AppVLM: A Lightweight Vision Language Model for Online App Control

Georgios Papoudakis[1*], Thomas Coste[1*], Jianye Hao[1,2], Jun Wang[3], Kun Shao[1†]

[1]Huawei Noah's Ark Lab, [2]Tianjin University, [3]AI Centre, University College London
{georgios.papoudakis1,thomas.coste,haojianye,shaokun2}@huawei.com
jun.wang@cs.ucl.ac.uk,

Abstract

The utilisation of foundation models as smartphone assistants, termed app agents, is a critical research challenge. These agents aim to execute human instructions on smartphones by interpreting textual instructions and performing actions via the device's interface. While promising, current approaches face significant limitations. Methods that use large proprietary models, such as GPT-4o, are computationally expensive, while those that use smaller fine-tuned models often lack adaptability to out-of-distribution tasks. In this work, we introduce AppVLM, a lightweight Vision-Language Model (VLM). First, we fine-tune it offline on the AndroidControl dataset. Then, we refine its policy by collecting data from the AndroidWorld environment and performing further training iterations. Our results show that AppVLM achieves the highest offline action prediction accuracy in AndroidControl, compared to all evaluated baselines, and matches GPT-4o in online task completion success rate on AndroidWorld, while being up to ten times faster. This makes AppVLM a practical and efficient solution for real-world deployment.

1 Introduction

The development of smartphone assistants using foundation models is an open research challenge. These assistants, which we refer to as app agents, should be capable of executing human instructions on a smartphone, interacting with apps through the same interface as a human user. The user provides a textual description of a goal, and the app agent must take a sequence of actions to successfully complete the task. Such technology has the potential to revolutionise smartphone interactions, providing significant business value by enabling automation for productivity tools, customer service, and accessibility features. Moreover, it could enhance smartphone accessibility for a wider range of users, including individuals with disabilities or those less familiar with digital interfaces.

Two primary approaches have been explored for developing app agents. The first relies on large foundation models, such as GPT-4, combined with prompt engineering methods to solve tasks. While flexible, this is expensive, both in financial resources and execution time; making real-world deployment impractical. The second approach focuses on fine-tuning smaller models (e.g., Bai et al., 2024; Ma et al., 2024; Christianos et al., 2024; Wang et al., 2024c), typically using an offline dataset and, in some cases, incorporating online-collected trajectories. While these methods demonstrate promising results, many evaluations are limited to offline action predictions or online tasks drawn from the same distribution as the training dataset. However, findings from Chen et al. (2024) suggest that when these models are tested in out-of-distribution (OOD) settings, their success rates drop significantly. This highlights a critical challenge in generalising beyond the training distribution.

In this work, we propose AppVLM, an app agent designed to overcome these challenges by achieving both efficiency and strong generalisation to tasks OOD compared to the original offline dataset. Our model is lightweight, enabling fast and cost-effective inference for real-time execution, and capable of adapting to OOD tasks, unlike standard offline-trained models. To achieve this, we assume access

* First authors with equal contribution.
† Corresponding author.

to an offline dataset of near-optimal human trajectories of phone interactions, which are used for Supervised Fine-Tuning (SFT) as an initial step on top of a pretrained vision-language model (VLM). This allows the model to become familiar with the observations and actions required for interacting with an Android phone. We then introduce a Reinforce Fine-Tuning (RFT) pipeline, consisting of data collection, utilising a distributed client-server architecture to balance resources and promote efficiency, followed by offline fine-tuning, where the collected data is used to refine decision-making capabilities. Using this pipeline, we iteratively fine-tune our model, which we refer to as AppVLM.

Our main contributions are summarised as follows:

- We develop AppVLM, the first lightweight (3B) VLM agent capable of successfully solving tasks in the AndroidWorld environment.
- We outperform GPT-4o baselines and fine-tuned models on both the in-domain-data (IDD) and OOD AndroidControl test sets, achieving state-of-the-art results on this dataset to the best of our knowledge.
- We demonstrate that AppVLM achieves performance comparable to GPT-4o baselines in AndroidWorld, exceeding some, and only coming 4% short of the best-performing one, while operating at a fraction of GPT-4o's cost in both time and resources. Additionally, to the best of our knowledge, it outperforms all non-proprietary and fine-tuned models.

By striking a balance between efficiency and generalisation, AppVLM provides a practical and scalable solution for real-world app agents, bridging the gap between foundation models and robust smartphone automation.

2 Related Work

2.1 Prompt Engineering Agents

Several recent works focus on developing agents that execute actions in smartphone or desktop environments in order to complete textual commands. With the advancement of foundation models, the research community has been exploring ways to leverage the general cross-domain knowledge of these pretrained models for app control. Yang et al. (2023); Wang et al. (2024b) were some of the first works that utilised large foundation models to perceive smartphone observations and generate human-like actions. To successfully solve more complex tasks requiring long-term planning and history awareness, several frameworks were proposed with dedicated prompt-engineering components for steps like planning, reflection, etc. (Wang et al., 2024a; Wang & Liu, 2024; Song et al., 2024). Although these added reasoning steps improved performance considerably, they significantly increased the computational cost and wall-time of each interaction. Other works tried to obtain app-specific knowledge utilising memory (Wen et al., 2023; Lee et al., 2024), which stores past interaction between the agent and specific apps.

2.2 Fine-Tuned Agents

To address the gap between the general capabilities of foundation models and the specific needs of smartphone environments, as well as to reduce the cost of querying general foundation models, several works have focused on fine-tuning to implement more specialised app agents. Wang et al. (2024d); Gou et al. (2024) use large foundation models for the high-level proposal of actions or plans, while they fine-tune a smaller VLM to ground this action. Ma et al. (2024) proposed CoCoAgent, a small foundation model that aims to predict actions for app control in smartphones by decomposing the actions into action type prediction and optionally the target UI element that this action will be applied to. Similarly, LiMAC (Christianos et al., 2024) introduced a small action transformer to predict the action type and the target UI element, while integrating a fine-tuned VLM for text completion. InfiGUIAgent (Liu et al., 2025) proposed a two-stage fine-tuning process, which first focuses on learning details about the screenshot observations, such as predicting the text of specific UI elements, and then learns how to generate actions based on user's instructions.

