling despite reaching the bottom of a page. Scrolling is particularly challenging for SeeAct, as its original implementation did not require this action. Although SeeAct

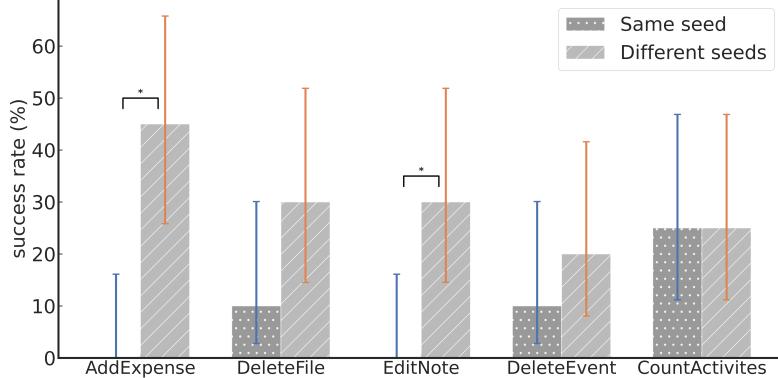


Figure 3: Success rate variation across tasks. Using a fixed seed, the agent appears completely incapable of solving some tasks due to “bad luck” with the seed. In contrast under different task parameterizations, we observe the agent is capable of solving the tasks fairly often. This contrast between the two groups is reflected by higher variance for the different seed group (orange) vs. the same seed group (blue). Significant differences, with  $p$ -value  $< 0.05$ , are indicated by “\*”.

can sometimes recover from errors, it often does not do so quickly enough, leading to termination when the maximum number of steps is reached.

Finally, we also note the use of large foundation models introduces significant latency, and agents are about three times slower than humans to finish a task on average. On average, M3A takes 3.9 minutes to complete a task, with the text-only version taking 2.5 minutes.

### 4.3 Agent robustness under random seeds

We evaluate agent robustness under two conditions: (1) identical tasks with the same parameters and (2) tasks with different parameter combinations, which change the initial state and task definition. Due to computational constraints, we perform this analysis on a representative subset of ANDROIDWORLD tasks (listed in C.4). We use the strongest agent, M3A using the accessibility tree and GPT-4, for this analysis. Our results are shown in Figure 3.

In the baseline experiment using the same seed, the agent is unable to perform the add and edit tasks, and rarely solves the two delete tasks. For the add and edit tasks, the agent struggles with UI operations, while for the delete file task, it often gets confused before the step budget is consumed. Surprisingly, the agent’s performance varies even with a fixed seed, indicating the model’s non-determinism affects agent reliability.

Under varying seeds, agent performance is much more variable, with statistically significant differences observed in the add expense and edit note tasks compared to the same seed group. Interestingly, the agent can solve the tasks fairly often under different seeds, suggesting the model’s sensitivity to task parameters and the potential for improvement via RL-like mechanisms. This experiment demonstrates the importance of supporting parameterizable tasks to better characterize real-world performance of autonomous agents.

Further, the high intra-task variation indicates current agents’ lack of robustness to environmental changes and unreliable performance due to model non-determinism. As observed in RL environments [17, 40, 8], we also observe agent performance is more accurately represented using the mean across random seeds. Finally, we note the non-zero reward observed under some seeds suggests the possibility of future improvements through RL-like mechanisms.

## 5 Limitations

ANDROIDWORLD currently supports tasks with open-source Android apps with at least 1 million downloads and built-in Android system apps. While testing on more trending apps is desirable, we found open-source apps to be equally realistic and, in some cases, more challenging than apps with

larger user bases. Trending apps tend to have UIs which are heavily optimized for smooth user experience, offering more UI functionality and shortcuts. Testing on less-optimized UIs makes the agent’s task harder. In an example failure we reported in the Appendix, the agent needed to delete all notes in a list and failed by repeatedly searching for a “delete-all” button. Instead, an agent with stronger reasoning capabilities would have probably searched once for that functionality, realized it was not available, and deleted the notes one by one.

