DigiRL: Training In-The-Wild Device-Control Agents with Autonomous Reinforcement Learning

Abstract

Pre-trained vision language models (VLMs), though powerful, typically lack training on decision-centric data, rendering them sub-optimal for decision-making tasks such as in-the-wild device control through Graphical User Interfaces (GUIs) when used off-the-shelf. While training with static demonstrations has shown some promise, we show that such methods fall short when controlling real GUIs due to their failure to deal with real world stochasticity and dynamism not captured in static observational data. This paper introduces a novel autonomous RL approach, called DigiRL, for training in-the-wild device control agents through fine-tuning a pre-trained VLM in two stages: offline and offline-to-online RL. We first build a scalable and parallelizable Android learning environment equipped with a VLM-based general-purpose evaluator and then identify the key design choices for simple and effective RL in this domain. We demonstrate the effectiveness of DigiRL using the Android-in-the-Wild (AitW) dataset, where our 1.5B VLM trained with RL achieves a 49.5% absolute improvement – from 17.7 to 67.2% success rate – over supervised fine-tuning with static human demonstration data. It is worth noting that such improvement is achieved without any additional supervision or demonstration data. These results significantly surpass not only the prior best agents, including AppAgent with GPT-4V (8.3% success rate) and the 17B CogAgent trained with AitW data (14.4%), but also our implementation of prior best autonomous RL approach based on filtered behavior cloning (57.8%), thereby establishing a new state-of-the-art for digital agents for in-the-wild device control.

1. Introduction

Advances in vision-language models (VLMs), especially in regards to their remarkable common-sense, reasoning, and generalization abilities imply that realizing a fully autonomous digital AI assistant, that can simplify human life by automating day-to-day activities on computer devices via natural language interfaces, is no longer a distant aspiration (Koh et al., 2024; Yan et al., 2023; Zhou et al., 2023). An effective device control AI assistant should be able to complete tasks in-the-wild through Graphical User Interfaces (GUIs) on digital devices: make travel plans; experiment with presentation designs; and operate a mobile device autonomously, all while running amidst stochasticity and distractors on the device, the Internet, and the tools it interacts with. However, enhanced reasoning or common-sense abilities do not directly transfer to intelligent assistant behavior: ultimately we want AI assistants to accomplish tasks, exhibit rational behavior, and recover from their mistakes as opposed to simply producing a plausible completion to a given observation based on the data seen during pre-training. This implies that a mechanism to channel abilities from pre-training into a deployable AI "agent" is lacking.

Even the strongest proprietary VLMs, such as GPT-4V (OpenAI Team, 2023) and Gemini 1.5 Pro (Gemini Team, 2024b), still struggle to produce the right actions when completing tasks on devices. While general-purpose vision-language abilities help these models still make meaningful abstract deductions about novel scenes when deployed, these deductions do not transfer to accurate reasoning for control (Yang et al., 2023; Yan et al., 2023; Zheng et al., 2024; Xie et al., 2024). As a result, most prior work for building device agents construct complex wrappers around proprietary VLMs, combining them with prompting, search, or tool use (Yang et al., 2023; Xie et al., 2024; Zhang et al., 2024b;a; Yan et al., 2023). While building prompting or retrieval wrappers to improve decision-making performance of existing VLMs provides a "stop-gap" solution in the short run, without updating the weights, the effectiveness of resulting agents is inherently limited by the capabilities of the base model (Zeng et al., 2023; Chen et al., 2023). For example, we found that off-the-shelf VLMs make reasoning failures that derail the agent (e.g., Figure 1 and Figure 11), and these are a direct consequence of the base model. A different solution is to fine-tune the model on demonstrations via imitation learning. However, the dynamic nature of the web and device means that models trained to mimic actions in stale data can result in sub-optimalilty as the eco-system changes (Pan et al., 2024). Additionally, agents trained in this way struggle to recover from out-of-distribution states resulting from the agents' own mistakes (Ghosh et al., 2021; Jiang et al., 2024).

If we can instead build an interactive approach to *train* a VLM to directly adapt and learn *from its own experience* on the device and the Internet, that can be used to build a robust and reliable device-control agent, without needing wrappers on top of proprietary models. However, this learning-based approach must satisfy some desiderata. First, it must use online interaction data since static demonstration data would not be representative of the task when the model is deployed: for instance, even in the setting of web navigation alone, dynamic nature of in-the-wild websites means that the agent will frequently encounter website versions that differ significantly from the scenarios seen during training and will need to behave reliably despite changes in visual appearance and distractions. Second, learning on-the-fly means the approach must learn from multi-turn interaction data from the model itself, a large of chunk of which would consist of failures. Proper mechanisms must be designed to automatically pick out the correct actions while filtering the wrong ones.

We evaluate our agent trained with DigiRL in carrying out diverse instructions from **Android in the Wild dataset** (Rawles et al., 2023) on real Android device emulators and find that our agent can achieve a **49.5% improvement** over the existing state-of-the-art agents (from 17.7% to 67.2% success rate) AutoUI (Zhang and Zhang, 2023) and CogAgent (Hong et al., 2023), and over 9% improvement over our implementation of the prior best autonomous learning approach based on Filtered Behavior Cloning. The performance of our agent also significantly surpasses wrappers on top of state-of-the-art proprietary VLMs such as GPT-4V (OpenAI Team, 2023) and Gemini 1.5 Pro (Gemini Team, 2024b) (17.7% success rate), despite using a significantly smaller model (with 1.5B parameters). To our knowledge, *this is the **first** work to successfully build an autonomous offline-to-online RL approach to enable state-of-the-art performance on device-control problems.*

2. DigiRL: autonomous RL for building a strong device control agent

We now present our autonomous RL framework for training device agents. We pose the device control problem as a partially-observed Markov decision process (POMDP) and develop RL methods for this POMDP. The core of our approach is based on a simple and scalable off-policy RL method, advantage-weighted regression (AWR) (Peng et al., 2019), but we make crucial modifications to handle stochasticity and highly-variable task difficulty, through the use of value functions trained with appropriate losses, and an automatic curriculum, induced by an instruction-level value function to maximize learning.

