# ANDROIDWORLD: A DYNAMIC BENCHMARKING ENVIRONMENT FOR AUTONOMOUS AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Autonomous agents that execute human tasks by controlling computers can enhance human productivity and application accessibility. However, progress in this field will be driven by realistic and reproducible benchmarks. We present ANDROIDWORLD, a fully functional Android environment that provides reward signals for 116 programmatic tasks 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 more realistic suite of tasks. To ensure reproducibility, each task includes dedicated initialization, success-checking, and tear-down logic, which modifies and inspects the device's system state.

We experiment with baseline agents to test ANDROIDWORLD and provide initial results on the benchmark. Our best agent can complete 30.6% of 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-platform agents. Finally, we also conduct a robustness analysis, showing that task variations can significantly affect agent performance, demonstrating that without such testing, agent performance metrics may not fully reflect practical challenges.

## 1 INTRODUCTION

Autonomous agents that interpret natural language instructions and operate computing devices can provide enormous value to users by automating repetitive tasks, augmenting human intelligence, and accomplishing complex workflows. However, a key research challenge remains the realistic evaluation of these agents in real-world settings. Despite growing enthusiasm for building autonomous agents (Rawles et al., 2023; Deng et al., 2023; Zheng et al., 2024a; Koh et al., 2024; Kim et al., 2024; He et al., 2024; Gravitas, 2023; Age, 2024; Web, 2024; Wu et al., 2023; Xie et al., 2023), most existing approaches for evaluation compare an agent's actions at each step to a previously collected human demonstration (Deng et al., 2023; Rawles et al., 2023; Yang et al., 2023b; Zhang & Zhang, 2023; Lù et al., 2024; Zhang et al., 2024c; Yan et al., 2023; Li et al., 2024). Measuring performance in this way can be misleading because when performing tasks online in real environments agents can take multiple paths to solve tasks, environments may behave non-deterministically, and agents can dynamically learn from mistakes to correct their actions (Shinn et al., 2023; Liu et al., 2018b; Li et al., 2023b; Pan et al., 2024). For this reason, online evaluation of agents in realistic environments able to reward task outcome provides a gold standard for evaluation. While there is an emerging body of work to address this need across different environments (Zhou et al., 2023; Koh et al., 2024; Drouin et al., 2024; Lee et al., 2024; Xie et al., 2024; Bonatti et al., 2024), there is no comprehensive solution for mobile platforms, such as Android, which are used by billions of users and therefore represent environments in which automation agents may be very productively employed. We introduce ANDROIDWORLD to address this.

At its core, ANDROIDWORLD offers a reliable means of obtaining reward signals for tasks performed by agents in realistic mobile environments. 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 that the relevant

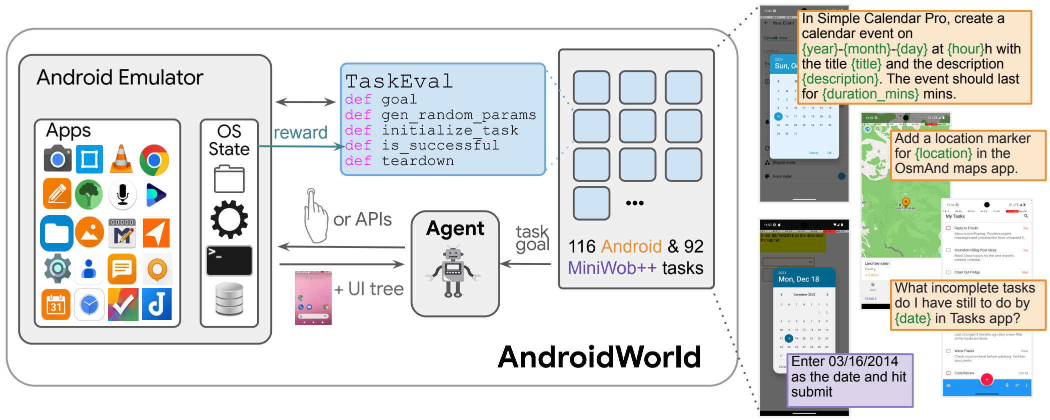



Figure 1: ANDROIDWORLD is an environment for building and testing autonomous agents.

message has been sent. Unlike simulated environments (Tassa et al., 2018; Shridhar et al., 2020) or games (Mnih et al., 2013; Silver et al., 2016; Vinyals et al., 2019; Wang et al., 2023b; Tan et al., 2024; Toyama et al., 2021), real-world apps and websites do not inherently offer explicit reward signals. While human (Rawles et al., 2023; Zheng et al., 2024a; Pan et al., 2024; Kinniment et al., 2023) or LLM-based (Chiang et al., 2024; Zheng et al., 2023; Liu et al., 2023; Du et al., 2023; Ma et al., 2023; Pan et al., 2024; He et al., 2024) judges can be employed to reward the outcome of a task, these approaches scale poorly or are not fully reliable, respectively. Alternatively, environments for autonomous agents which provide automated ground-truth rewards for complex workflows have been developed (Yao et al., 2023; Zhou et al., 2023; Koh et al., 2024; Xie et al., 2024; Bonatti et al., 2024). We find two problems with these environments. First, they are constrained to desktop computing environments, overlooking the mobile domain, which is of paramount importance given the ubiquity and diversity of mobile devices in the real world. Secondly, they are limited in their real-world diversity and scale. Crucially, unlike in real-world scenarios where conditions and task inputs vary widely, these environments support only static test specifications, meaning that when task parameters deviate, the reward signal is likely to break.

We seek to develop a comprehensive benchmark that addresses the limitations of the existing approaches above for evaluating automation agents in mobile environments. ANDROIDWORLD does this by spanning 20 Android apps on a total of 116 programmatic tasks to provide ground truth-rewards. Unlike existing test environments (with MiniWoB++ (Shi et al., 2017) being a notable exception), each task in ANDROIDWORLD 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 leverages actual Android applications. A main challenge that ANDROIDWORLD must address is how to ensure that reward signals are durable when using real-world applications and varying task parameters dynamically. 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 to store and update data.