## 6 Conclusion

We introduced ANDROIDWORLD, a realistic and robust agent environment for Android that enables the development and evaluation of autonomous agents across a wide range of tasks and apps. ANDROIDWORLD provides a reproducible task suite consisting of 116 tasks across 20 apps, with each task dynamically generated using random parameters to challenge agents with millions of unique goals. This dynamic nature not only tests the adaptability of agents for evaluation purposes but also opens up new research opportunities for online learning algorithms in computer control agents.

To showcase ANDROIDWORLD’s effectiveness as a benchmark, we developed and released M3A, a state-of-the-art agent, and established benchmark results on ANDROIDWORLD. We tested our agent using multimodal input and text-only. We found that incorporating multimodality, with Set-of-Mark prompting, does not improve performance on Android tasks and the text-only model achieved the best results. Despite outperforming a web agent adapted for Android, our best-performing agent achieved a success rate of 30.6%, which is much lower than the human success rate of 80.0%. This highlights the need for further research and development to build more reliable and robust computer control agents.

We also observe that agent performance varies significantly across variations in intent parameters, with performance even varying with the same parameters, due to unpreventable model stochasticity. These findings underscore the importance of comprehensive evaluation and the need to improve the reliability of these models under diverse real-world conditions.

ANDROIDWORLD provides a realistic and extensible environment for developing and evaluating agents in the Android ecosystem. By releasing ANDROIDWORLD and establishing benchmark performance with M3A, we aim to accelerate research and development in this area, ultimately leading to the creation of more reliable and robust computer control agents capable of operating effectively in real-world environments.

## References

- [1] AgentGPT, 2024. <https://agentgpt.reworkd.ai/>.
- [2] WebLlama, 2024. <https://webllama.github.io/>.
- [3] Josh Abramson, Arun Ahuja, Federico Carnevale, Petko Georgiev, Alex Goldin, Alden Hung, Jessica Landon, Timothy Lillicrap, Alistair Muldal, Blake Richards, Adam Santoro, Tamara von Glehn, Greg Wayne, Nathaniel Wong, and Chen Yan. Evaluating multimodal interactive agents, 2022.
- [4] William E Bishop, Alice Li, Christopher Rawles, and Oriana Riva. Latent state estimation helps ui agents to reason, 2024.
- [5] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A. Plummer. Mobile app tasks with iterative feedback (motif): Addressing task feasibility in interactive visual environments. *CoRR*, abs/2104.08560, 2021.
- [6] 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 Hebbgen 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. Evaluating large language models trained on code. July 2021.

- [7] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.
- [8] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.
- [9] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web, 2023.
- [10] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-Language models as success detectors. March 2023.
- [11] Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting. *arXiv preprint arXiv:2311.10775*, 2023.
- [12] Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with Instruction-Finetuned foundation models. May 2023.
- [13] Significant Gravitas. AutoGPT. <https://agpt.co>, 2023. <https://agpt.co>.
- [14] Izzeddin Gur, Natasha Jaques, Yingjie Miao, Jongwook Choi, Manoj Tiwari, Honglak Lee, and Aleksandra Faust. Environment generation for zero-shot compositional reinforcement learning, 2022.
- [15] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding html with large language models. *arXiv preprint arXiv:2210.03945*, 2022.
- [16] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.
- [17] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [18] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. *arXiv preprint arXiv:2312.08914*, 2023.
- [19] Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9466–9482. PMLR, 17–23 Jul 2022.
- [20] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*, 2023.
- [21] Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*, 2024.
- [22] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36, 2024.
- [23] Megan Kinniment, Lucas Jun Koba Sato, Haoxing Du, Brian Goodrich, Max Hasin, Lawrence Chan, Luke Harold Miles, Tao R Lin, Hjalmar Wijk, Joel Burget, et al. Evaluating language-model agents on realistic autonomous tasks. *arXiv preprint arXiv:2312.11671*, 2023.