Definitions & notation. To explain our approach in detail, we include several standard definitions used in reinforcement learning (RL). The Q function for a policy π represents the expected long-term return from taking a specific action at the current step and then following policy π thereafter: $Q^\pi(s_h, a_h, c) = \mathbb{E}_\pi\left[\sum_{t=h}^{H} r(s_t, a_t, c)\right]$. The value function $V^\pi(s_h, c)$ is calculated by averaging the Q-value, $Q^\pi(s_h, a_h, c)$, over actions a_h drawn from the policy π. The advantage $A^\pi(s_h, a_h, c)$ for a state-action pair is computed by subtracting the state's value under the policy from its Q-value: $A^\pi(s_h, a_h, c) = Q^\pi(s_h, a_h, c) - V^\pi(s_h, c)$.

2.1. Backbone of our approach: off-policy RL via advantage-weighted regression

A starting point for our approach is the advantage-weighted regression (AWR) algorithm (Peng et al., 2019), which says that we can improve the policy reliably by regressing the policy towards exponentiated advantages induced by the reward function, as a proxy for optimizing the policy gradient while staying close to the previous policy (Kakade and Langford, 2002; Schulman et al., 2017; 2015):

$$\arg\max_\pi \mathbb{E}_\nu \left[\log \pi(a|s, c) \cdot \exp\left(A(s, a, c)/\beta\right)\right], \quad (2.1)$$

for some positive parameter β and the distribution of past experience ν, and $A(s, a, c)$ denotes the advantage of a state-action pair (s, a) given a context c. To avoid tuning the hyperparameter β, we consider an alternative that does "hard filtering" on the advantages instead of computing $\exp(A)$, similar to prior works (Nair et al., 2020; Wang et al., 2021). This leads to the following loss function for fine-tuning the model:

$$\mathcal{L}(\pi) = -\mathbb{E}_{\text{filter}(\nu)}[\log \pi(a|s, c)]. \quad (2.2)$$

Typically, these advantages are computed by running Monte-Carlo (MC) rollouts in the environment to estimate the value of a given state-action pair, and subtracting from it an estimate of the value of the state alone given by a learned value estimator. However, this approach is likely to produce high-variance advantages given the stochasticity of the device eco-system that affects MC rollouts.

2.2. Obtaining reliable advantage estimates from doubly-robust estimators

To reliably identify *advantageous* actions given significant environment stochasticity, we construct a per-step advantage estimator, inspired by doubly-robust estimators (van Hasselt et al., 2015; Schulman et al., 2018):

$$\begin{aligned} A^{\text{step}}(s_h, a_h, c) :=& \lambda^{H-h} r(s_H, a_H, c) + V^{\text{step}}(s_{h+1}, c) \\ &+ r(s_h, a_h, c) - V^{\text{step}}(s_h, c), \end{aligned} \quad (2.3)$$

where λ is a weighting hyper-parameter. This construction of the advantage estimator is a simplified version of Generalized Advantage Estimation (GAE) (Schulman et al., 2018), and balances an advantage estimator with higher variance Monte-Carlo estimates $\lambda^{H-h} r(s_H, a_H, c)$ (due to stochasticity) and an estimator with higher bias $V^{\text{step}}(s_{h+1}, c) + r(s_h, a_h, c) - V^{\text{step}}(s_h, c)$ (due to imperfect fitting of the value function). We observed that com-

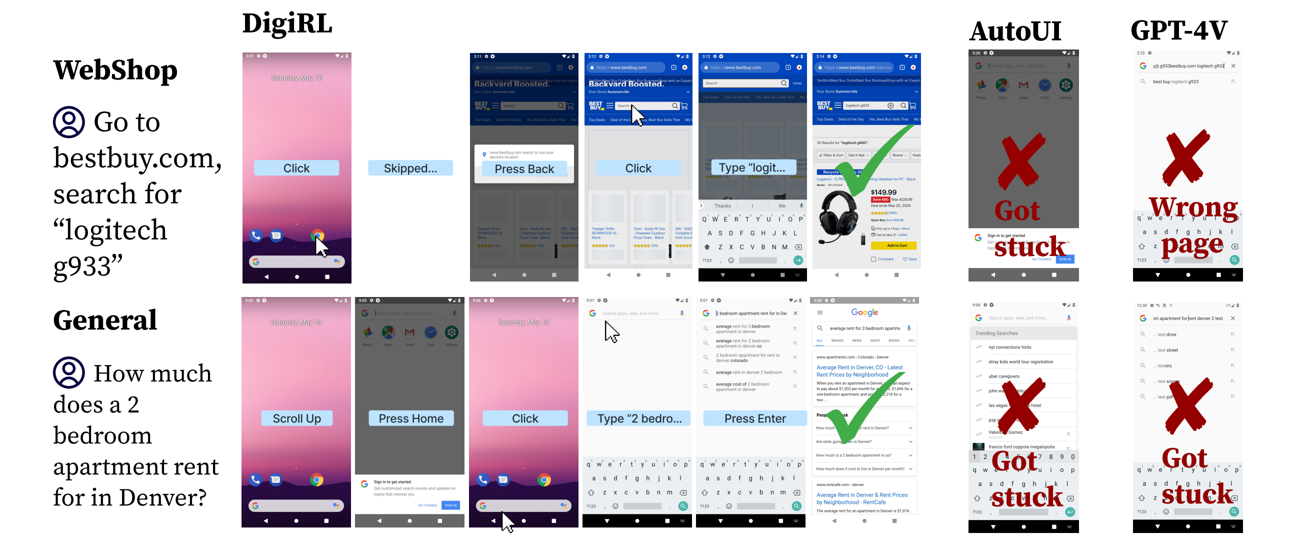


Figure 1: **Qualitative comparison between DigiRL and other approaches.** AutoUI trained from static human demonstrations can easily get stuck in out-of-distribution states while GPT-4V often get on a wrong goal (searched “logitech g933bestbuy.com logitech g933” in Google instead of bestbuy.com). In contrast, DigiRL can recover from such states and complete complex instruction as requested.