In addition to providing a comprehensive benchmark, ANDROIDWORLD is lightweight, requiring only 2 GB of memory and 8 GB of disk space, and is designed with convenience in mind. It connects agents to the Android OS by leveraging the Python library AndroidEnv (Toyama et al., 2021) to connect to the freely available Android Emulator.[1] In addition to the 116 Android tasks, we extend ANDROIDWORLD with web tasks by integrating the MiniWoB++ (Shi et al., 2017; Liu et al., 2018a) benchmark into it.

To demonstrate ANDROIDWORLD's usefulness as a benchmark, we build and release a multi-modal 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 observe that while multimodal perception can improve performance in some cases, it gen-

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

erally does not outperform the text-only approach. On ANDROIDWORLD, M3A achieves a 30.6% success rate, which 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 UI control agents, our study includes comprehensive tests under varied real-world conditions, demonstrating significant performance variations primarily driven by changes in intent parameters.

Overall, we make the following contributions: (i) the creation of a new, highly diverse and realistic mobile UI 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 variable task parameters and conditions due to the inherent stochasticity in both models and environments.

## 2 RELATED WORK

Table 1 compares existing evaluation environments for autonomous UI agents.

### 2.1 INTERACTIVE EVALUATION ENVIRONMENTS

Effective evaluation of autonomous agents requires benchmarks that mimic real-world scenarios, but also interactive environments that provide reward signals upon successful task completion (Rawles et al., 2023; Deng et al., 2023; Abramson et al., 2022; Ruan et al., 2023; Chen et al., 2021). Many existing benchmarking environments target web browsing. MiniWoB++ (Shi et al., 2017; Liu et al., 2018b) consists of small, synthetic HTML pages with parameterizable tasks which allow for unlimited task variability. WebShop (Yao et al., 2023) provides a simulated e-commerce environment, whereas WebArena (Zhou et al., 2023) and VisualWebArena (Koh et al., 2024) consist of simulated websites across up to six domains. WorkArena (Drouin et al., 2024) consists of 29 tasks for enterprise software. GAIA (Mialon et al., 2023) is a static dataset that tests an agent's ability to interact with live web environments. MMInA (Zhang et al., 2024e) is a multihop and multimodal benchmark designed to evaluate agents for compositional Internet tasks.

Towards building generalist agents that control operating systems (OSes), OSWorld (Xie et al., 2024) and WindowsAgentArena (Bonatti et al., 2024) provide a test suite of tasks for desktop computer use cases and custom execution-based evaluation scripts across across 9 and 11 apps, respectively. In the mobile domain, existing benchmarks are limited and do not capture the diversity of real-world mobile interactions, containing low-complexity tasks or on a limited number of applications. B-MoCA's (Lee et al., 2024) evaluation is based on 6 simple tasks (e.g., "Call 911", "turn on airplane mode") across 4 apps[2], validated using regular expressions. Mobile-Env (Zhang et al., 2024b) offers task reproducibility limited to 13 task templates for a single app (WikiHow).

While ANDROIDWORLD shares the mobile platform focus of B-MoCA and Mobile-Env, it is more comparable to OSWorld (and WindowsAgentArena, which builds on top of OSWorld) in terms of task complexity and the diversity of interactions it supports. ANDROIDWORLD enhances OSWorld's approach by dynamically constructing the start states of an agent's run and varying the task parameters in unlimited ways, thus allowing for a new type of evaluation under varying real-world conditions.

Other studies leverage human evaluation (Rawles et al., 2023; Zheng et al., 2024a; Bishop et al., 2024) for tasks where automatic evaluation is not available. Lastly, emerging research (Pan et al., 2024; He et al., 2024; Xing et al., 2024) 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.

AndroidEnv (Toyama et al., 2021) provides a mechanism to manage communication with the Android emulator, similar to Playwright and Selenium for web environments. While ANDROIDWORLD leverages this functionality, it diverges in its reward system. AndroidEnv's approach requires modifying application source code and implementing task-specific logging statements, making it well-suited for gaming environments with easily verifiable success criteria. In contrast, ANDROIDWORLD implements a non-invasive reward mechanism, allowing it to create a benchmark suite

[2] Based on what reported in the Experiments Section of the B-MoCA manuscript as of October $1^{st}$, 2024.

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 |
| WEBVOYAGER | ✗ | 15 | 643 | 1 | LLM judge | 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) |
| ANDROIDCONTROL | ✗ | 833 | 15283 | 1 | None | Android (Apps+Web) |
| OMNIACT | ✗ | 60+ | 9802 | 1 | None | Desktop (Apps+Web) |
| ANDROIDARENA | ✗ | 13 | 221 | 1 | Action match/LLM | Android (Apps+Web) |
| LLAMATOUCH | ✗ | 57 | 496 | 1 | Screen match | Android (Apps+Web) |
| MINIWOB++ | ✓ | 1 | 114 | $\infty$ | HTML/JS 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 |
| WORKARENA | ✓ | 1 | 29 | 622.4 | cloud state | Desktop Web |
| MOBILE-ENV | ✓ | 1 | 13 | 11.5 | regex | Android (Apps) |
| B-MOCA | ✓ | 4 | 6 | 1.9 | regex | Android (Apps+Web) |
| MMINA | ✓ | 14 | 1050 | 1 | text-match | Desktop web |
| OSWORLD | ✓ | 9 | 369 | 1 | device/cloud state | Desktop (Apps+Web) |
| WINDOWSAGENTARENA | ✓ | 11 | 154 | 1 | device state | Desktop (Apps+Web) |
| **ANDROIDWORLD** | ✓ | 20 | 116 | $\infty$ | device state | Android (Apps+Web) |

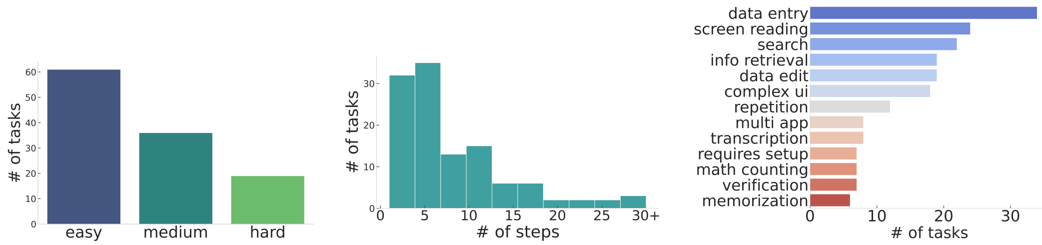
for apps where source code access is unavailable and to reuse validation components across different apps. This approach enables ANDROIDWORLD to cover a broader range of real-world mobile tasks.