- [24] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.
- [25] Juyong Lee, Taywon Min, Minyong An, Changyeon Kim, and Kimin Lee. Benchmarking mobile device control agents across diverse configurations. In *ICLR 2024 Workshop on Generative Models for Decision Making*, 2024.
- [26] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for Tool-Augmented LLMs. April 2023.
- [27] Tao Li, Gang Li, Zhiwei Deng, Bryan Wang, and Yang Li. A Zero-Shot language agent for computer control with structured reflection. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11261–11274, Singapore, December 2023. Association for Computational Linguistics.
- [28] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile UI action sequences. In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8198–8210. Association for Computational Linguistics, 2020.
- [29] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *6th International Conference on Learning Representations (ICLR '18)*, 2018.
- [30] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.
- [31] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. Learning design semantics for mobile apps. In *Proc. of the 31st Annual ACM Symposium on User Interface Software and Technology, UIST '18*, page 569–579, New York, NY, USA, 2018. Association for Computing Machinery.
- [32] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment, 2023.
- [33] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. WebLINX: Real-World website navigation with Multi-Turn dialogue. February 2024.
- [34] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.
- [35] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2023.
- [36] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.
- [37] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. November 2023.
- [38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [39] Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. April 2024.
- [40] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [41] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023.

- [42] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-Emulated sandbox. September 2023.
- [43] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *arXiv preprint arXiv:2310.11324*, 2023.
- [44] Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. From pixels to UI actions: Learning to follow instructions via graphical user interfaces. May 2023.
- [45] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proc. of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR, 06–11 Aug 2017.
- [46] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.
- [47] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks, 2020.
- [48] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, January 2016.
- [49] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. January 2018.
- [50] Gemini Team. Gemini: A family of highly capable multimodal models, 2023.
- [51] OpenAI Team. GPT-4 technical report, 2023.
- [52] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [53] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafar-ali Ahmed, Tyler Jackson, Shibli Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android, 2021.
- [54] Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan. Ugif: Ui grounded instruction following, 2022.
- [55] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P Agapiou, Max Jaderberg, Alexander S Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019.
- [56] Bryan Wang, Gang Li, and Yang Li. Enabling conversational interaction with mobile ui using large language models. In *Proc. of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23. Association for Computing Machinery, 2023.
- [57] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. 2023.

- [58] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024.
- [59] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osword: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.
- [60] Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. May 2023.
- [61] 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. GPT-4V in wonderland: Large multimodal models for Zero-Shot smartphone GUI navigation. November 2023.
- [62] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023.
- [63] Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023.
- [64] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023.
- [65] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. October 2022.
- [66] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*, 2024.
- [67] Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*, 2024.
- [68] Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents, 2023.
- [69] Ziniu Zhang, Shulin Tian, Liangyu Chen, and Ziwei Liu. MMInA: Benchmarking multihop multimodal internet agents. *arXiv preprint arXiv:2404.09992*, 2024.
- [70] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.
- [71] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and chatbot arena. June 2023.
- [72] Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control, 2024.
- [73] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2023.

## Appendix A ANDROIDWORLD environment

### A.1 Observation Space

In ANDROIDWORLD, the Android screen is represented using a `State` class, which includes the following attributes:

- **Pixels:** An RGB array representing the current screen capture of the device. The screenshot resolution is  $2400 \times 1080 \times 3$ .
- **Accessibility tree:** A raw representation of the accessibility tree.<sup>5</sup> This UI tree provides a detailed snapshot of all UI elements currently displayed on the screen. We utilize an accessibility forwarding app from AndroidEnv [53], which leverages gRPC to transmit the accessibility tree data efficiently to the device.
- **UI elements:** A list of processed UI elements extracted from the children of the accessibility tree. Each `UIElement` contains attributes such as text, content description, bounding boxes, and various state flags (e.g., clickable, scrollable, focused).

Since Android observations and actions are asynchronous, changes resulting from actions may take some time to manifest. Therefore, instead of using an RL-based interface, which assumes a tight coupling between actions and observations, we design an interface for the agent tailored for asynchronous interaction. This interface implements a `get_state` method responsible for capturing the current state of the environment, typically after executing an action. This method includes an optional `wait_to_stabilize` flag, which, when enabled, employs heuristics to ensure the UI elements are not in a transient state, thus providing a stable and accurate snapshot of the environment.