Previous research has also investigated online optimisation of app agents to overcome the limitations of trajectory diversity in static datasets. DigiRL (Bai et al., 2024) introduced an online RL framework that simulates app control tasks, training a policy that is first fine-tuned on an offline dataset. DistRL

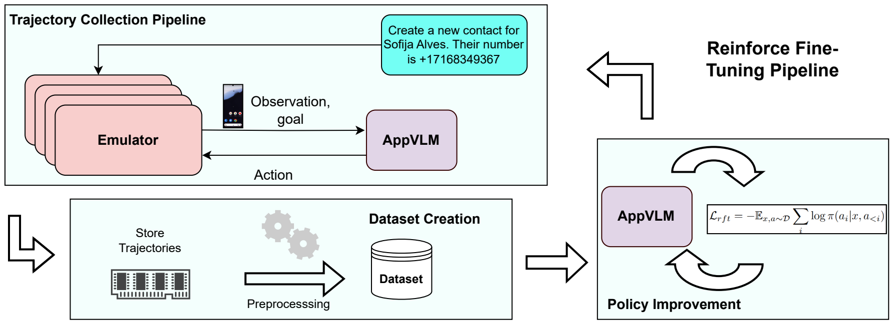

Figure 1: RFT pipeline visualisation. Data is gathered by interactions between the emulators and AppVLM, preprocessed and added to the dataset. A fine-tuning step can then be performed.

(Wang et al., 2024c) enhanced the training efficiency with asynchronous online learning. However, both methods depend on online tasks that follow the same distribution as the offline dataset. In contrast, our work aims to enable agents to tackle tasks beyond those encountered during the initial SFT within the offline dataset.

3 Methodology

3.1 Problem Formulation

We define the app control task as a Goal-conditioned Partially Observable Markov Decision Process (GPOMDP), represented as $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{G}, R, T, \Omega)$. Here, $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{O}$ is the set of observations, and $\mathcal{G}$ is the set of goals. The function T describes the state transition dynamics, and Ω represents the observation probability distribution. The reward function is denoted by R. We assume an agent with a parameterized policy π_θ, where θ represents the policy parameters. Our objective is to optimize the following expression:

$$\max_{\pi_\theta} \mathbb{E}_{g\sim\mathcal{G}} \left[\sum_{t=0}^{H-1} \gamma^t r_t \right],$$

where r_t is the reward at time step t of the episode and H is the horizon of the episode. For simplicity, we assume $\gamma = 1$ in this task. The reward function returns 1 when the episode terminates successfully, and 0 otherwise. To run our experiments, we specifically use the AndroidWorld environment (Rawles et al., 2024), which consists of parametrised tasks to be solved in an online fashion. For example, the task of adding a contact might be described as “Create a new contact for Sofija Alves. Their number is +17168349367.” In this case, the parameters are the first name, surname, and phone number, allowing for a vast number of task variations. Our objective is to train an agent that can solve as many tasks as possible, using a lightweight model that can generalise effectively within the parameter space.

3.2 Supervised Fine-Tuning

Before initiating any online interactions in the AndroidWorld environment, we first perform SFT on a VLM using the AndroidControl dataset (Li et al., 2024), to allow the model to learn essential Android phone interactions. We use the Paligemma-3B-896 (Beyer et al., 2024) as our base model for several reasons. First, with 3 billion parameters, it offers a good balance of performance and efficiency, making it lightweight enough for mobile device deployment, especially when quantised to lower precision. Furthermore, Paligemma-3B-896 downscales images from their original resolution of 2400x1080 pixels to 896x896 pixels. This preserves important visual details, such as legible text, while supporting higher accuracy in tasks that require visual comprehension. In contrast, many CLIP-based (Radford et al., 2021) vision transformers typically downscale images to $224x224$ pixels, a reduction that results in the loss of fine-grained details, making it difficult to retain important visual details and hindering task success. Paligemma-3B-896 has been fine-tuned for computer vision tasks, and is therefore not inherently capable of executing app control commands based on textual instructions. As such, the SFT step in this work is essential for adapting the model to execute app-specific tasks within AndroidWorld.

The input for Paligemma is constructed as follows: For each observation, we use a screenshot annotated with bounding boxes and a label indicating the UI element number for each interactive item. This information is available in the UI tree of the observation, which can be extracted from both the AndroidControl dataset and any Android device. In addition to the visual data, the textual input includes the specified goal and the history of actions. A more detailed explanation of the observation preprocessing can be found in Sections 4.1 and 4.2. To reduce computational costs during both training and inference, we avoid including the full history of observations. Instead, we only include the history of recent actions, as this provides valuable context with minimal added token complexity. Similar approaches have been explored in previous research (Putta et al., 2024).

3.3 Reinforce Fine-Tuning

After fine-tuning the agent on the AndroidControl dataset, we deploy it within an interaction and fine-tuning pipeline using the AndroidWorld environment. We refer to this procedure as Reinforce Fine-Tuning (RFT), also known as Reinforced Self-Training (ReST) (Gulcehre et al., 2023), or iterative SFT with rejection sampling. Please note, that RFT should not be confused with the on-policy REINFORCE algorithm (Willams, 1992). Collecting data and fine-tuning in this way is an essential step to enable the agent to adapt to tasks in AndroidWorld, which differ from the training data provided by AndroidControl. The RFT pipeline consists of two steps, executed sequentially: (1) data collection and preprocessing and (2) fine-tuning our model, AppVLM. These two steps of data collection and policy improvement using the SFT loss correspond to the ”grow” and ”improve” steps of the ReST algorithm (Gulcehre et al., 2023). In contrast to ReST, our RFT does not use the original offline dataset during the policy improvement.