### 2.2 STATIC DATASETS FOR UI AUTOMATION

Datasets derived from human interactions provide proxy metrics that correlate with real-world agent performance (Li et al., 2020; Burns et al., 2021; Deng et al., 2023; Rawles et al., 2023). On mobile platforms, AitW (Rawles et al., 2023), AndroidControl (Li et al., 2024), PixelHelp (Li et al., 2020), AndroidArena (Xing et al., 2024), LlamaTouch (Zhang et al., 2024d), UGIF (Venkatesh et al., 2022), and MoTIF (Burns et al., 2021) 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 (Deng et al., 2023), OmniAct (Kapoor et al., 2024) 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 (Li et al., 2023a), ToolTalk (Farn & Shin, 2023), and ToolBench (Xu et al., 2023) 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 (Liu et al., 2018b; Li et al., 2020; Shvo et al., 2021; Gur et al., 2022a; Humphreys et al., 2022). More recent work tends to leverage off-the-shelf foundational models (Team, 2023a;b; Touvron et al., 2023) with in-context learning (ICL) and fine-tuning applied to mobile (Rawles et al., 2023; Hong et al., 2023; Wang et al., 2023a; Yan et al., 2023; Zhang & Zhang, 2023; Bishop et al., 2024; Zhang et al., 2023), desktop web (Zheng et al., 2024a; Deng et al., 2023; Zhou et al., 2023; Koh et al., 2024; Cheng et al., 2024; Lai et al., 2024), and desktop OS (Wu et al., 2024; Zhang et al., 2024a; Xie et al., 2024). Recent work explores agents that reflect on system state (Shinn et al., 2023; Yao et al., 2022; Madaan et al., 2024) by leveraging exploration, self-evaluation, and retry-capabilities enabling continual learning and adaptation (Li et al., 2023b; Yang et al., 2023b; Pan et al., 2024; Wu et al., 2024; Gao et al., 2023; Murty et al., 2024).




(a) (b) (c)

Figure 2: Annotators performed the tasks assigned to them, assigned a difficulty level (2a) and estimated the number of steps required to complete each task (2b), using the action space available to an agent. For each task, they selected relevant category tags from a predefined list (2c).

# 3 AndroidWorld

## 3.1 Android for autonomous agents

Android is an ideal environment for developing autonomous agents. It is the most widely-used OS globally[3] and is highly flexible for research, while providing an open world of the Web[4] and over 2M apps for agents to operate in. Using emulation, an Android environment is easy to deploy, does not require specialized hardware, and can be run on a laptop. Android Virtual Devices or emulator images are well suited for research as they are self-contained, easy to distribute, 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 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 (Toyama et al., 2021) 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 B 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 C 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).

[3] `https://gs.statcounter.com/os-market-share`

[4] Mobile is the most popular platform for accessing the web; `https://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/`

Table 2: Selected tasks with code describing validation logic.

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

To achieve a high degree of reproducibility in real-world scenarios, ANDROIDWORLD precisely controls the OS and app states 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 and consists of both open-source apps and 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.[5] OS-level apps' versions are determined by the Android OS, which is also fixed. To maintain a reproducible environment, ANDROIDWORLD utilizes apps that do not require login/authentication and can 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 C.2 and C.3 for more details), ensuring a fully reproducible suite of tasks.

Automatic task *parameterization* is a critical mechanism, unique to ANDROIDWORLD, to evaluate agents on a much larger and more realistic suite of tasks than current benchmarks support. Achieving this requires significantly more effort than randomly generating new task parameters because it involves developing evaluation logic that remains valid across different task instantiations. It is exactly through its careful state management that in addition to reproducibility AndroidWorld ensures that the reward mechanisms function correctly. Task parameters, initialized randomly at the start of each task based on a controlled random seed, dictate the initial state and influence reward outcomes. Similarly to MiniWoB++ (Shi et al., 2017; Liu et al., 2018a), 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 (Shi et al., 2017; Liu et al., 2018a; Humphreys et al., 2022; Gur et al., 2022a). It also simplifies the generation of distinct train/test datasets, facilitating supervised learning experiments (Humphreys et al., 2022; Shaw et al., 2023; Furuta et al., 2023).

### 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

[5] `https://f-droid.org/`

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.

### 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" was 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++

We implement MiniWoB++ in the ANDROIDWORLD framework and term it MobileMiniWoB++. Each MobileMiniWoB++ task is instantiated using the standard ANDROIDWORLD interface, inheriting from `TaskEval` base class, and contains methods 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 rendered with native Android UI widgets like the date-picker (see Figure 4), 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. See Appendix B.3 for more details.

## 4 ANDROIDWORLD AS A COMPUTER-CONTROL BENCHMARK

To test ANDROIDWORLD's 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 (Yao et al., 2022) and Reflexion-style (Shinn et al., 2023) 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) (Yang et al., 2023a)

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

| Agent | Input | Base model | $SR_{AndroidWorld}$ | $SR_{MobileMiniWoB++}$ |
|---|---|---|---|---|
| *Human* | *screen* | *N/A* | *80.0* | *100.0* |
| M3A | a11y tree | GPT-4 Turbo | **30.6** | 59.7 |
| M3A | a11y tree | Gemini 1.5 Pro | 19.4 | 57.4 |
| M3A | SoM (screen + a11y tree) | GPT-4 Turbo | 25.4 | **67.7** |
| M3A | SoM (screen + a11y tree) | Gemini 1.5 Pro | 22.8 | 40.3 |
| SeeAct (Zheng et al., 2024a) | SoM (screen + a11y tree) | GPT-4 Turbo | 15.5 | 66.1 |

annotated screenshot, which includes bounding boxes with numeric labels on the top-left corner for each UI element (see screenshot in Figure 5). 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 agent 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. See Appendix D for more details on the agent.