### A.2 Action space

Actions are stored using a Python dataclass (shown below) and are executed using adb. Each action type corresponds to specific ADB commands that interact with the Android environment. For click-based actions (click, double tap, long press), we use ADB to simulate touch events at specified coordinates on the screen. For text input actions, we first focus on the text input field and then use ADB to type the desired text and press the enter button. Navigation actions (home, back) involve sending corresponding key events to the device. Scrolling and swiping actions, which are essentially inverse operations, are both implemented by generating and issuing swipe commands through ADB to simulate these gestures. To launch applications, we use ADB to start the desired app.

```
1 ACTION_TYPES = {
2     "CLICK": "click",
3     "DOUBLE_TAP": "double_tap",
4     "SCROLL": "scroll",
5     "SWIPE": "swipe",
6     "INPUT_TEXT": "input_text",
7     "NAVIGATE_HOME": "navigate_home",
8     "NAVIGATE_BACK": "navigate_back",
9     "KEYBOARD_ENTER": "keyboard_enter",
10    "OPEN_APP": "open_app",
11    "STATUS": "status",
12    "WAIT": "wait",
13    "LONG_PRESS": "long_press",
14    "ANSWER": "answer",
15    "UNKNOWN": "unknown"
16 }
17
18 @dataclasses.dataclass()
19 class JSONAction:
20     """Represents a parsed JSON action.
21
22     # Example
23     result_json = {'action_type': 'click', 'x': %d, 'y': %d}
24     action = JSONAction(**result_json)
25
26     Attributes:
27         action_type: The action type.
28         index: The index to click, if action is a click. Either an index or a <x, y>
29             should be provided. See x, y attributes below.
30         x: The x position to click, if the action is a click.
31         y: The y position to click, if the action is a click.
32         text: The text to type, if action is type.
```

<sup>5</sup>Represented using all current windows; <https://developer.android.com/reference/android/view/accessibility/AccessibilityWindowInfo>

```

33     direction: The direction to scroll, if action is scroll.
34     goal_status: If the status is a 'status' type, indicates the status of the goal.
35     app_name: The app name to launch, if the action type is 'open_app'.
36 """
37     action_type: str
38     index: int = None
39     x: int = None
40     y: int = None
41     text: str = None
42     direction: str = None
43     goal_status: str = None
44     app_name: str = None

```

Listing 1: Pseudo-code representation of the action space.

### A.3 MobileMiniWoB++

Authors manually completed all tasks in MiniWoB++ to verify solvability on a mobile interface. MobileMiniWoB++ differs from MiniWoB++ due to the touch-based interface, which required different approaches for certain tasks. For instance, highlighting text from the `highlight-text` tasks involves using Android's long-press and cursor-moving functionalities. HTML5 `<input>` elements are natively rendered with native Android UI widgets like the date-picker (see Figure 4).

Our implementation of MiniWoB++ contains 92 tasks in total. We exclude the following tasks: `chase-circle` (requires near-realtime movement, unachievable by humans on emulators), `moving-items` (too hard to click in emulator), `drag-cube` (drags will scroll the screen, moving the task out of view), `drag-items-grid` (elements are not interactable on Android), `drag-items` (elements are not interactable on Android), `drag-shapes` (drags will scroll the screen, moving the task out of view), `drag-sort-numbers` (elements are not interactable on Android), `text-editor` (cannot underline everything, weird glitch), `number-checkboxes` (not correctly rendered: only three columns), `use-slider-2` (slider implementation not working), `use-spinner` (slider implementation not working), and `click-menu` (the menu responsiveness breaks and the task does not behave as intended).

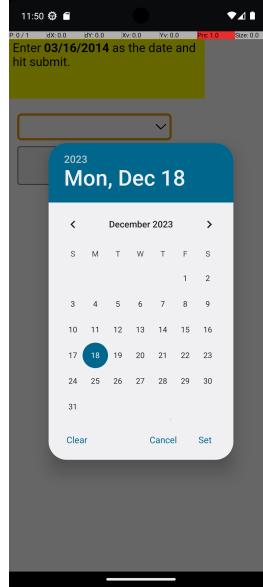


Figure 4: Native Android UI widget rendering for HTML5 `<input>` element.

## Appendix B ANDROIDWORLD Benchmark Details

### B.1 App selection

Our selection of apps (summarized in Table 4) was guided by three main factors: use case, popularity, and the need for consistency and reproducibility.

Table 4: List of ANDROIDWORLD apps and number of tasks for each one.