3.3.1 Data Collection and Preprocessing

To facilitate efficient and scalable data collection, we implement a distributed client-server architecture, illustrated in Figure 1. In this setup, the Android emulator acts as the client, while the AppVLM agent operates as the server. Tasks, along with their associated parameters and agent settings, can be submitted to a shared “task queue”. When an emulator client becomes idle, it retrieves a task from the queue and executes it. The script running the emulator will request action generations from the AppVLM server at every step, providing the current environment observation. These requests are maintained in an “action request” queue, to be processed sequentially by the AppVLM server. Generated actions are then returned and executed in the client emulator, enabling it to proceed to the next step. This distributed client-server architecture enables us to run several emulators in parallel, efficiently pulling tasks from the same queue and calling AppVLM from different machines.

To ensure diverse data collection, each task in AndroidWorld is initially added several times to the task queue, with high sampling temperatures. Then, we identify tasks that have been solved fewer than a threshold τ number of times and repeat, focusing on these tasks. The process is repeated for multiple rounds, ensuring that more challenging tasks receive increased attention and maintaining a broad representation of all tasks. Each successful trajectory is stored locally and preprocessed before being added to the training dataset. However, early trajectories, particularly those collected before fine-tuning on AndroidWorld, often contain erroneous actions, even when successful. To reduce these errors, we apply a filtering step, comparing consecutive screenshots and removing the earlier timestep if the observation has not changed. This prevents redundant or erroneous action sequences from being reinforced during fine-tuning. After data collection and preprocessing, the training dataset is constructed by oversampling tasks solved fewer than τ times. This guarantees that the model is not biased toward tasks that were easier to solve during data collection, helping it generalise more effectively across different types of interactions. We set $\tau = 10$, such that a diverse number of samples can be used for training, but not so high that the oversampling might become extreme.

3.3.2 RFT Policy Improvement

After collecting a dataset that contains successful AndroidWorld trajectories, we fine-tune AppVLM to enhance the agent’s ability to solve a broader range of tasks. The optimisation objective follows the standard maximum likelihood objective weighted by a *return* term, where successful trajectories are assigned a return of one, and unsuccessful ones receive a return of zero. In practice, trajectories with zero return are excluded from the fine-tuning dataset. The RFT optimisation objective is:

$$\mathcal{L}_{rft} = -\mathbb{E}_{x,a\sim\mathcal{D}_{on}} \sum_i \log \pi(a_i|x, a_{<i}) \tag{1}$$

where $\mathcal{D}_{on}$ is the gathered preprocessed dataset.

3.4 THE TRAINING ARCHITECTURE

Having outlined the individual training steps, this section provides a summary of the entire training pipeline, presented in Algorithm 1. As described in Section 3.2, the process begins with SFT on the pretrained Paligemma model. The resulting model, referred to as AppVLM-base, can interact with Android emulators and generate actions in the correct format. However, it faces challenges in online interactions. To address this, the model undergoes further refinement using RFT on data generated within the AndroidWorld environment. This pipeline alternates between collecting trajectories and fine-tuning the model by maximising the likelihood of actions that led to successful task completions. The policy improvement phase of the RFT is conducted offline to allow recalibrating the dataset by removing duplicate observations and to ensure a more balanced distribution of data across tasks. This approach prevents over-representation of simpler tasks, which could cause the model to overfit. In our experiments, we use three iterations of RFT. As a final step, all data collected throughout the entire RFT procedure is used to fine-tune the agent of the initial SFT stage, AppVLM-base, using the standard maximum likelihood objective. In Section 4.6, we present an ablation study demonstrating that this approach achieves superior performance in both online and offline settings compared to fine-tuning directly the output of the RFT pipeline.

4 EXPERIMENTS

4.1 ANDROIDWORLD ENVIRONMENT

AndroidWorld (Rawles et al., 2024) offers a benchmark of 116 unique tasks with randomised parametrisation, leading to an infinite number of task instances. It provides a real Android environment in which agents can attempt to solve these app control tasks, evaluated with ground-truth reward signals based on the phone's state. This facilitates and standardises the execution of actions, and allows for a realistic and fair evaluation of app agents. AndroidWorld leverages the AndroidEnv (Toyama et al., 2021) library to allow communicating with an emulated Android virtual device.

Actions: AndroidWorld uses a fixed action space, as action grounding is a standard practice in app control tasks, to translate model outputs to environment actions. The action space that is used by AppVLM is very similar to the one used in AndroidWorld and only minor action translations are required. Appendix A.1 further discusses our action space and AndroidWorld conversions.

Observations: AndroidWorld observations consist of a phone's screenshot and an accessibility UI tree that provides details about UI elements, including their text content, type, position, and attributes (e.g., whether they are clickable). We use the accessibility tree to identify clickable elements and determine their bounding box coordinates. These coordinates are then drawn as overlays on the screenshot, with numbered labels for clarity. Each task in the AndroidWorld environment includes a high-level textual instruction, or goal, that defines the agent's objective. This goal, along with the agent's past actions, is provided as textual input to AppVLM alongside the annotated screenshot. Further details on how we process AndroidWorld observations are available in Appendix A.2.

Agent: To operate within the AndroidWorld environment, the VLM must be encompassed within an agent. For a given task, at each timestep, the agent receives the current goal and observation from the environment. These are fed into the model, using the observation processing described above, to generate a new action. We then convert this action into the expected format and execute it in the environment, updating the phone state. This procedure continues until either the the task is solved, for which AndroidWorld provides a reward signal dependent on phone state, or the agent fails to solve it in the allotted number of steps.

Table 1: Comparison across agents of action accuracy in the four splits of the AndroidControl test set.