#### 4.1.2 SEEACT BASELINE

We implement a baseline agent based on SeeAct (Zheng et al., 2024a), which was originally designed for GPT-4V for web navigation. Specifically, we implement the best-performing variant, $\text{SeeAct}_{\text{choice}}$, 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 (Deng et al., 2023) 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. See Appendix D.5 for more details on the implementation.

### 4.2 EXPERIMENTAL RESULTS

We evaluate M3A and SeeAct on ANDROIDWORLD and MobileMiniWoB++. We set the seed to 30 and provide a task-specific step budget. We use Gemini 1.5 Pro and GPT-4 Turbo as base models. For MobileMiniWoB++, we evaluate on a subset of 62 tasks, consistent with recent studies (Zheng et al., 2024b; Kim et al., 2024; Gur et al., 2022b).

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 (Xie et al., 2024). We posit SoM plays a more critical role in MobileMiniWoB++ tasks due to the often incomplete accessibility tree, compared to that of native Android apps.

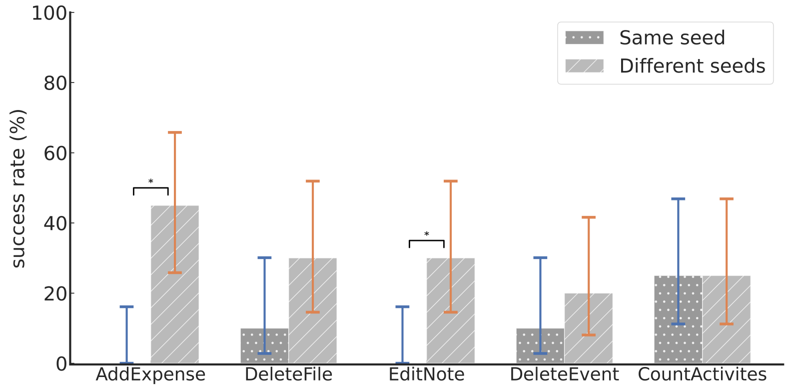



Figure 3: Success rate variation across tasks due to the parametrization built into ANDROIDWORLD. Using a fixed seed, the agent appears completely incapable of solving some tasks due to "bad luck" with the seed. In contrast, under different task parameterizations, we observe the agent is capable of solving the tasks fairly often. Wilson binomial proportion confidence intervals (95%) are shown for the different seed group (orange) and the same seed group (blue). The different seed group has higher variance than the same seed group. Significant differences, with p-value $< 0.05$, are indicated by "*".

### 4.3 ANALYSIS

Agents have difficulty understanding mobile UIs, often failing to detect visual cues that are essential for task completion (see Figure 6a). Additionally, agents struggle with certain UI patterns and affordances, and when they make reasoning mistakes (see Figure 6b), they often lack the capability to explore and adapt as humans do (see Figure 6c). 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.

The agents struggle with grounding, particularly when executing precise interactions, such as manipulating text (see Figure 7) or operating sliders, and they are often unable to recover from mistyping errors. In addition, for tasks that demand memory, such as performing transcriptions across apps, multiplying numbers, or scrolling, the agents struggle as they are unable to "remember" content.

SeeAct performs less effectively than M3A on the ANDROIDWORLD task suite and similarly on MobileMiniWoB++, reflecting its optimization for web rather than mobile environments. It struggles with mobile-specific actions like long-presses and swipes, and often fails to select appropriate actions due to not incorporating screen elements during action generation. Memory-intensive tasks are particularly challenging, as SeeAct only caches actions without remembering outcomes, leading to repetitive, ineffective behaviors such as endless scrolling. This lack of quick error recovery often results in task termination once maximum steps are reached.

Finally, we note that large foundation models significantly increase latency, taking three times longer than humans on average to complete tasks. On average, M3A takes 3.9 minutes to complete a task, with the text-only version taking 2.5 minutes.

### 4.4 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 Appendix D.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 with a constant seed, the agent fails 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 tasks, it often gets confused before the step budget is consumed. Surprisingly, the agent's performance varies even with a fixed seed, suggesting the model's non-determinism affects agent reliability.

Agent performance varies much more using different seeds, with statistically significant differences for the add expense and edit note tasks. The high intra-task variation indicates the model's sensitivity to the seed and its task parameters, stemming from a lack of robustness from the underlying model.

Research in RL environments (Henderson et al., 2018; Raffin et al., 2021; Colas et al., 2018) has noted similar sensitivity to seeds. Consistent with these studies, we observe agent performance is best represented by the mean across random seeds. Notably, the observation of non-zero rewards under some seeds points to potential enhancements through RL-like mechanisms in future work.

This experiment underscores the importance of extensively testing agents under varied task parameters to ensure they can handle real-world variability, a capability that ANDROIDWORLD supports effectively.

## 5 LIMITATIONS

ANDROIDWORLD currently supports tasks from open-source Android apps with at least 1 million downloads and from 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 show in Figure 6c, 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. 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 computer control agents capable of operating effectively in real-world environments. Further, the dynamic nature of ANDROIDWORLD opens up new research opportunities for online learning algorithms in computer control agents.

## REFERENCES

AgentGPT, 2024. https://agentgpt.reworkd.ai/.

WebLlama, 2024. https://webllama.github.io/.

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.

William E Bishop, Alice Li, Christopher Rawles, and Oriana Riva. Latent state estimation helps ui agents to reason, 2024.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows Agent Arena: Evaluating Multi-Modal OS Agents at Scale, 2024. URL https://arxiv.org/abs/2409.08264.

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. URL https://arxiv.org/abs/2104.08560.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. July 2021.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024.

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.

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.

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.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

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.

Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting. *arXiv preprint arXiv:2311.10775*, 2023.

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.

Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, et al. Assistgui: Task-oriented desktop graphical user interface automation. *arXiv preprint arXiv:2312.13108*, 2023.