App name	Description	# tasks
Simple Calendar Pro	A calendar app for creating, deleting, and managing events and appointments.	17
Settings	The Android system settings app for managing device settings such as Bluetooth, Wi-Fi, and brightness.	15
Markor	A note-taking app for creating, editing, deleting, and managing notes and folders.	14
Broccoli - Recipe App	A recipe management app for adding, deleting, and organizing recipes.	13
Pro Expense	An expense tracking app for adding, deleting, and managing expenses.	9
Simple SMS Messenger	An SMS app for sending, replying to, and resending text messages.	7
OpenTracks	A sport tracking app for recording and analyzing activities, durations, and distances.	6
Tasks	A task management app for tracking tasks, due dates, and priorities.	6
Clock	An app with stopwatch and timer functionality.	4
Joplin	A note-taking app.	4
Retro Music	A music player app.	4
Simple Gallery Pro	An app for viewing images.	4
Camera	An app for taking photos and videos.	3
Chrome	A web browser app.	3
Contacts	An app for managing contact information.	3
OsmAnd	A maps and navigation app with support for adding location markers, favorites, and saving tracks.	3
VLC	A media player app for playing media files.	3
Audio Recorder	An app for recording and saving audio clips.	2
Files	A file manager app for the Android filesystem, used for deleting and moving files.	2
Simple Draw Pro	A drawing app for creating and saving drawings.	1

**Use case and categories** We analyzed popular app categories in app stores, focusing on productivity, communication, and multimedia. Selected apps had to meet criteria such as not requiring a login and storing data locally on the device. Additionally, we considered apps from categories that the authors commonly used, ensuring the selection was representative of real-world Android usage.

**Popularity** We used download statistics from the Google Play Store to gauge app popularity, selecting apps with over 1 million downloads. Most of the selected apps exceeded this threshold. Less popular apps were also included if they featured common UI patterns and affordances, ensuring they are indicative of typical Android app usage. For instance, Simple Calendar Pro, though less downloaded, has a UI comparable to the widely-used Google Calendar app.

**Consistency and reproducibility** All apps were sourced from F-Droid, an open-source Android app repository. This allowed us to manage app versions precisely by selecting and distributing specific APKs. We use the newest version of each app at the time of download.

## B.2 Task classification and generation

We categorize tasks into two types: those with side-effects and those without. Tasks with side-effects are those that modify the internal state of the device or applications, such as turning off Wi-Fi or creating a calendar event. These tasks are implemented as distinct Python classes, each with its own parameter generation, initialization, evaluation, and teardown methods.

Below we show an example of the task evaluation for a SendSms task, which involves sending and validating a text message. The pseudocode illustrates the task initialization, success check, and parameter generation methods. Each task has its own random parameter generation method and success logic.

```

1 class SendSms(TaskEval):
2     """Task sending and validating a text message has been sent.
3
4     It checks the SMS telephony database, which is located at:
5     /data/data/com.android.providers.telephony/databases/mmssms.db."""
6
7     template = (

```

```

8     "Send a text message using Simple SMS Messenger to "
9     "{number} with message: {message}"
10    )
11
12    def initialize_task(self, env: interface.AsyncEnv) -> None:
13        """Sets up the initial state of the task."""
14        super().initialize_task(env)
15        clear_sms_database(env.base_env)
16
17    def is_successful(self, env: interface.AsyncEnv) -> float:
18        """Checks if the SMS was sent successfully."""
19        super().is_successful(env)
20        messages = get_messages(env.base_env)
21        return check_message_exists(
22            phone_number=self.params["number"],
23            body=self.params["message"],
24        )
25
26    @classmethod
27    def generate_random_params(cls) -> dict[str, Any]:
28        number = generate_random_number()
29        message = generate_random_message()
30        return {
31            "number": number,
32            "message": message,
33        }

```

### B.3 Information retrieval tasks

Tasks without side-effects are Information Retrieval tasks, requiring the agent to answer a question based on the device or app’s current state. For these tasks, instead of a Python class, we create a protobuf structure to specify the prompt, parameter values, and initialization and validation logic. We decided to use a structured data format with the belief that it would allow us to define new information retrieval tasks by simply adding new entries, making it easier to scale up the number of tasks without needing to write and maintain Python classes for each one.