	Agent	Input Type	Action Accuracy ↑			
			IDD	Task-Unseen	Cat-Unseen	App-Unseen
GPT-4o	SeeAct	screen + UI tree	31.5	30.7	30.6	30.9
	T3A	UI tree	56.1	55.8	56.5	54.2
	M3A	screen + UI tree	60.8	59.3	60.8	60.4
Fine-Tuned	Llama-3	UI tree	65.5	58.7	58.3	57.1
	LT-all-r64*	UI tree	70.8	59.6	57.4	58.5
	AppVLM-base	screen + b-boxes	**73.9**	**65.9**	**65.1**	**65.4**
	AppVLM	screen + b-boxes	69.0	62.7	61.9	62.2

4.2 AndroidControl Dataset

Before the RFT in AndroidWorld, we perform an SFT step on AndroidControl (Li et al., 2024), due to its similarities with AndroidWorld. AndroidControl is an open-source app control dataset that contains a large number of human demonstrations for a wide range of phone navigation tasks, from setting alarms to adding items to a shopping basket.

Importantly, episodes in the AndroidControl dataset present themselves similarly to trajectories from AndroidWorld tasks. Each episode contains a textual goal along with a list of per-step observations and corresponding human-selected actions. Much like AndroidWorld, observations are composed of both a screenshot of the current phone screen and a UI tree. In line with our processing of AndroidWorld observations, we use the UI tree information to annotate screenshots with bounding boxes and associated number labels. Moreover, AndroidControl actions are again grounded to a fixed action space, which, other than minor discrepancies discussed in Appendix A.1, is identical to AndroidWorld's. As described in Section 3.2, the final model input is formed by combining the task goal and actions of the previous five steps in text format, and the annotated screenshot.

4.3 Evaluated Baselines

GPT-4o methods: T3A and M3A (Rawles et al., 2024) were introduced alongside AndroidWorld and are widely used as reference points. They are based on the same two-step prompting method: summarising the previous action in one step, and generating an action based on the current observation and history summary in the other. T3A is text-only, receiving observations as a list of UI elements and descriptions based on the UI accessibility tree, while M3A also receives screenshots of the phone screen annotated with UI element bounding boxes and labels. Additionally, we include SeeAct (Zheng et al., 2024), another popular two-step GPT-prompting method. Specifically, we use the $\text{SeeAct}_{\text{choice}}$ variant, as in Rawles et al. (2024), since this has been found to be the best-performing (Zheng et al., 2024). In this variant, GPT-4o is first given the task and screenshot and prompted to produce a high-level description of the proposed action. The next step is an action grounding step, where a multiple-choice list of UI elements is provided, along with the first-step action proposal and details about expected action formats, and GPT-4o is tasked with producing the final action output.

Fine-tuned models: We also include smaller models, fine-tuned on AndroidControl, as evaluation baselines. We fine-tuned the 8B-parameter Llama-3 model (Dubey et al., 2024) using a similar observation format as AppVLM, but instead of using screenshots as input, we provide a condensed textual form of the UI tree. To reduce computational requirements, LoRA (Hu et al., 2021) adapters are used. In addition, for AndroidControl, we include the action prediction accuracy of the LT-all-r64 model as reported by Li et al. (2024). LT-all-r64 is PALM-2S fine-tuned using LoRA adapters, and achieves the highest accuracy in Li et al. (2024). To the best of our knowledge, it is highest reported accuracy to this day on AndroidControl. It is important to note that this comparison may not be entirely consistent. While we have made every effort to faithfully reproduce their evaluation protocol, minor differences could impact the comparison with AppVLM. Since the LT-all-r64 model is unavailable, we do not include it in online experiments. We do however similarly report the success rate of InfiGUIAgent (Liu et al., 2025) in AndroidWorld as stated in its original paper. Methods with results taken directly from their papers are marked with an asterisk (*) in the result tables.

Table 2: Comparison of different agents in terms of success rate in the AndroidWorld environment.

	Method	Size ↓	Average Infer. Time (s)↓	Success Rate ↑			Overall Success Rate ↑
				Easy	Medium	Hard	
GPT-4o	SeeAct	-	15.82	34.2	15.5	4.2	22.0
	T3A	-	4.29	**64.9**	26.2	**14.6**	**41.9**
	M3A	-	11.42	60.5	20.2	8.3	36.6
Fine-Tuned	Llama-3	8B	2.35	31.6	6.0	4.2	17.5
	InfiGUIAgent*	2B	-	25.0	0.0	0.0	9.0
	AppVLM-base	3B	0.91	21.9	2.4	2.1	11.4
	AppVLM	3B	0.91	57.9	**27.4**	8.3	37.8

4.4 Experimental Setup

Our experiments focus on two evaluations: an offline evaluation of the action prediction accuracy in AndroidControl, and an online evaluation of the success rate in the AndroidWorld environment tasks. Details about these can be found in Sections 4.1 and 4.2 respectively. Here we discuss specifically how we conduct evaluation in these settings.

AndroidControl: Each timestep is a datapoint, composed of a goal, observation, and an action. Models are tasked with generating an action, which will be compared against the ground truth. Fine-tuned models are trained to provide the appropriate action format, while GPT-4o methods are provided with a large prompt detailing the format, as in Rawles et al. (2024). A relaxed action prediction accuracy is reported for all methods, whereby a click target is considered correct as long as its bounding box is within the target element, following previous works (Li et al., 2024).

AndroidWorld: Online evaluation is performed, where agents are tasked with taking steps until a task is either solved or the maximum number is steps is reached. Task success is evaluated at every step using the provided reward signal, and a task is considered unsuccessful if the maximum number of steps is reached. In addition to overall success rate, we report per-difficulty success rates, using the task difficulty information provided by the benchmark. Due to the nature of our agents, action space, and evaluation process, certain tasks are omitted from the evaluation, notably verification and Q&A tasks. Discussed further in Appendix A.3, our final benchmark consists of 82 tasks, with a harder difficulty distribution than the full 116-task benchmark. In our AndroidWorld experiments, we perform evaluation across three different seeds, leading to different task parameters (e.g. contact name), and report the average performance across runs.