Significant Gravitas. AutoGPT. https://agpt.co, 2023. https://agpt.co.

Izzeddin Gur, Natasha Jaques, Yingjie Miao, Jongwook Choi, Manoj Tiwari, Honglak Lee, and Aleksandra Faust. Environment generation for zero-shot compositional reinforcement learning, 2022a.

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*, 2022b.

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.

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.

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.

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 (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9466–9482. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/humphreys22a.html.

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.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

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.

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.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. Autowebglm: Bootstrap and reinforce a large language model-based web navigating agent. *arXiv preprint arXiv:2404.03648*, 2024.

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.

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 2023a.

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 (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 11261–11274, Singapore, December 2023b. Association for Computational Linguistics.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on computer control agents. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*, 2024. URL https://arxiv.org/abs/2406.03679.

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*, pp. 8198–8210. Association for Computational Linguistics, 2020. URL https://www.aclweb.org/anthology/2020.acl-main.729/.

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)*, 2018a. URL https://openreview.net/forum?id=ryTp3f-0-.

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, pp. 569–579, New York, NY, USA, 2018b. Association for Computing Machinery. ISBN 9781450359481. doi: 10.1145/3242587.3242650. URL https://doi.org/10.1145/3242587.3242650.

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.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. WebLINX: Real-World website navigation with Multi-Turn dialogue. February 2024.

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.

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.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. November 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. BAGEL: Bootstrapping agents by guiding exploration with language, 2024. URL https://arxiv.org/abs/2403.08140.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. April 2024.

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.

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.

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.

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.

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 (eds.), *Proc. of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3135–3144. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/shi17a.html.

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

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.

Maayan Shvo, Zhiming Hu, Rodrigo Toro Icarte, Iqbal Mohomed, Allan Jepson, and Sheila A. McIlraith. Appbuddy: Learning to accomplish tasks in mobile apps via reinforcement learning, 2021. URL https://arxiv.org/abs/2106.00133.

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. doi: 10.1038/nature16961.

Weihao Tan, Ziluo Ding, Wentao Zhang, Boyu Li, Bohan Zhou, Junpeng Yue, Haochong Xia, Jiechuan Jiang, Longtao Zheng, Xinrun Xu, Yifei Bi, Pengjie Gu, Xinrun Wang, Börje F. Karlsson, Bo An, and Zongqing Lu. Towards General Computer Control: A Multimodal Agent For Red Dead Redemption II As A Case Study. *arXiv preprint arXiv:2403.03186*, 2024.

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.

Gemini Team. Gemini: A family of highly capable multimodal models, 2023a.

OpenAI Team. GPT-4 technical report, 2023b.

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.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android, 2021. URL `https://arxiv.org/abs/2105.13231`.

Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan. Ugif: Ui grounded instruction following, 2022. URL `https://arxiv.org/abs/2211.07615`.

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.

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, 2023a. ISBN 9781450394215. doi: 10.1145/3544548.3580895. URL `https://doi.org/10.1145/3544548.3580895`.

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

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.

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.

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, Leo Z. Liu, Yiheng Xu, Hongjin Su, Dongchan Shin, Caiming Xiong, and Tao Yu. Openagents: An open platform for language agents in the wild, 2023.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 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*, pp. 6061–6072, 2024.

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.

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.

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*, 2023a.

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*, 2023b.

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.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023.

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*, 2024a.

Danyang Zhang, Zhennan Shen, Rui Xie, Situo Zhang, Tianbao Xie, Zihan Zhao, Siyuan Chen, Lu Chen, Hongshen Xu, Ruisheng Cao, and Kai Yu. Mobile-env: Building qualified evaluation benchmarks for llm-gui interaction, 2024b. URL `https://arxiv.org/abs/2305.08144`.

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*, 2024c.

Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. Llamatouch: A faithful and scalable testbed for mobile ui task automation, 2024d. URL `https://arxiv.org/abs/2404.16054`.

Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents, 2023.

Zhuosheng Zhang, Yao Yao, Aston Zhang, Xiangru Tang, Xinbei Ma, Zhiwei He, Yiming Wang, Mark Gerstein, Rui Wang, Gongshen Liu, et al. Igniting language intelligence: The hitchhiker's guide from chain-of-thought reasoning to language agents. *arXiv preprint arXiv:2311.11797*, 2023.

Ziniu Zhang, Shulin Tian, Liangyu Chen, and Ziwei Liu. MMInA: Benchmarking multihop multimodal internet agents. *arXiv preprint arXiv:2404.09992*, 2024e.

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*, 2024a.

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.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control, 2024b.

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 ETHICAL CONSIDERATIONS

**Malicious use** There is a risk that malicious actors could engineer agents to bypass security measures like CAPTCHAs or engage in activities like spamming. Additionally, they could alter prompts or screen outputs to further harmful objectives.

**Societal impact** Automation agents may transform societal norms, disrupt employment, and modify human behavior. While they can enhance efficiency, this improvement could pose risks if exploited by malevolent forces.

# APPENDIX B ANDROIDWORLD ENVIRONMENT

## B.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.[6] This UI tree provides a detailed snapshot of all UI elements currently displayed on the screen. We utilize an accessibility forwarding app from AndroidEnv (Toyama et al., 2021), 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.