Initialization is defined per app, including only the state relevant to the prompt’s answer and exclusion conditions for generating random states. This ensures that no random state contains information that could alter the expected answer. The initial state and prompt are parameterized using random values from the specified task parameters. For validation, we define the expected answer format within the prompt and use a few supported functions (“count”, “sum”, “identity”) to generate the answer from the initial state.

Once an app and its specific logic are programmed, new tasks can be generated using an LLM to generate the task’s protobuf. The process is not automatic and requires human review. Common issues with LLM-generated tasks include missing fields, hallucinated fields, incompatible parameter generation, insufficient parameter usage, and non-specific task prompts. We observed that the complexity of the proto structure correlates with an increase in generated task issues. Despite these challenges, we found that editing LLM-generated protobufs can be more efficient than writing a complete task from scratch.

Below we show a simplified version of the task definition for the `SimpleCalendarEventsOnDate` task which involves checking which events are on a certain date. It specifies the relevant event, the exclusion conditions for any noisy event, how to determine success, and possible parameter values to be chosen at random that will be used to fill out the task definition.

```

1 tasks {
2     name: "SimpleCalendarEventsOnDate"
3     prompt: "What events do I have {date} in Simple Calendar Pro? Answer with the titles only. If there
4         are multiple titles, format your answer as a comma separated list."
5     complexity: 1
6     relevant_state {
7         // Defines information for the goal events.
8         state: {
9             calendar {
10                 events {
11                     start_date: "{date}"
12                     start_time: "{time}"
13                     duration: "{duration}"
14                     title: "{title}"
15                 }
16             }
17             // Non-goal events.
18             exclusion_conditions {
19                 field: "start_date"
20                 operation: EQUAL_TO
21                 value: "{date}"
22             }
23         }
24     }
25 }

```

```

22     }
23   }
24   success_criteria {
25     expectations {
26       field_transformation {
27         operation: IDENTITY
28         field_name: "title"
29       }
30       match_type: STRING_MATCH
31     }
32   }
33
34   task_params {
35     name: "time"
36     possible_values: "11:00am"
37     // ...
38   }
39
40   task_params {
41     name: "date"
42     possible_values: "October 15 2023"
43     // ...
44   }
45   task_params {
46     name: "duration"
47     possible_values: "30 m"
48     // ...
49   }
50   task_params {
51     name: "title"
52     possible_values: "Data Dive"
53     // ...
54   }
55 }
```

## B.4 Task examples

Table 5 lists some additional examples of tasks and highlights which task attributes can be parameterized in unlimited ways.

Table 5: Examples of ANDROIDWORLD tasks. We list the task nickname, the task template which highlight which task attributes can be parameterized, the initialization logic that is executed before the task starts and pseudo code describing the success evaluation.

Task nickname	Task template	Initialization logic	Validation code
VlcCreatePlaylist	Create a playlist in VLC, titled "{playlist.name}" with the following files, in order: {files}	Clear app state. Create new mpeg files: files + non_goal_files. Add them to VLC videos folder.	execute_sql(vlc_query) == files
RecipeAddMultipleRecipesFromImage	Add the recipes from recipes.jpg in Simple Gallery Pro to the recipe app.	Clear app state. Write a receipt file with recipes to gallery photos.	sql_rows_exist(expected_recipes)

## Appendix C ANDROIDWORLD agent details

### C.1 M3A observations

ANDROIDWORLD consumes the raw screen pixels, the screen shot with Set-of-Mark (SoM) [62] annotations, and a list of UI elements on screen.

```

1 Here is a list of descriptions for some UI elements on the current screen:
2
3 UIElement0: UIElement(text="VLC", content_description=None, class_name="android.widget.EditText",
4 bbox_pixels=BoundingBox(x_min=98, x_max=886, y_min=146, y_max=311), ...)
5 UIElement1: UIElement(text=None, content_description="Clear search box", class_name="android.widget.
ImageButton",
6 bbox_pixels=BoundingBox(x_min=886, x_max=1023, y_min=160, y_max=297), ...)
7 UIElement2: UIElement(text="15:11", content_description="15:11", class_name="android.widget.TextView",
8 bbox_pixels=BoundingBox(x_min=50, x_max=148, y_min=1, y_max=128), ...)
9 ... More elements listed ...
10
11 ... Guidelines on action selection emitted ...
12
13 Now output an action from the above list in the correct JSON format, following the reason why you do
    that. Your answer should look like:
14
15 Reason: ...
```

```
16 Action: {"action_type":....}
```

Listing 2: The prompt format pertaining to screen representation with UI elements.