4.5 Results and Analysis

Table 1 shows the action accuracy of all methods on the four splits of the AndroidControl test set. We find that AppVLM-base, which is fine-tuned only on AndroidControl, outperforms all baselines as well as AppVLM. It is important to highlight that AppVLM-base achieves the best action accuracy on this task, surpassing the previous benchmark achieved by LT-all-r64. Even AppVLM achieves comparable accuracy to LT-all-r64 in IDD test set, and higher accuracy in OOD test splits.

The decline in action accuracy of AppVLM compared to AppVLM-base is expected, as the final SFT step relies solely on new data from the AndroidWorld environment. This shift reduces model accuracy in AndroidControl. A key factor in this decline is the rigidity of AndroidControl action accuracy evaluation. For example, in AndroidControl, the trajectory typically includes a `wait` action after performing `open-app`. In contrast, AndroidWorld introduces an automatic two-second delay between actions, eliminating the need for an explicit `wait` action. During the online dataset preprocessing, these `wait` actions are usually removed, as they do not affect the phone's screenshot. Finally, AppVLM also outperforms Llama-3 in AndroidControl, which may indicate that for the specific task of predicting actions that match the ground truth, the image may be more informative. A similar pattern is observed when comparing M3A with T3A, providing further evidence that visual information plays a crucial role in action prediction.

Table 3: Comparison of RFT iterations.

Agent	AndroidControl	AndroidWorld
AppVLM-RFT_1	72.5	17.9
AppVLM-RFT_2	71.0	23.2
AppVLM-RFT_3	66.0	30.5
AppVLM-RFT_4	64.3	35.0

Table 4: Comparison of AppVLM ablations.

Agent	AndroidControl	AndroidWorld
AppVLM	69.0	37.8
AppVLM-RFT_4	64.3	35.0
AppVLM-AWO	29.8	22.4

Table 2 presents the success rate of the online evaluation of AppVLM and related baselines in AndroidWorld across three different difficulty levels. AppVLM achieves performance comparable to M3A/T3A while requiring significantly fewer resources, both in cost and computation time. Indeed, it exceeds both SeeAct and M3A's performance, while coming only 4% short of T3A's performance. Moreover, its average inference time is a fraction of GPT-4o's, with AppVLM being more than 10 times faster than SeeAct and M3A, and almost 5 times faster than T3A. We also emphasise that AppVLM achieves the highest AndroidWorld success rate among all fine-tuned models.

AppVLM-base shows strong performance in AndroidWorld, despite being fine-tuned only on AndroidControl. Llama-3, which has also been fine-tuned exclusively on AndroidControl, exhibits comparable results. This suggests that Llama-3 could serve as an alternative to Paligemma as the base model for AppVLM. However, Paligemma remains the preferred choice, as it is almost three times smaller, enabling much faster inference. Interestingly, we observe that higher accuracy in AndroidControl does not always lead to a higher success rate in AndroidWorld, even for models fine-tuned on the same data. For example, Llama-3 outperforms AppVLM-base and T3A outperforms M3A in AndroidWorld, but the opposite is true in AndroidControl.

4.6 Additional Studies

In this section, we provide additional studies to further explain the design choices of AppVLM. First, in Table 3, we present the action accuracy and success rate in AndroidControl and AndroidWorld respectively for different iterations of RFT. We observe that RFT plays a crucial role in improving AppVLM's success rate in AndroidWorld, with a linear increase over three iterations. However, further iterations of RFT showed that the improvement in success rate was lower compared to performing SFT in AppVLM-base, as evidenced in Tables 1 and 4. Additionally, the action accuracy in AndroidControl drops across RFT iterations, as previously discussed in Section 4.5.

We also provide an analysis of how the final SFT step influences both offline and online performance (see Table 4). We compare AppVLM, against AppVLM-RFT_4, and AppVLM-AWO (AndroidWorld Only), which skips SFT on AndroidControl entirely, performing SFT on top of the pretrained Paligemma model using the collected AndroidWorld dataset. Our results show that AppVLM-RFT_4 suffers from lower action prediction accuracy in AndroidControl compared to AppVLM. This follows the downward trend observed in Table 3 over successive RFT training steps. Similarly, its AndroidWorld success rate is lower than that of AppVLM. We hypothesise that AppVLM-RFT_4's performance saturates as it starts to overfit to the simpler tasks. AppVLM-AWO, on the other hand, performs poorly in offline evaluations since it has not been fine-tuned on AndroidControl. Its online success rate is also relatively low, because it has not learned basic Android interactions that would have been acquired through SFT on AndroidControl, which also represents a much more significant amount of training data than the collected AndroidWorld dataset. By applying the final SFT step on AppVLM-base, we retain high action prediction accuracy on AndroidControl while achieving the highest success rate on AndroidWorld tasks compared to the fine-tuning baselines.

