

## A Experimental Details

### A.1 Implementation Details

We use Llama-8B-Instruct as our base model. We fine-tune the model using LoRA [17] adapters to minimise computational requirements. The AdamW optimiser [25] is used across all experiments.

During SFT, we update the model parameters for three epochs using the AndroidControl training set. We apply a learning rate of  $10^{-4}$  that linearly decays to 0 throughout the three epochs. We use an effective batch size of 64. The LoRA adapters are configured with 64 dimensions, lora- $\alpha$  of 32, and dropout rate of 0.05.

During RL fine-tuning, we train the model for 15K episodes, equivalent to approximately 200K transitions. This transition count varies between algorithms since those with higher success rates complete episodes more quickly, resulting in fewer overall transitions. For training, we implement data parallelism with 8 concurrent processes. Each process asynchronously interacts with emulators and collects data until accumulating 100 on-policy transitions. These 100 timesteps, combined with 50 transitions sampled from the STR, are used for learning. We perform mini-batch gradient updates with batch size 64 and set the  $\epsilon$  hyperparameter to 0.2. The learning rate for RL fine-tuning remains constant at  $10^{-5}$  throughout training. We conduct a single training epoch for all algorithms except PPO, for which we perform two epochs on the training data.

For value function estimation, we add an extra affine value head on top of the policy’s last hidden layer. We allow value loss gradients to propagate through all trainable model parameters. To balance the policy and value loss functions, we set  $\lambda$  to 0.5.

### A.2 Data and benchmark

#### A.2.1 Action and observation space

In this section, we describe the action space and observation inputs used by our SoLS agent, as well as other RL methods. For the remaining baselines, we adopt their respective action spaces and observation formats, using provided code with only minor modifications where applicable.

The action space of our agent adheres to a standardised JSON-style format, specifying both the action type and any optional parameters: `{"action-type":<type>, "action-extra":<extra>}`. The list of available action-type and action-extra options is detailed in Table 2, and is consistent across both the AndroidControl and AndroidWorld environments. Standardising the action space across the SFT dataset and the RL environment is critical to ensure effective transfer of training. Therefore, actions are consistently converted to and from this format during training and evaluation in both environments. As in prior work [e.g., 9, 20, 28, 38], the `click` and `long-press` actions operate on target UI elements rather than on explicit  $(x, y)$  screen coordinates. This design choice simplifies action generation and increases the likelihood of generating robust, executable actions, especially in the absence of GUI-grounding training. The referenced target element can be translated into an  $(x, y)$  coordinate at execution time using its bounding box.

Action type	Action extras
open-app	<app name>
input-text	<text>
click	<target element>
long-press	<target element>
wait	-
scroll-up	-
scroll-down	-
scroll-left	-
scroll-right	-
navigate-home	-
navigate-back	-

Table 2: Our action space.

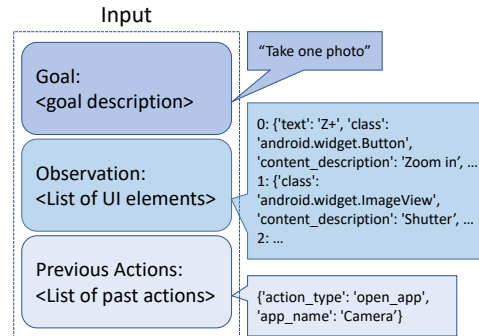


Figure 4: Input to SoLS and other RL methods.

As described in Sections 2.2 and 3.1, both AndroidControl and AndroidWorld provide the task goal, the UI accessibility tree, and a screenshot of the phone interface at each step. Since SoLS and the other RL baselines we evaluate are based on Llama-3-8B and are text-only models, the screenshot is omitted. This significantly reduces the number of input tokens, thereby decreasing the model’s inference time. The UI tree is transformed into a list of UI elements, each represented in a JSON-like format that includes any relevant text or metadata provided by the tree. This list of UI elements is concatenated with the textual goal description to form the initial part of the model’s input. The final component of the input is the history of actions, which is recorded as the agent progresses through each task and appended to the model’s input. A visualisation of the overall observation and input structure is shown in Figure 4.

## A.2.2 Evaluation and benchmark set details

As discussed in Section 2.2, the AndroidWorld benchmark is used for evaluation throughout our experiments. Although the original benchmark consists of 116 tasks, we employ a subset of 80 tasks. Q&A tasks are excluded, as they represent a distinct category not supported by our agents, action space, or by AndroidControl. Verification tasks are also removed, since our evaluation procedure checks for task success at every step, rendering these tasks trivially solvable. Lastly, we exclude tasks that require free-form actions, such as drawing, as these are incompatible with the agents’ current action spaces. Notably, most of the omitted tasks fall under the "easy" difficulty category, resulting in a task distribution that is more challenging than that of the full benchmark suite, as shown in Figure 7.