bining both high-variance and high-bias estimators gave us a sweet-spot in terms of performance. To implement the step-level hard filtering, we simply threshold this doubly robust estimator as $A^{\text{step}}(s_h, a_h, c) > 1/H$ to decide which actions progress towards the goal.

2.3. Automatic curriculum using an instruction-level value function

While the AWR update (Equation 2.1) coupled with a robust advantage estimator (Equation 2.3) is likely sufficient on standard RL tasks, we did not find it to be effective enough for device control in preliminary experiments. Often this was the case because the task set presents tasks with highly-variable difficulties that collecting more data on tasks that the agent was already proficient at affected sample efficieny negatively. In contrast, maximal learning signal can be derived by experiencing the most informative tasks for the agent during training. To this end, we design an instruction-level value function $V^{\text{instruct}}(c)$ to evaluate if a given rollout can provide an effective learning signal:

$$\begin{aligned} A^{\text{instruct}}(s_h, a_h, c) &:= \textstyle\sum_{t=h}^{H} r(s_t, a_t, c) - V^{\text{instruct}}(c) \\ &= r(s_H, a_H, c) - V^{\text{instruct}}(c), \end{aligned} \tag{2.4}$$

where $\sum_{t=h}^{H} r(s_t, a_t, c)$ is a Monte-Carlo estimator of $Q(s_h, a_h, c)$. The equality holds because the POMDP formulation only provides rewards at the end of a rollout. Intuitively, if a rollout attains a high value of $A^{\text{instruct}}(s_h, a_h, c)$, it means the value function V^{instruct} is small. Therefore, this rollout represents a valuable experience of the agent accomplishing a difficult task, and thus should be prioritized, akin to ideas pertaining to prioritized experience (Schaul et al., 2016) or level replay (Jiang et al., 2020). When training the actor with a buffer of historical off-policy data, we first perform a filtering step to identify the top-p datapoints with highest $A^{\text{instruct}}(s_h, a_h, c)$. Then, we use it for AWR (Equation 2.1) with the doubly-robust advantage estimator (Equation 2.3).

Implementation details. Inspired by the findings in some recent works (Farebrother et al., 2024; Kumar et al., 2023) that modern deep learning architectures like transformers (Vaswani et al., 2023) are better trained with cross-entropy losses instead of mean-squared losses, we utilize a cross-entropy objective based on the Monte-Carlo estimate of the trajectory reward for training both of our value functions:

$$\begin{aligned} \mathcal{L}(V^{\text{traj}}) = &- \mathbb{E}_{\nu}[r(s_H, a_H, c) \log V^{\text{traj}}(c) \\ &+ (1 - r(s_H, a_H, c)) \log(1 - V^{\text{traj}}(c))] \end{aligned} \tag{2.5}$$

$$\begin{aligned} \mathcal{L}(V^{\text{step}}) = &- \mathbb{E}_{\nu}[r(s_H, a_H, c) \log V^{\text{step}}(s_h, a_h, c) \\ &+ (1 - r(s_H, a_H, c)) \log(1 - V^{\text{step}}(s_h, a_h, c))] \end{aligned} \tag{2.6}$$

3. Experimental evaluation

The goal of our experiments is to evaluate the performance of DigiRL on challenging Android device control problems. Specifically, we are interested in understanding if DigiRL can produce agents that can effectively learn from autonomous interaction, while still being able to utilize offline data for learning. To this end, we perform a comparative analysis of DigiRL against several prior approaches, including state-of-the-art agents in Section 3.1. We also perform several ablation experiments to understand the necessity and sufficiency of various components of our approach in Section B.

3.1. Main results

Our main results are summarized in Table 1 and Figure 3. we find that in both AitW General and AitW Web Shop-

			AitW General		AitW Web Shopping	
			Train	Test	Train	Test
Prompting	Set-Of-Marks	GPT-4V	5.2	13.5	3.1	8.3
		Gemini 1.5 Pro	32.3	16.7	6.3	11.5
	AppAgent	GPT-4V	13.5	17.7	12.5	8.3
		Gemini 1.5 Pro	14.6	16.7	5.2	8.3
Learning	Supervised Training	CogAgent	7.8	7.8	8.6	14.4
		AutoUI	12.5	14.6	14.6	17.7
	Offline	Filtered BC	51.7 ± 5.4	50.7 ± 1.8	44.7 ± 1.6	45.8 ± 0.9
		Ours	46.9 ± 5.6	62.8 ± 1.0	39.3 ± 6.0	45.8 ± 6.6
	Off-to-On	Filtered BC	53.5 ± 0.8	61.5 ± 1.1	53.6 ± 4.7	57.8 ± 2.6
		Ours	**63.5 ± 0.0**	**71.9 ± 1.1**	**68.2 ± 6.8**	**67.2 ± 1.5**

Table 1: **Main comparisons of different agents across various settings.** Each offline experiment is repeated three times and the mean and standard deviation are reported. Each online experiment is repeated two times. Results are evaluated with our autonomous evaluator with the first 96 instructions in the train and test set.

ping subsets, our agent trained via DigiRL significantly outperforms prior state-of-the-art methods based on prompting and retrieval (AppAgent + GPT-4V/Gemini 1.5 Pro) or training on static demonstrations (CogAgent and AutoUI), by a large margin with more than **49.5% absolute improvement** (from 17.7% to 71.9% on the General subset and from 17.7% to 67.2% on the Web Shopping subset). Notably, this improvement from DigiRL is realized *fully autonomously without making use of human supervision* (e.g. manually labeled demonstrations or hand-written verifiers).