## B.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.

[6] Represented using all current windows; `https://developer.android.com/reference/android/view/accessibility/AccessibilityWindowInfo`

```
ACTION_TYPES = {
    "CLICK": "click",
    "DOUBLE_TAP": "double_tap",
    "SCROLL": "scroll",
    "SWIPE": "swipe",
    "INPUT_TEXT": "input_text",
    "NAVIGATE_HOME": "navigate_home",
    "NAVIGATE_BACK": "navigate_back",
    "KEYBOARD_ENTER": "keyboard_enter",
    "OPEN_APP": "open_app",
    "STATUS": "status",
    "WAIT": "wait",
    "LONG_PRESS": "long_press",
    "ANSWER": "answer",
    "UNKNOWN": "unknown"
}

@dataclasses.dataclass()
class JSONAction:
  """Represents a parsed JSON action.

  # Example
  result_json = {'action_type': 'click', 'x': %d, 'y': %d}
  action = JSONAction(**result_json)

  Attributes:
    action_type: The action type.
    index: The index to click, if action is a click. Either an index or a <x, y>
      should be provided. See x, y attributes below.
    x: The x position to click, if the action is a click.
    y: The y position to click, if the action is a click.
    text: The text to type, if action is type.
    direction: The direction to scroll, if action is scroll.
    goal_status: If the status is a 'status' type, indicates the status of the goal.
    app_name: The app name to launch, if the action type is 'open_app'.
  """
  action_type: str
  index: int = None
  x: int = None
  y: int = None
  text: str = None
  direction: str = None
  goal_status: str = None
  app_name: str = None
```

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

### B.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).

## APPENDIX C ANDROIDWORLD BENCHMARK DETAILS

### C.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.

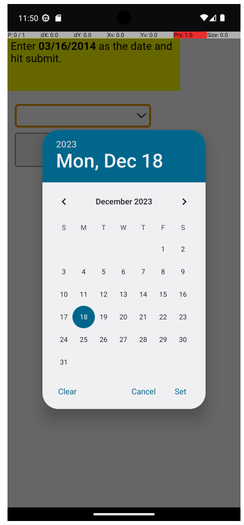



Figure 4: Native Android UI widget rendering for HTML5 <input> element.

**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.

### C.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.

```python
class SendSms(TaskEval):
  """Task sending and validating a text message has been sent.

  It checks the SMS telephony database, which is located at:
  /data/data/com.android.providers.telephony/databases/mmssms.db."""

  template = (
      "Send a text message using Simple SMS Messenger to "
      "{number} with message: {message}"
  )

  def initialize_task(self, env: interface.AsyncEnv) -> None:
    """Sets up the initial state of the task."""
    super().initialize_task(env)
    clear_sms_database(env.base_env)

  def is_successful(self, env: interface.AsyncEnv) -> float:
```

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 |

```
    """Checks if the SMS was sent successfully."""
    super().is_successful(env)
    messages = get_messages(env.base_env)
    return check_message_exists(
        phone_number=self.params["number"],
        body=self.params["message"],
    )

  def teardown(self, env: interface.AsyncEnv) -> None:
    """Clears the SMS database."""
    super().teardown(env)
    clear_sms_database(env.base_env)

  @classmethod
  def generate_random_params(cls) -> dict[str, Any]:
    number = generate_random_number()
    message = generate_random_message()
    return {
        "number": number,
        "message": message,
    }
```

### C.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.

```
tasks {
  name: "SimpleCalendarEventsOnDate"
  prompt: "What events do I have {date} in Simple Calendar Pro? Answer with the titles only. If there are
      multiple titles, format your answer as a comma separated list."
  complexity: 1
  relevant_state {
    // Defines information for the goal events.
    state: {
      calendar {
        events {
          start_date: "{date}"
          start_time: "{time}"
          duration: "{duration}"
          title: "{title}"
        }
      }
    }
    // Non-goal events.
    exclusion_conditions {
      field: "start_date"
      operation: EQUAL_TO
      value: "{date}"
    }
  }
  success_criteria {
    expectations {
      field_transformation {
        operation: IDENTITY
        field_name: "title"
      }
      match_type: STRING_MATCH
    }
  }

  task_params {
    name: "time"
    possible_values: "11:00am"
    // ...
  }

  task_params {
    name: "date"
    possible_values: "October 15 2023"
    //...
  }
  task_params {
    name: "duration"
    possible_values: "30 m"
    // ...
  }
  task_params {
    name: "title"
    possible_values: "Data Dive"
    // ...
  }
}
```

### C.4 Humans for task analysis

During development, we recruited six volunteers with proficient programming skills to analyze task difficulty, duration, and category. Each human was assigned an equal portion of tasks and tasked with identifying bugs during this annotation phase. This process resulted in the discovery and resolution of over 30 bugs.

To evaluate human performance, we enlisted two software engineers to complete the tasks using an Android emulator. Participants were provided with task descriptions and attempted to achieve the goals based on their interpretations. Each participant had one attempt per task. The majority of errors stemmed from misinterpretations or minor errors, such as entering an incorrect file extension. Other errors occurred when participants encountered unfamiliar user interfaces, impeding their ability to solve the tasks on their first attempt.

In both exercises, we informed participants about the intended use of the collected data. Participants were not required to enter any personal information in the tested tasks.

### C.5 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 indicating 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** | **Success evaluation code** |
|---|---|---|---|
| VlcCreatePlaylist | Create a playlist in VLC, titled "{playlist_name}" with the following files, in order: {files} | Create new mpeg files: files + "noise" files that should not be added. Add them to VLC videos folder. | `execute_sql(vlc_query) == files` |
| RecipeAddMultiple RecipesFromImage | Add the recipes from recipes.jpg in Simple Gallery Pro to the recipe app. | Write a receipt file with recipes to Simple Gallery. | `sql_rows_exist(expected_recipes)` |
| MarkorEditNote | Edit {file_name} in Markor. {file_operation}. | Generate file with starting content, along with "noise" files not relevant to goal. *Note:* `file_operation` *can be to add a footer, header, or update note content.* | `file_exists(file_name, content=expected_content)` |
| ExpenseAddSingle | Add the following expenses into pro expense: {expense_csv} | Add to the app's SQLite database the expense that should be deleted, along with "noise" expenses that should not be deleted. | `sql_rows_exist(expense_obj)` |
| SimpleCalendarDelete EventsOnRelativeDay | In Simple Calendar Pro, delete all events scheduled for this {day_of_week}. | add to the app's SQLite database calendar events on specified day, along with "noise" events that should not be deleted. | `!sql_rows_exist(expected_events)` |
| FilesDeleteFile | Delete the file {file_name} from the Android filesystem located in the {subfolder} folder within the sdk_gphone_x86_64 storage area. | Generate specified file, along with "noise" files that should not be deleted. | `!file_exists(file_name)` |
| SportsTrackerActivities CountForWeek | How many {category} activities did I do this week in the OpenTracks app? Express your answer as a single integer. | add to the app's SQLite database activities for the specified category, along with "noise" activities. | `int(agent_response) == expected_count` |