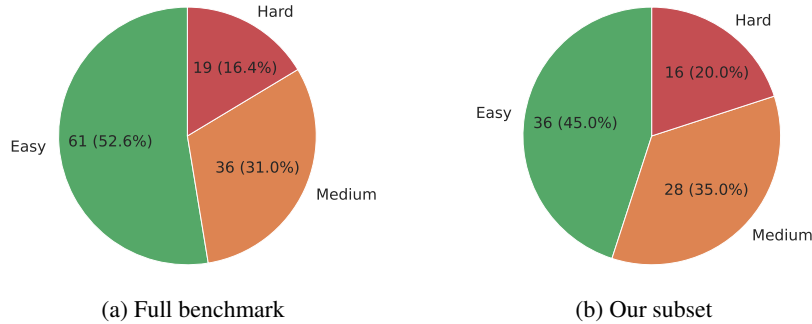


Figure 5: Pie charts comparing task difficulty distribution between the full AndroidWorld benchmark, and the task subset used in this work.

## B Additional Results

### B.1 Inference Time and Model Size

Figure 5b illustrates the trade-off between success rate and inference time across different agents, demonstrating that SoLS not only outperforms the baselines in terms of success rate but also achieves significantly faster inference times. In Figure 6, we further examine the model sizes and inference times of the various agents side-by-side, offering insight into the memory, compute, and latency requirements of each approach. Purely fine-tuned methods, such as OS-Atlas-Pro and our RL implementations, are the least demanding in terms of both model size and inference time. Note that we report the model size of AriaUI as 24.9B, reflecting the total number of parameters, even though only 3.9B are active during inference.

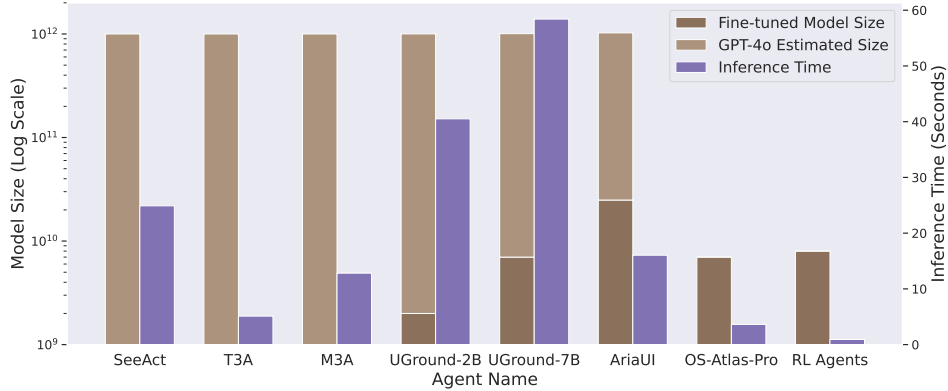


Figure 6: Model size and inference time of different agents. GPT-4o size is estimated at 1 trillion parameters and is shown on top of the fine-tuned grounding model size for mixed prompting-fine-tuned agents.

## B.2 Failure Modes and Case Studies

The main failure modes and an analysis of the types of tasks that SoLS repeatedly fails to solve are introduced in Section 4.4. This section provides further explanation, examples, and illustrations of these failure cases. A representative example for each failure category is included in Table 3.

**Lack of memory.** Some tasks require the agent to retain information across multiple steps. While SoLS receives a history of past actions as part of its input, this provides only limited information about prior states. Due to context-length limitations and computational considerations, the history of observations is not included. Consequently, SoLS struggles with tasks that depend on remembering past information, as illustrated by the example in Table 3. A possible mitigation strategy in future work is the integration of a state summarisation mechanism at each step, which can be fed into subsequent model inputs. A common approach in related works [e.g., 14, 28, 36] is to use GPT-4o as an external summarisation module. However, this incurs the cost of querying GPT-4o at every step.