Are inference-time prompting and retrieval techniques or supervised training enough for device control? Delving into Table 1, we observe that off-the-shelf proprietary VLMs, even when supplemented with the set-of-marks mechanism, do not attain satisfactory performance: both GPT-4V and Gemini 1.5 Pro achieve success rates under 20%. One possible cause could be the under-representation of Android device data in the pre-training data. Moreover, inference-time adaptation strategies such as AppAgent (Yang et al., 2023) show minimal improvement, with gains not exceeding 5% for either model, suggesting a limited scope for improvement without fine-tuning of some sort. As illustrated in Figure 4, the primary failures of these VLMs stem from hallucinatory reasoning that lead the VLMs to land on a relevant but wrong page. This suggests that while state-of-the-art VLMs excel at high-level reasoning in code or math problems, their reliability of reasoning in less familiar domains, such as device control, remains inadequate. For example, for the instruction "Go to newegg.com, search for 'alienware area 51', and select the first entry", a GPT-4V based agent erroneously searched "alien area 51 ebay" in Google.com and decided that it had made progress towards the task (Figure 11).

Training on domain-specific human demonstrations, however, does boost performance, allowing the smaller, specialized VLM, AutoUI, to match or surpass the larger, generalist VLMs like GPT-4V and Gemini 1.5 Pro. Nonetheless, this supervised imitation learning approach still fall short, with success rates on both subsets remaining below 20%. This shortcoming is not addressed via enhancements in model scale or architecture, as evidenced by CogAgent (Hong et al., 2023), with 17 billion parameters still achieving similar performance to AutoUI (Zhang and Zhang, 2023), which has only 1.5 billion parameters. As depicted in Figure 4, a predominant failure mode for these agents is an inability to rectify their own errors. An example trajectory that we observed is that for the instruction "what's on the menu of In-n-Out", the agent accidentally activated the voice input button, and failed to quit that page until the step limit. In contrast, DigiRL is able to recover from the errors more efficiently(Appendix D.2).

Comparison of different RL approaches. In Table 1 and Figure 3, we present a comparative analysis of various RL approaches. Notably, both offline and offline-to-online configurations demonstrate that our RL approach, when augmented with a continuous stream of autonomous interaction data and reward feedback, substantially improves performance. This improvement is evident from an increase in the success rate from under 20% to over 40%, as the agent learns to adapt to stochastic and non-stationary device interfaces. Moreover, although the total sample sizes for offline and offline-to-online settings are equivalent, the top-performing offline-to-online algorithm markedly surpasses its offline counterpart (75% versus 62.8% on the General subset). This highlights the critical role and efficacy of autonomous environment interaction, and establishes the efficacy of DigiRL in learning from such uncurated, sub-optimal data. Lastly, DigiRL consistently outperforms the state-of-the-art alternative, Filtered BC, across both the General and Web Shopping subsets, improving from 61.5% to 71.9% and 57.8% to 61.4%, respectively, highlighting DigiRL's performance and efficiency.

References

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *ArXiv*, abs/2310.05915, 2023. URL https://api.semanticscholar.org/CorpusID:263829338.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop regressing: Training value functions via classification for scalable deep rl, 2024.

2023 Gemini Team. Gemini: A family of highly capable multimodal models, 2024a.

2024 Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024b.

Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why Generalization in RL is Difficult: Epistemic POMDPs and Implicit Partial Observability. *NeurIPS*, 2021.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2023.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. *CoRR*, abs/2010.03934, 2020. URL https://arxiv.org/abs/2010.03934.

Yiding Jiang, J Zico Kolter, and Roberta Raileanu. On the importance of exploration for generalization in reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002. URL https://api.semanticscholar.org/CorpusID:31442909.

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.

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes, 2023.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *CoRR*, abs/2006.09359, 2020. URL https://arxiv.org/abs/2006.09359.

2023 OpenAI Team. Gpt-4 technical report, 2023.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*, 2024.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019.

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.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/shi17a.html.

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.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression, 2021.

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.

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, 2023.

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.

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

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms, 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.

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, 2024b.

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

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

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *ArXiv*, abs/2307.13854, 2023. URL https://api.semanticscholar.org/CorpusID:260164780.

Appendices

A. Problem setup and preliminaries

Problem formulation. We are interested in pixel-based interaction with virtual devices. We scope our study in the control of Android devices: this is already significantly more challenging and more general than previous learning-based environments that focus solely on web navigation (Koh et al., 2024; Zhou et al., 2023; Drouin et al., 2024), where the web browser itself is merely one application within our broader environment, and link-based device controls (Yang et al., 2023; Zhang et al., 2024a) are inadequate for tasks like games that do not support link inputs.