## C.2 Agent actions

For the SoM prompting, the screen is annotated based on the UI elements extracted from the accessibility tree, which form the agent’s action space. Figure 5 shows one example.



Figure 5: Set-of-marks overlaid on an Android screen.

## C.3 Error analysis

We analyze agent errors based on broader categories we observe during evaluation.

**Grounding errors** Grounding errors occur when the model fails to correctly interact with the UI elements based on their spatial or contextual positioning.

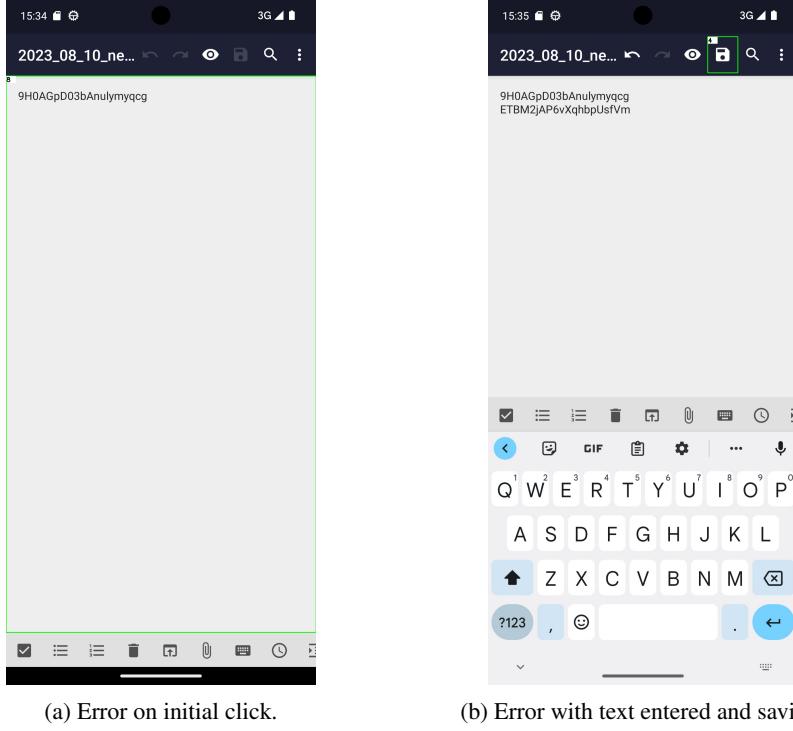
For the task below, the agent needs to update the Markor note by prepending text to the existing text. Figure 6 illustrates the errors the agent makes. It clicks the entire text field area, highlighted in green, which automatically places the cursor after the current text, resulting in the new text being appended after the current content.

*Update the Markor note ‘2023\_08\_10\_neat\_wolf.txt’ by adding the following text, along with a new blank line before the existing content: “ETBM2jAP6vXqhbpUsfVm”, and rename it to ‘sure\_ocean\_uRnI.txt’.*

Then, in the next screen, the text has been entered after the existing content, and the agent clicks the save button.

**Perceptual errors** Perceptual errors are caused when the model fails to recognize crucial elements on the screen necessary for task completion.

For the task below, the model does not recognize that the “All-day” checkbox is currently not checked (see Figure 7a):



(a) Error on initial click.

(b) Error with text entered and saving.

Figure 6: Screens illustrating grounding errors.

*In Simple Calendar Pro, create a recurring calendar event titled ‘Review session for Budget Planning’ starting on 2023-10-15 at 14h. The event recurs weekly, forever, and lasts for 60 minutes each occurrence. The event description should be ‘We will understand software updates. Remember to confirm attendance.’*

**Reasoning errors** Reasoning errors occur when the model misinterprets the task requirements or the current state, leading to incorrect actions.