4.7 Case Study and Failure Analysis

For illustration purposes, we show examples of AndroidWorld trajectories in Figure 2 and Appendix C. Figure 2 demonstrates a failure case for AppVLM. In this trajectory, the agent almost completes the task correctly, but fails to clear the existing text before adding the filename in the penultimate step. This is a common mistake, where the agent does most of the task correctly but forgets to perform one minor action. This often happens because the AndroidControl dataset used for the initial fine-tuning does not contain tasks where such a step, for example clearing a text field, is required. In addition, because our model is relatively small, it may lack certain intuition or reasoning needed to realise

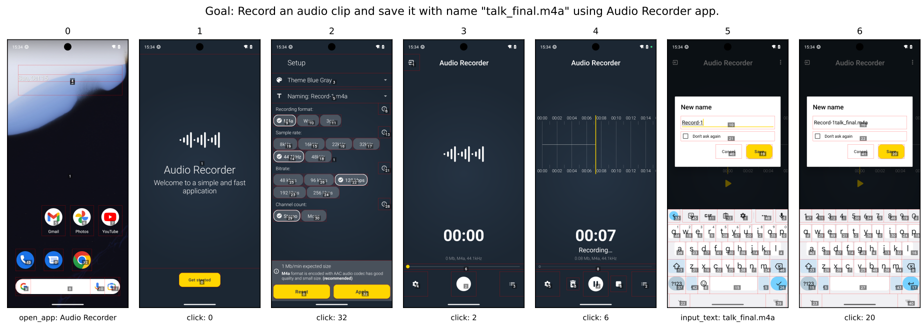

Figure 2: Example trajectory in AndroidWorld, with the goal at the top and the taken actions below each timestep's screenshot.The agent almost succeeds in solving this task, but forgets to clear the text field before typing in the penultimate step.

it must do this. Therefore, RFT can help mitigate this behaviour if the agent learns to perform the missing action during data collection. In the next iteration, this action will be reinforced and the agent will solve the task more frequently. For example, our final model learns to delete the existing text in the task from Figure 2, as seen in Figure 4. However, this type of failure still occurs in our agent for tasks that it has never managed to solve, and thus struggles to learn which step could be missing. Nevertheless, our iterative fine-tuning learns to solve many previously unsolved tasks, even taking key actions which might be absent or extremely rare in the initial fine-tuning AndroidControl dataset, such as `long-press` (see Appendix C).

5 Conclusion

In this work, we introduced AppVLM, the first lightweight VLM capable of successfully solving online tasks in AndroidWorld. We present a complete pipeline for fine-tuning a pretrained VLM to efficiently tackle goal-based tasks on Android smartphones. Our results demonstrate that AppVLM-base achieves the highest AndroidControl action prediction accuracy, compared to all baselines. Moreover, in online evaluations within AndroidWorld, AppVLM delivers performance comparable to, and in some cases exceeding, agents that rely on GPT-4o, while requiring significantly less time and computational resources. Notably, AppVLM can compute actions up to ten times faster than GPT-4o agents, making it an efficient alternative for app control.

Despite its strong performance, AppVLM has limitations stemming primarily from the constraints of its training data. The model struggles with tasks involving operations it has never encountered, such as using the phone's "clipboard". Since it has not been exposed to the concept of a clipboard, it fails to recognise and execute related actions. Addressing these gaps requires expanding the scope of training data to better capture app control tasks. A promising direction is integrating a broader range of mobile interactions during pretraining, such as UI element detection, UI tree generation, etc. Existing datasets such as AndroidControl and AitW (Rawles et al., 2023) provide valuable benchmarks, but they lack a unified format. For example, AitW does not include UI trees and focuses more on generalisation across Android versions. To advance this field, the community should prioritise the creation of a large-scale, standardised dataset tailored specifically for app control.

Another crucial challenge is data generation. Currently, most datasets rely on human demonstrations, a process that is expensive, time-consuming, and impractical at scale. However, automatically generating trajectories is limited by the lack of reward functions for such tasks. AndroidWorld is the only app control environment that provides an internal reward function. Other approaches leverage large foundation models (e.g., GPT-4o) to evaluate trajectories, but these methods are slow, costly, and highly sensitive to prompt variations, making them unreliable for systematic evaluation. To overcome these challenges, we believe that the development of dedicated reward models for app control is necessary. Recent studies have explored using models as reward functions (Ma et al., 2022; Chan et al., 2023), yet no robust, and app control-specific, reward model has been proposed. Such a model would enable scalable evaluation and unlock new possibilities for RL in app control.

REFERENCES

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *arXiv preprint arXiv:2406.11896*, 2024.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

H Chan, V Mnih, F Behbahani, M Laskin, L Wang, F Pardo, M Gazeau, H Sahni, D Horgan, K Baumli, et al. Vision-language models as a source of rewards. In *Second Agent Learning in Open-Endedness Workshop*, 2023.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, et al. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Filippos Christianos, Georgios Papoudakis, Thomas Coste, Jianye Hao, Jun Wang, and Kun Shao. Lightweight neural app control. *arXiv preprint arXiv:2410.17883*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 1119–1133, 2024.

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. *arXiv preprint arXiv:2406.03679*, 2024.

Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. *arXiv preprint arXiv:2501.04575*, 2025.

Ilya Loshchilov, Frank Hutter, et al. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5, 2017.

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. Comprehensive cognitive llm agent for smartphone gui automation. *arXiv preprint arXiv:2402.11941*, 2024.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, 2021.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36, 2023.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

Zirui Song, Yaohang Li, Meng Fang, Zhenhao Chen, Zecheng Shi, Yuan Huang, and Ling Chen. Mmac-copilot: Multi-modal agent collaboration operating system copilot. *arXiv preprint arXiv:2404.18074*, 2024.

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. *arXiv preprint arXiv:2105.13231*, 2021.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024a.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024b.

Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agents. *arXiv preprint arXiv:2410.14803*, 2024c.

Xiaoqiang Wang and Bang Liu. Oscar: Operating system control via state-aware reasoning and re-planning. *arXiv preprint arXiv:2410.18963*, 2024.

Yiqin Wang, Haoji Zhang, Jingqi Tian, and Yansong Tang. Ponder & press: Advancing visual gui agent towards general computer control. *arXiv preprint arXiv:2412.01268*, 2024d.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 543–557, 2023.

Ronald J Willams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

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

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

A Datasets and Environment

This section presents additional information and examples about our datasets and environment.