Each episode begins with the emulator initialized to the home screen. Subsequently, a task is selected from a predefined set of language instructions, some examples of which are shown in Appendix C.1. An agent is then tasked with manipulating the emulator to fulfill this instruction. At each time step, the agent receives a screenshot of the current screen as the observation. Following the action space in prior literature (Rawles et al., 2023), the available actions include tapping and sliding based on normalized (x, y) coordinates (ranging from 0 to 1 relative to the screen dimensions), typing text strings of variable length, and pressing special buttons such as HOME, BACK, and ENTER, as illustrated in Figure 2. Our train and test instructions comes from General and Web Shopping subsets in AitW (Rawles et al., 2023). These tasks consist of information-gathering tasks like "What's on the menu of In-n-Out?", and shopping tasks on the web like "Go to newegg.com, search for razer kraken, and select the first entry".

Challenges of stochasticity. Real-world device contrl presents unique challenges of stochasticity absent in simulated environments (Zhou et al., 2023; Shi et al., 2017) such as: **(1)** the dynamic nature of websites and applications, which undergo frequent updates, causing the online observations to be different from stale offline data, **(2)** various unpredictable distractors such as pop-up advertisements, login requests, and the stochastic order of search results. **(3)** technical challenges and glitches such as incomplete webpage loading or temporary access restrictions to certain sites. Examples of scenarios with such stochasticity from our experiments are shown in Figure 2. We observe that these stochastic elements pose significant challenges for pre-trained VLMs, including even those fine-tuned on device control data.

Setup for reliable and scalable online RL. As autonomous RL interleaves data collection and training, to maximize learning amidst stochasticity, it is crucial to have a real-time data collection pipeline to collect enough experience for gradient updates. While this is not possible in single-thread Android emulator environments (Pan et al., 2024; Toyama et al., 2021) due to latency, we parallelize our Android emulator using appropriate error handling as discussed in Appendix C.1. In addition, the environment must provide a reward signal by judging whether the current observation indicates the agent has successfully completed the task. To generalize our *evaluator* to support a wide range of tasks, we extend Pan et al. (2024)'s end-to-end autonomous evaluator that does not require accessing the internal states of the emulator or human-written rules for each task. This contrasts previous works that manually write execution functions to verify the functional completeness of each task (Koh et al., 2024; Yao et al., 2023; Shi et al., 2017; Xie et al., 2024). We adopt Gemini 1.5 Pro (Gemini Team, 2024a;b) as the backbone of the autonomous evaluator. We seed this model with few-shot rollouts and the associated human-labeled success indicators to guide evaluation of novel queries. This pipeline enables a single evaluator that can evaluate all AiTW tasks. The evaluator is highly aligned with human annotations (average error rate 2.8%), validated in Figure 5.

Baselines and comparisons. We compare DigiRL with: **(a)** state-of-the-art agents built around proprietary VLMs, with the use of several prompting and retrieval-style techniques; **(b)** running imitation learning on static human demonstrations with the same instruction distribution, and **(c)**a filtered BC approach (Pan et al., 2024). For proprietary VLMs, we evaluate **GPT-4V** (OpenAI Team, 2023) and **Gemini 1.5 Pro** (Gemini Team, 2024b) both zero-shot and when augmented with carefully-designed prompts. For the zero-shot setting, we use the prompt from Yang et al. (2023) and augment the observation with Set-of-Marks (Zheng et al., 2024). Set-of-Marks overlays a number for each interactable element over the screenshot, so that a VLM can directly output the number of the element to interact with in plain text instead of attempting to calculate pixel coordinates, which is typically significantly harder. We also compare with AppAgent (Yang et al., 2023), which first prompts the VLM to explore the environment, and appends the experience collected to the test-time prompt. We also compare with two state-of-the-art fine-tuning methods for Android device control: **AutoUI** (specifically AutoUI-Base (Zhang and Zhang, 2023)) and **CogAgent** (Hong et al., 2023). AutoUI-Base uses an LM with 200M parameters, and a a vision encoder with 1.1B parameters. CogAgent has 11B parameters for its vision encoder and 7B for its LM. The supervised training corpus for both AutoUI-Base and CogAgent contains AitW, including the instruction set and the emulator configuration we use.

Base VLM and offline dataset. Both **Filtered BC** and **DigiRL** use trained AutoUI-Base checkpoints with the image encoder frozen. The instruction and step-level value functions for DigiRL employ this same frozen image encoder. The

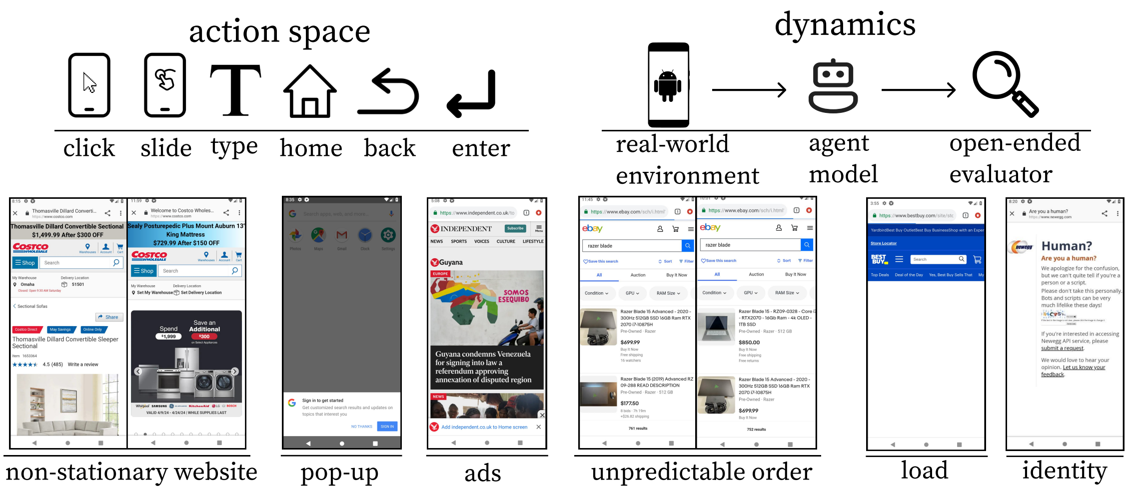


Figure 2: **Environment details.** *Top:* actions space and dynamics of the environment. *Bottom:* examples of the read-world non-stationarity and dynamism of the environment.