## Appendix D AndroidWorld agent details

### D.1 M3A observations

AndroidWorld consumes the raw screen pixels, the screen shot with Set-of-Mark (SoM) (Yang et al., 2023a) annotations, and a list of UI elements on screen.

```
Here is a list of descriptions for some UI elements on the current screen:

UIelement0: UIElement(text="VLC", content_description=None, class_name="android.widget.EditText",
bbox_pixels=BoundingBox(x_min=98, x_max=886, y_min=146, y_max=311), ...)
UIelement1: UIElement(text=None, content_description="Clear search box", class_name="android.widget.
      ImageButton",
bbox_pixels=BoundingBox(x_min=886, x_max=1023, y_min=160, y_max=297), ...)
UIelement2: UIElement(text="15:11", content_description="15:11", class_name="android.widget.TextView",
bbox_pixels=BoundingBox(x_min=50, x_max=148, y_min=1, y_max=128), ...)
... More elements listed ...

... Guidelines on action selection emitted ...

```

```
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:

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

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

### D.2 M3A 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.

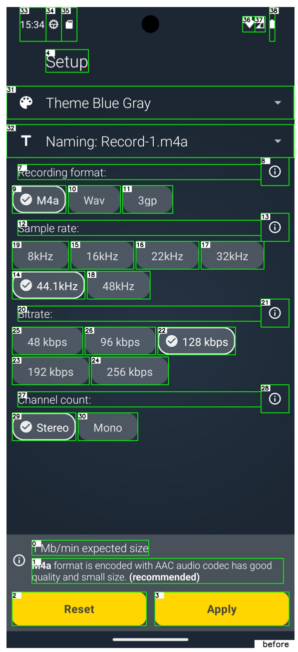



Figure 5: Set-of-marks overlaid on an Android screen.

### D.3 ERROR ANALYSIS

We analyze M3A errors based on broader categories we observe during evaluation.

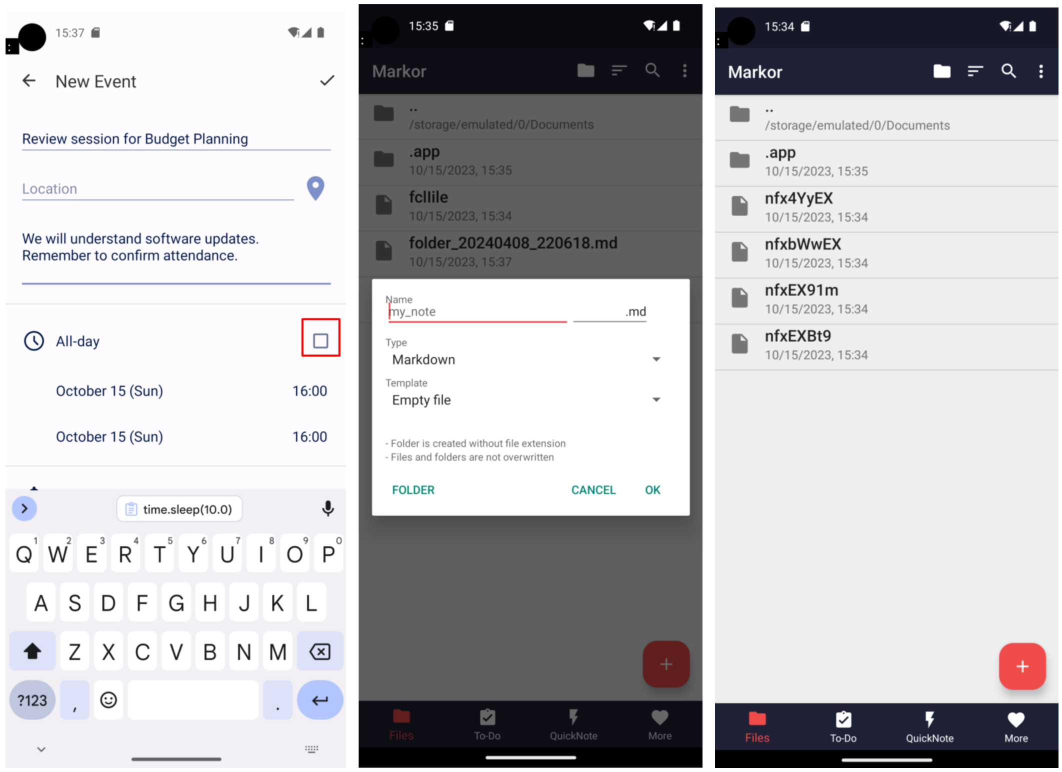
**Perceptual errors** Perceptual errors are caused when the model fails to recognize crucial elements on the screen necessary for task completion.

For the task below, the model does not recognize that the "All-day" checkbox is currently not checked (see Figure 6a):

> *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 6b):




(a) Perceptual error. Red square highlights issue.

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

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

Figure 6: Screens illustrating perceptual, reasoning and missing knowledge errors.

> *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.*

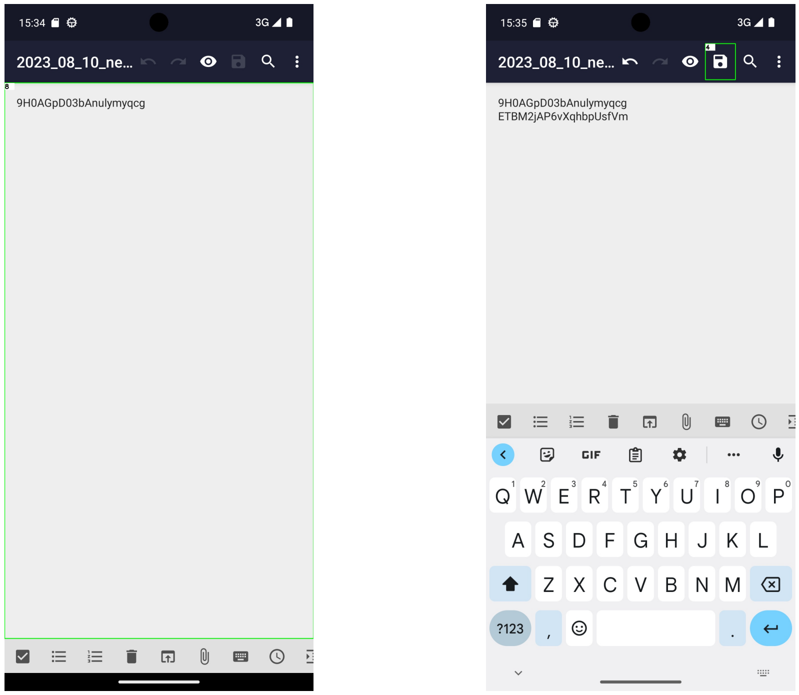
**Missing knowledge errors** Missing knowledge errors occur when the model lacks the necessary understanding of the application's UI to complete a task efficiently.

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 6c):

> *Delete all files in the current directory.*

**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 7 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.




(a) Error on initial click. (b) Error with text entered and saving.

Figure 7: Screens illustrating grounding errors.

> *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.

### D.4 Agent Robustness Experiments

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

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

More details about these tasks can be found in Table 5.

### D.5 SeeAct Details

We modify the SeeAct prompt (Zheng et al., 2024a) to reflect that the environment is Android by inputting elements from the accessibility tree and supporting additional actions (e.g., scrolling). Below we include the updated prompt. We annotate the system, user, and assistant roles that are each provided to the OpenAI API.