A.1 Action Space

As introduced in Section 4.1, AppVLM has a fixed action space. This helps standardise actions for training and grounding the model's outputs into valid actions. Our action space is presented in Table 5, along with example actions as they are expected to be generated by AppVLM. The action spaces of AndroidWorld and AndroidControl are very similar, with only minor naming differences, as well as a couple action alterations. AndroidWorld includes a `keyboard-enter` action which we omit, since it is not present in AndroidControl and thus our initial fine-tuning. AndroidControl also includes a click target as part of its `input-text` action, while we choose to keep these as separate actions as in AndroidControl.

Table 5: Action space, along with example actions for each type.

Action	Example
`open-app` + <app-name>	`{"action-type":"open-app","app-name":"Clock"}`
`click` + <target-element>	`{"action-type":"click","target-element":1}`
`long-press` + <target-element>	`{"action-type":"long-press","target-element":1}`
`input-text` + <text>	`{"action-type":"input-text","text":"Hello World"}`
`scroll-{up/down/left/right}`	`{"action-type":"scroll-up"}`
`navigate-home`	`{"action-type":"navigate-home"}`
`navigate-back`	`{"action-type":"navigate-back"}`
`wait`	`{"action-type":"wait"}`

The main difference between our action space and that of our evaluation environments lies in our use of click targets for `click` and `long-press` actions, rather than x-y coordinates. In such cases, we translate the index of the target element into the centre coordinates of its bounding box, leveraging the provided UI tree information in both AndroidWorld and AndroidControl. Finally, we ensure actions are converted into the specific format expected by either AndroidWorld or AndroidControl, so that actions are correctly executed.

To train on the AndroidControl dataset, it is also necessary to convert the x-y coordinates for ground-truth `click` and `long-press` actions to target element indices corresponding to bounding box labels on the screenshot, as expected by our model. The UI element tree information is used to select the best candidate element in this case, and actions which our model can train on are obtained.

A.2 Observation Space

Sections 4.1 and 4.2 introduced the observation processing performed on AndroidWorld and AndroidControl respectively. An example observation from AndroidWorld is illustrated in Figure 3, though an observation from AndroidControl would be essentially identical. As previously described, this observation contains both a visual input, the annotated screenshot, and a textual input, composed of the goal and history of actions.

The history of actions provides crucial context for the current state and offers options for error recovery and mitigation. To reduce computational costs, with the objective of creating a lightweight agent, we limit the size of this history to only the five most recent actions. Additionally, the target element index component of `click` and `long-press` actions is not very informative as part of this history once the timestep's screenshot is no longer observable. Therefore, the agent stores an alternate representation of actions in its history, sourced from the UI tree data. A condensed textual representation of the target element is used, containing information such as the type of object and its textual content or description, as can be seen in the textual input of Figure 3.

A.3 AndroidWorld Benchmark Set

While the full AndroidWorld benchmark consists of 116 tasks, we use a reduced subset of 82 tasks for our experiments. Firstly, we remove the verification tasks, such as

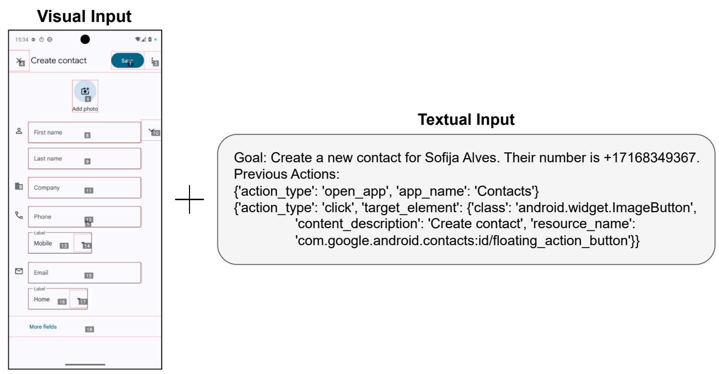

Figure 3: Example AndroidWorld observation passed as input to AppVLM. The visual input is composed of the current screenshot, annotated with bounding boxes surrounding clickable UI elements, along with numbered labels. The textual input is composed of the task goal, as well as the history of actions. This observation corresponds to the input for step 2 in Figure 5.

`ClockStopWatchPausedVerify`, because we check whether tasks have been successfully completed at each timestep and these tasks would automatically succeed. We also remove all Q&A tasks because they tackle a separate type of task, and are outside of AndroidControl's action space, and thus ours. Finally, we remove the drawing tasks since agents are not equipped for drawing with the current fixed directional scroll actions. The resulting subset of AndroidWorld is used for all agent evaluations, and actually has a higher difficulty distribution than the full set, as shown in Table 6. As demonstrated by the table, this is because we removed proportionally more "easy" tasks than "medium" or "hard" tasks.

Table 6: AndroidWorld benchmark count and distribution of tasks per difficulty.

Benchmark	Easy (%)	Medium (%)	Hard (%)	Total
Full Benchmark	61 (52.6%)	36 (31.0%)	19 (16.4%)	116
Our Subset	38 (46.3%)	28 (34.1%)	16 (19.5%)	82

B Implementation Details

In this section we discuss the implementation details of AppVLM. Algorithm 1 illustrates the pseudocode for our training process.

Algorithm 1 Pseudocode of AppVLM training pipeline

```
# Initial VLM Fine-Tuning
AppVLM-base ← SFT(VLM, 𝒟)
𝒟_on ← ∅, AppVLM-RFT_0 ← AppVLM-base
for i ← 0 to N do
    # Collect data from AndroidWorld
    𝒟_on ← 𝒟_on ∪ CollectData(AndroidWorld)
    # Improve the AppVLM Policy
    AppVLM-RFT_i+1 ← RFT(AppVLM-RFT_i, 𝒟_on)
end for
# Perform a final SFT on AppVLM-base
AppVLM ← SFT(AppVLM-base, 𝒟_on)
return AppVLM
```

As we already discussed we use Paligemma-3b-pt-896 as our base model. All fine-tuning rounds, both for the initial SFT, the RFT, and the last SFT steps use the AdamW optimiser (Loshchilov et al., 2017) with 3×10^{-6} learning rate. The learning rate is gradually reduced to zero during the course of the training. Additionally we fine-tune always for three epochs and we use effective batch size of 64. We perform full fine-tuning of the model without using any adapters.