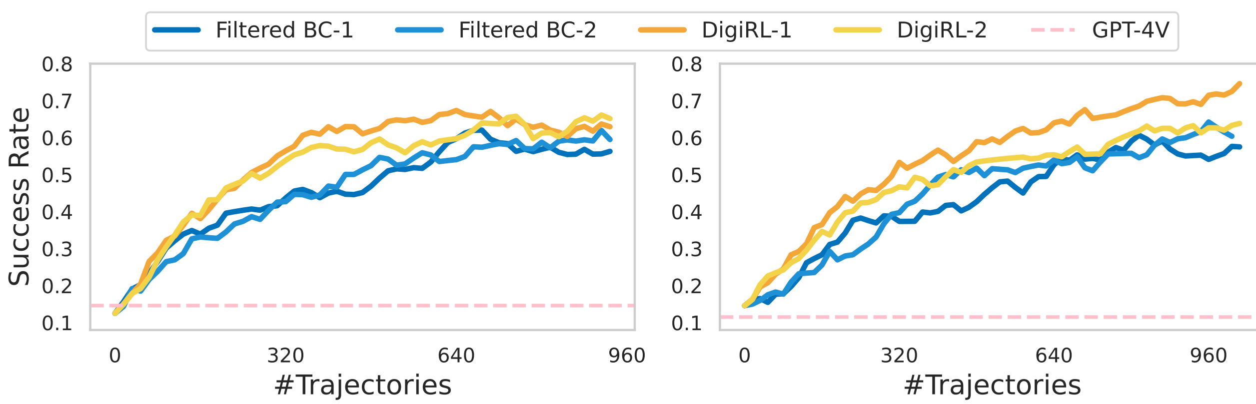


Figure 3: **Offline-to-online training curves for Filtered BC and DigiRL.** Curves are smoothed with exponential weighted averaging to start from the performance of supervised trained policy. Two runs for each model are started on two different dates with at least two days apart. Observe that DigiRL is able to improve faster with a fewer number of samples. Since the data collection frequency is the bottleneck, these performance trends directly reflect performance trends against wall-clock time as well.

visual features output from the encoder are concatenated with instruction features derived from RoBERTa (Liu et al., 2019). A two-layer MLP is then used to predict the value function. In the offline phase, the offline dataset is collected by rolling out the initial AutoUI-Base supervised trained checkpoint as policy. For fair comparisons, we keep the number of offline data collected in the pure offline training roughly the same as the total number of data collected in the offline-to-online training. Due to the dynamic nature of the Internet-device eco-system, our offline data was stale by the time we were able to run our offline-to-online experiments, and this presented additional challenge in offline-to-online learning. In both General and Web Shopping subsets, offline experiments make use of around 1500 trajectories while offline-to-online experiments start with around 500 offline trajectories and update with another 1000 online trajectories. In the offline phase, DigiRL skips instruction-level filtering and instead trains the actor with all successful trajectories to make full use of the offline data. See a detailed breakdown of our dataset in Appendix C.1.

B. Discussions

Failure mode analysis. While all the types of failure modes benefit from offline and offline-to-online RL training as shown in Figure 4, the most consistent and significant reduction is probably for the failure mode of failing to recover from mistakes. This is because while pre-trained models, generating plausible future tokens, can get distracted by the dynamic nature of