**Lack of visual input.** SoLS relies solely on text-based inputs, which leads to two primary challenges. First, although UI accessibility trees typically provide high-quality textual representations of elements, they can be inadequate for image-based elements. This is exemplified in Figure 7a, where the maze’s layout and the agent’s position are not described textually, leaving the agent to take essentially random actions. Second, some tasks require extracting information from images, capabilities that a text-only model cannot support. One such case is shown in Table 3. Mitigating this failure mode may require transitioning to a Vision-Language Model (VLM). Alternatively, one could apply Optical Character Recognition (OCR) to extract textual information from image components of the UI element list and incorporate it into the input.

**Unseen interactions.** Tasks in AndroidWorld are out-of-distribution (OOD) relative to the AndroidControl SFT data. Although many tasks can be addressed by generalising to new applications through similar interaction patterns, some require the use of previously unseen device features or unfamiliar actions. For instance, the clipboard functionality, illustrated in Table 3, has never been encountered by SoLS. Given the rarity of the long-press action in the training data (only 0.2%), the agent is unlikely to discover and reinforce correct usage through exploration. A similar issue is presented in Figure 7b, where the agent attempts to append text to a pre-filled field without first clearing it, again requiring a rarely used long-press action. A potential mitigation is to expand the SFT dataset to include a broader variety of tasks and interactions, or to inject exploratory behaviours from other models during RL training.

**Long-horizon tasks.** Certain tasks in AndroidWorld involve lengthy action sequences. Specifically, nine tasks have optimal trajectories exceeding 20 steps, and three exceed 30. One particularly difficult example in Table 3 requires a 60-step optimal solution. Such tasks are inherently difficult for any agent, as the probability of making a critical error increases with sequence length. This is especially

Table 3: Example failed tasks for each failure category.

Failure Mode	Example
Memory	Open the file task.html in Downloads in the file manager; when prompted open it with Chrome. Then click the button 5 times, remember the numbers displayed, and enter their product in the form.
Visual Input	Add the recipes from recipes.jpg in Simple Gallery Pro to the Broccoli recipe app.
Unseen Interactions	Copy the following text to the clipboard: {clipboard_content}
Long Tasks	Save a track with waypoints Ruggell, Liechtenstein, Bendern, Liechtenstein in the OsmAnd maps app in the same order as listed.

problematic in sparse reward settings, where the agent only receives a positive reward upon full task completion. In such scenarios, RL agents can only reinforce successful strategies after discovering a full solution, which is highly improbable given the length of these tasks. While difficult to fully mitigate, incorporating memory mechanisms (as discussed in the first failure mode) could help improve performance on these long-horizon tasks.

**Combined failure modes.** Many of the most challenging tasks fall under multiple failure categories simultaneously. This is particularly true for tasks involving visual input, which often also require memory to retain image-derived information over multiple steps. In fact, many such tasks are additionally long-horizon, making them exceptionally difficult under current system limitations.

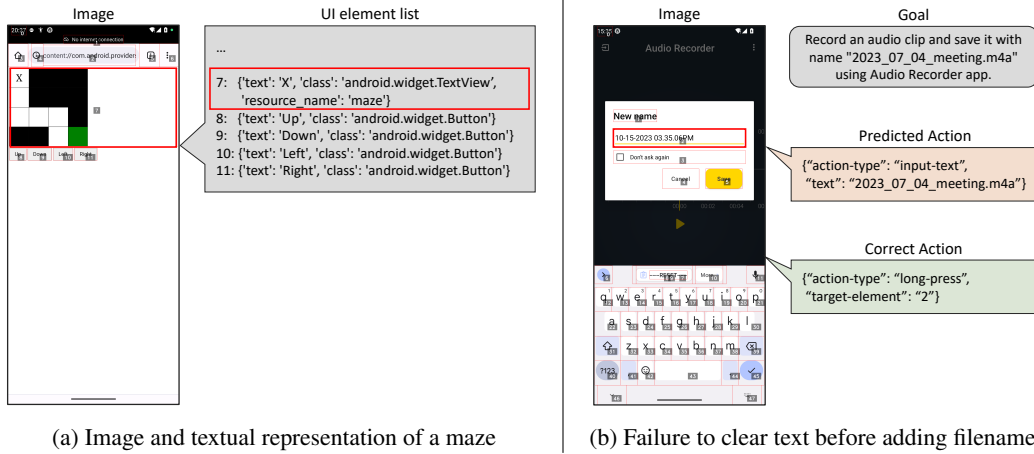


Figure 7: Two examples illustrating the cause of failure cases. (left) The textual representation of the maze, highlighted in red, does not describe the content of the maze visible in the image. (right) Having never seen pre-filled text fields, the agent tries to input the filename, instead of trying to clear it first, with the long-press action.