```

> Role: SYSTEM
```

```
Imagine that you are imitating humans operating an Android device for a task step by step. At each stage, you can see the Android screen like humans by a screenshot and know the previous actions before the current step decided by yourself through recorded history. You need to decide on the first following action to take. You can tap on an element, long-press an element, swipe, input text, open an app, or use the keyboard enter, home, or back key. (For your understanding, they are like ‘adb shell input tap’, ‘adb shell input swipe’, ‘adb shell input text’, ‘adb shell am start -n’, and ‘adb shell input keyevent’). One next step means one operation within these actions. Unlike humans, for typing (e.g., in text areas, text boxes), you should try directly typing the input or selecting the choice, bypassing the need for an initial click. You should not attempt to create accounts, log in or do the final submission. Terminate when you deem the task complete or if it requires potentially harmful actions.

> Role: USER
You are asked to complete the following task: <GOAL>

Previous Actions:
<PREVIOUS ACTIONS>

The screenshot below shows the Android screen you see. Follow the following guidance to think step by step before outlining the next action step at the current stage:

(Current Screen Identification)
Firstly, think about what the current screen is.

(Previous Action Analysis)
Secondly, combined with the screenshot, analyze each step of the previous action history and their intention one by one. Particularly, pay more attention to the last step, which may be more related to what you should do now as the next step. Specifically, if the last action involved a INPUT TEXT, always evaluate whether it necessitates a confirmation step, because typically a single INPUT TEXT action does not make effect. (often, simply pressing 'Enter', assuming the default element involved in the last action, unless other clear elements are present for operation).

(Screenshot Details Analysis)
Closely examine the screenshot to check the status of every part of the screen to understand what you can operate with and what has been set or completed. You should closely examine the screenshot details to see what steps have been completed by previous actions even though you are given the textual previous actions. Because the textual history may not clearly and sufficiently record some effects of previous actions, you should closely evaluate the status of every part of the screen to understand what you have done.

(Next Action Based on Android screen and Analysis)
Then, based on your analysis, in conjunction with human phone operation habits and the logic of app design, decide on the following action. And clearly outline which element on the Android screen users will operate with as the first next target element, its detailed location, and the corresponding operation.

To be successful, it is important to follow the following rules:
1. You should only issue a valid action given the current observation.
2. You should only issue one action at a time
3. For handling the select dropdown elements on a screen, it’s not necessary for you to provide completely accurate options right now. The full list of options for these elements will be supplied later.

> Role: ASSISTANT
<AGENT RESPONSE TO ABOVE>

> Role: USER
(Reiteration)
First, reiterate your next target element, its detailed location, and the corresponding operation.

(Multichoice Question)
Below is a multi-choice question, where the choices are elements on the screen. All elements are arranged in the order based on their height on the screen, from top to bottom (and from left to right). This arrangement can be used to locate them. From the screenshot, find out where and what each one is on the screen, taking into account both their text content and details. Then, determine whether one matches your target element. Please examine the choices one by one. Choose the matching one. If multiple options match your answer, choose the most likely one by re-examining the screenshot, the choices, and your further reasoning. If you would like to perform a swipe action, you can optionally select the choice where you will swipe.

A. "Home" icon
B. "Phone" icon
C. "Messages" icon
D. "Chrome" icon
E. "Search" icon
...
If none of these elements match your target element, please select Z. None of the other options match the correct element.

(Final Answer)
Finally, conclude your answer using the format below. Ensure your answer is strictly adhering to the format provided below. Please do not leave any explanation in your answers of the final standardized format part, and this final part should be clear and certain. The element choice, action, and value should be in three separate lines.

Format:

ELEMENT: The uppercase letter of your choice. (No need for TERMINATE, KEYBOARD ENTER, WAIT, ANSWER, OPEN APP, NAVIGATE HOME, NAVIGATE BACK; and optional for SWIPE.)

ACTION: Choose an action from {CLICK, INPUT TEXT, LONG PRESS, NAVIGATE BACK, TERMINATE, KEYBOARD ENTER, SWIPE, WAIT, ANSWER, OPEN APP, NAVIGATE HOME}.

VALUE: Provide additional input based on ACTION.
```

```
The VALUE means:
If ACTION == INPUT TEXT, specify the text to be typed.
If ACTION == SWIPE, specify the direction: up, down, left, right.
If ACTION == OPEN APP, provide the name of the app to be opened.
If ACTION == ANSWER, specify the text of your answer to respond directly to a question or request for
      information.
For CLICK, LONG PRESS, KEYBOARD ENTER, NAVIGATE HOME, NAVIGATE BACK, WAIT, and TERMINATE, write "None".
```