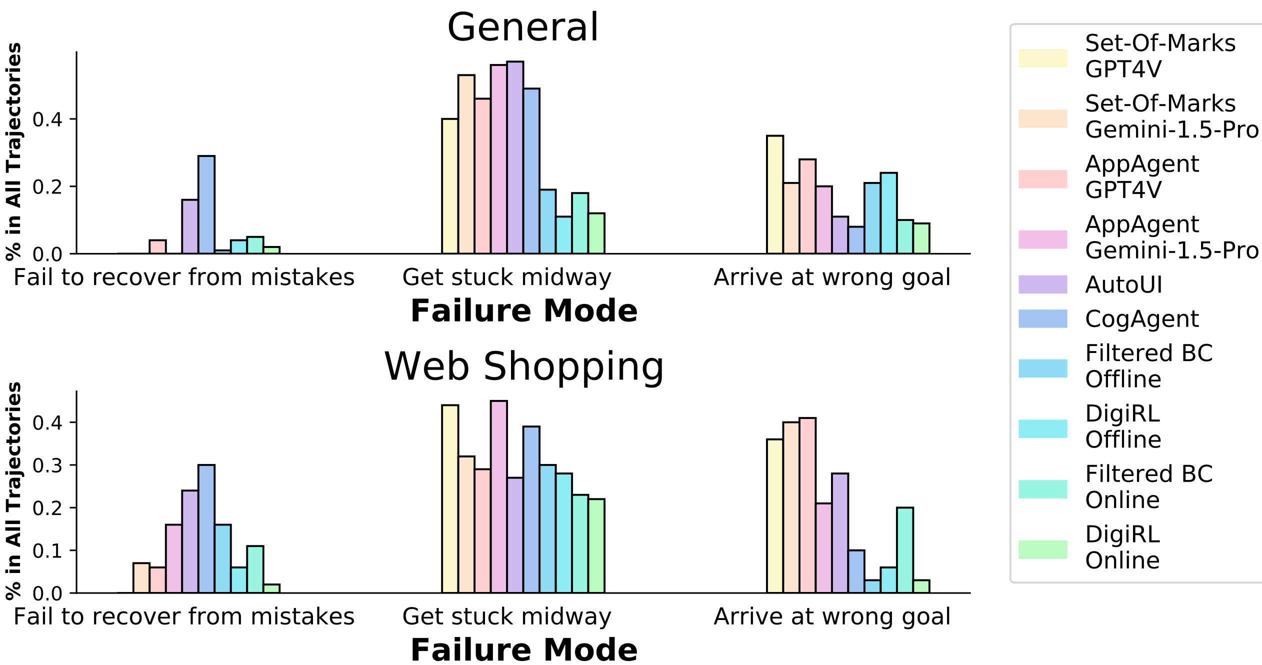


Figure 4: **Failure modes for each approach** on both the AiTW General and Web Shopping subsets. We found that the failure mode RL training is most effective at reducing compared to model supervised trained on human data is "Fail to recover from mistakes". A more fine-grained decomposition can be found in Appendix E.

the environment and, as a result, encounter at never-before-seen states. With no clue of how to escape such states, these methods are unable to recover and fail to solve the task. In contrast, by training on autonomously-collected rollouts, our agent DigiRL is able to learn from its own mistakes and reduces failures to recover over training.

Ablation study of each component in DigiRL. We conduct an ablation study on different components of DigiRL in Figure 6 (right). We find that all the components used by our approach are necessary: **(1)** using cross-entropy for training the value functions boosts performance by around 12% (compare Ours and Ours w/ Regression); **(2)** using step-level advantages improves efficiency by 12% (comparing Ours and Ours w/o step-level advantage); **(3)** the use of automatic curriculum improves the speed of learning by around 25% (comparing Ours w/o step-level advantage and Filtered BC); **(4)** Ours outperforms vanilla AWR that does not employ a doubly-robust advantage estimator or curriculum.

Additionally, we also observe no degradation in performance as a result of "hard-filtering", as show by nearly comparable performance of our approach and the best run of exponential filtering obtained via an extensive tuning of the temperature hyperparameter τ in naïve AWR (comparing Ours and Ours w/ vanilla AWR reweighting), despite simplicity of implementa-

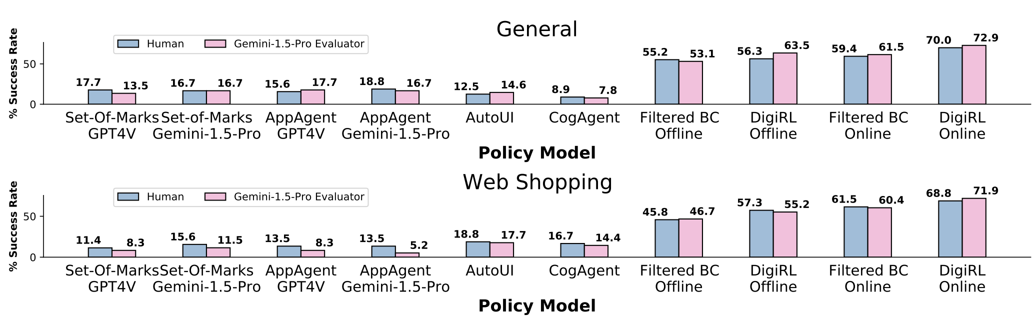


Figure 5: **Correlation between our autonomous evaluator and human judgements for all policy models** on General and Web Shopping subsets. For repeated offline and online runs, we report the correlation results for the run with the highest autonomous evaluation success rate.

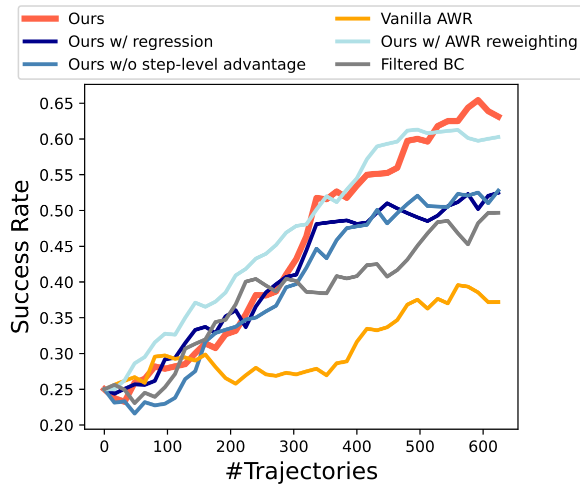


Figure 6: **Ablation study results on the AitW Web Shopping subset.**

tion in the hard filtering approach. Putting together, these choices result in a new state-of-the-art RL approach for device control.

Evaluation of our autonomous evaluator. In Figure 5, we present the findings from a user study aimed at assessing the accuracy of our autonomous evaluator. Our results indicate that the success rates reported by our automatic evaluator are remarkably consistent with those assessed by human evaluators across almost all models, with differences less than 3%. Furthermore, we observed that evaluations on the Web Shopping subset are more precise compared to those on the General subset. This increased accuracy likely stems from the fact that tasks in the General subset are formulated in free-form language, which can introduce ambiguity, whereas the Web Shopping subset features a narrower range of language expressions, reducing potential variability.

C. Environment details

C.1. Post-processing of AitW

The Android in the Wild (AiTW) task set is a large-scale dataset for android device control, containing five subsets: GoogleApps, Install, Web Shopping, General, and Single, where we select the General and Web Shopping subsets. Single subset is not considered here because all tasks in Single can be completed within one step and thus this subset fails to examine the multi-step challenges that we are interested in this paper. Install and GoogleApps are not considered due to security reasons as those tasks require an active Google account and parallel emulations can flag security concerns.