C Case Studies

Sample AndroidWorld trajectories from our final model are illustrated in the following figures. Figure 4 shows our agent correcting the audio recorder task from Figure 2. In this example, AppVLM successfully deletes the existing text before typing the filename. This is particularly impressive because the model has learned to generate a `long-press` action to do so, an action which is extremely rare in the initial AndroidControl dataset, featuring less than 1% of the time. It shows the merit of our RFT pipeline, which enables the model to teach itself behaviour it does not have initially. This happens when an agent successfully explores during the data collection phase and the advantageous interaction is reinforced by the rejection sampling and subsequent training.

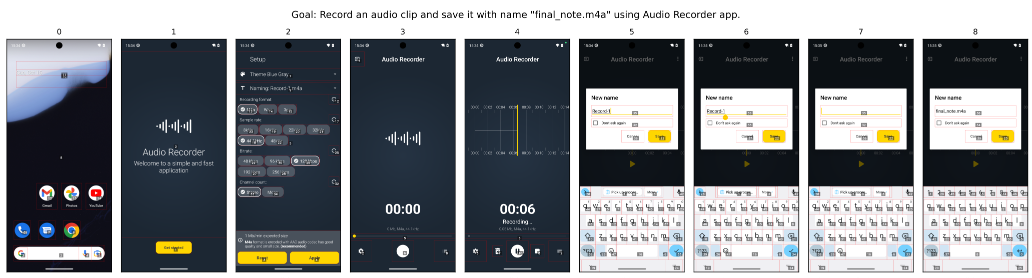

Figure 4: Example trajectory in AndroidWorld, with the goal at the top and the taken actions below each timestep's screenshot. AppVLM successfully creates an audio recording and saves it with the appropriate filename. Step 6 is noteworthy, with the agent opting for a `long-press` action, which is very rare in the initial AndroidControl dataset. This figure is in direct juxtaposition with Figure 2.

Figures 5-7 contain further example trajectories our agent solves successfully in an online fashion. These range from creating a new contact (Figure 5), to sending an sms (Figure 6) or deleting specific recipes from a dedicated app (Figure 7).

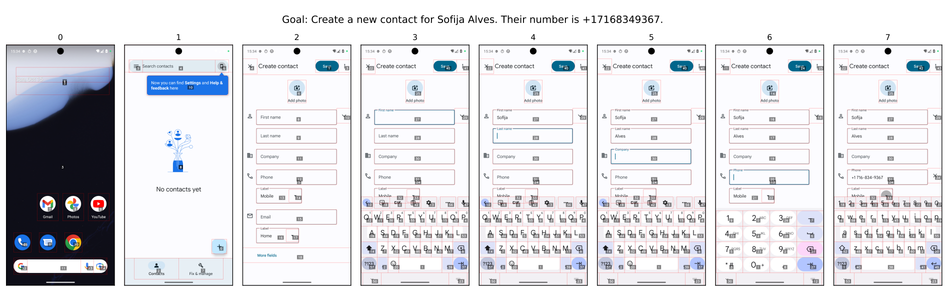

Figure 5: Example trajectory in AndroidWorld, with the goal at the top and the taken actions below each timestep's screenshot. AppVLM successfully creates a new contact, filling out several form fields to do so.

https://huggingface.co/google/paligemma-3b-pt-896

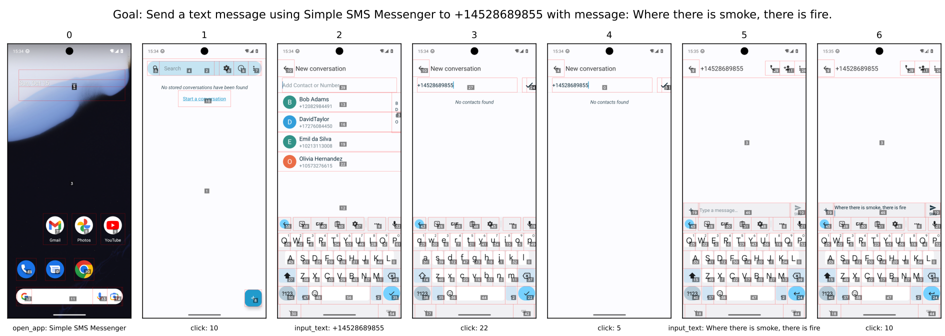

Figure 6: Example trajectory in AndroidWorld, with the goal at the top and the taken actions below each timestep's screenshot. AppVLM successfully sends a message to a specified phone number.

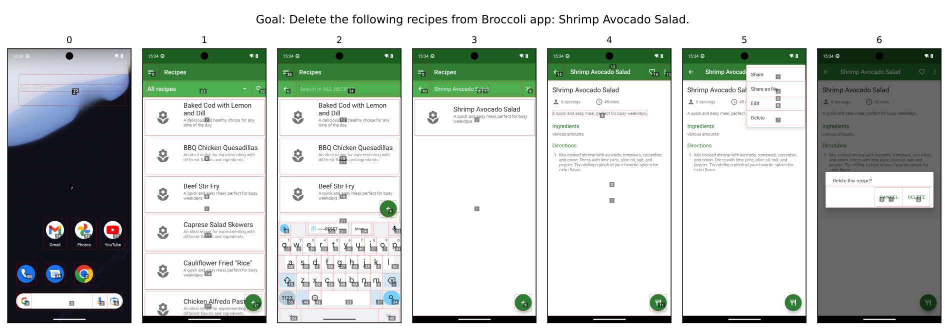

Figure 7: Example trajectory in AndroidWorld, with the goal at the top and the taken actions below each timestep's screenshot. AppVLM successfully deletes a specific recipe, even when this recipe is not immediately visible in the list