For example, in the task below, the model mistakenly believes the note name has already been entered, so it types the note text into the “Name” field and cannot recover from this error (see Figure 7b):

*Create a new note in Markor named ipsum\_mNcKd36pDq.txt with the following text: ...*

Here is the agent’s reasoning trace:

*The user wants to create a new note with a specific name and text. The name of the note has already been entered, so all that needs to be done is to enter the text and click on the OK button.*

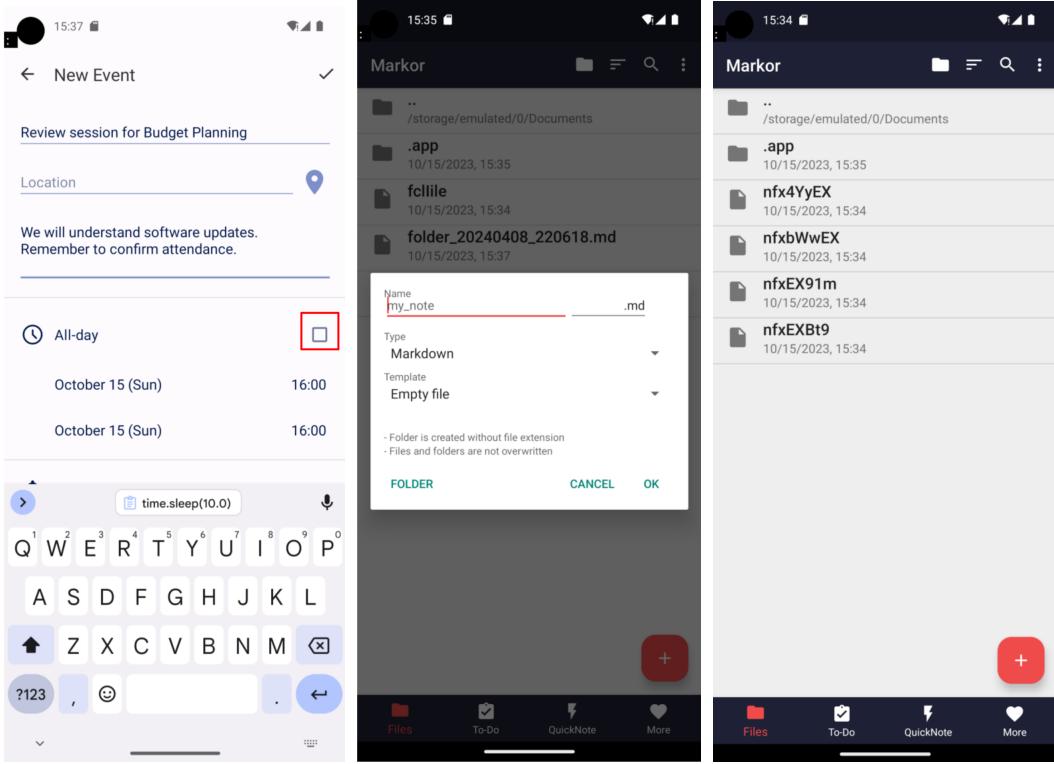
**Missing knowledge errors** Missing knowledge errors occur when the model lacks the necessary understanding of the application’s UI to complete a task efficiently.

For example, in the task below, the agent does not know how to delete all files at once. It looks for an option to do that instead of deleting each file one by one (see Figure 7c):

*Delete all files in the current directory.*

#### C.4 Agent robustness experiments

We ran the agent on the following tasks (the nicknames shown in the figures in parentheses):



(a) Perceptual error. Red square highlights issue.

(b) Reasoning error. The agent's next action is to start entering the delete all notes, the agent mistake's contents, which is incorrect because it needs to enter the note's name first.

(c) Missing knowledge error. To do this, the agent only looks for an option to delete all the notes at once, rather than trying to do it note-by-note.

Figure 7: Screens illustrating perceptual, reasoning and missing knowledge errors.

- `MarkorEditNote (EditNote)`
- `ExpenseAddSingle (AddExpense)`
- `SimpleCalendarDeleteEventsOnRelativeDay (DeleteEvent)`
- `FilesDeleteFile (DeleteFile)`
- `SportsTrackerActivitiesCountForWeek (CountActivities)`