General. The General set focuses on searching for information and basic application usage. For example, it contains searching for latest news in Chile, search for flights from NYC to Sydney, opening Gmail, etc. We use all 545 tasks in the training set for training and the first 96 tasks in the test set for testing due to computational and budget constraints. The maximum allowed number of steps for this subset is 10. Offline data is collected by rolling our the initial AutoUI policy with tasks from the training set. The offline data used for the offline-to-online setting contains 608 trajectories while the offline data used for the offline setting contains 1552 trajectories. Some task examples are shown in Table 3.

Web Shopping. The Web Shopping subset comprises search instructions on various shopping websites, like searching for razer blader on ebay. As some websites (e.g. Amazon) and operations (e.g. adding items to cart) frequently require captcha verifications, we post-process the Web Shopping subset to exclude such operations and websites and also make the task easy to evaluate for our autonomous evaluator. The resulting task set involves navigating through five websites (costco.com, bestbuy.com, target.com, walmart.com, newegg.com) and three basic operations (go to website, search in the website, and select items from the searched results). Our post-processed training set contains 438 tasks and our testing set contains 96 tasks. Example tasks after post-processing can be found in Table 3. The maximum allowed number of steps for this subset is 20. Offline data is collected by rolling our the initial AutoUI policy with tasks from the training set. The offline data used for the offline-to-online setting contains 528 trajectories while the offline data used for the offline setting contains 1296

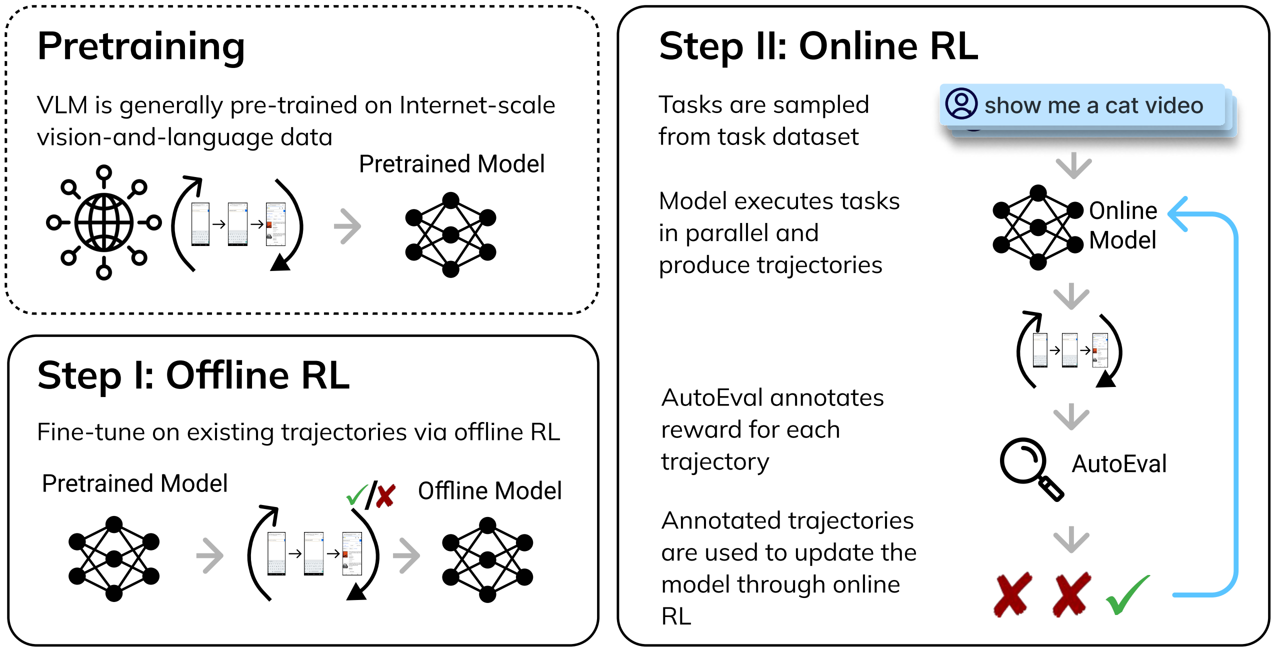


Figure 7: **DigiRL overview.** DigiRL is built upon a VLM that has been pre-trained on extensive web data to develop fundamental skills such as common knowledge, reasoning, and visual grounding. Initially, we employ offline RL to fine-tune the VLM using stale task-specific data, which helps in eliciting goal-oriented behaviors. Subsequently, our agent engages with real-world graphical user interfaces, continuously enhancing its performance through online RL and autonomous performance evaluations.

Task Example
How do I get to the nearest Verizon Store?
How much does a 2 bedroom apartment rent for in Denver?
Search for flights from Barcelona to Boston
What's a good restaurant in New York?
What's on the menu at Burger King?

Table 2: Examples of task descriptions in the AiTW General task set.

trajectories.

Difficulty	Task Example
1	Go to costco.com
	Go to walmart.com
2	Go to costco.com, search for "bose soundsport free"
	Go to walmart.com, search for "logitech g910"
3	Go to costco.com, search for "bose soundsport free" and select the first entry
	Go to walmart.com, search for "logitech g910" and select the first entry

Table 3: Examples of task descriptions in the AiTW Webshopping task set.

D. Qualitative examples

D.1. Random sample of trajectories for different agents

In Figures 8 and 9, we provide trajectories of DigiRL, AutoUI, and GPT-4V randomly sampled from our test set to offer a qualitative understanding of the agents' performance. As shown in these examples, DigiRLcan efficiently carry out in-the-wild device control tasks and less likely to get stuck or get to a wrong page compared to AutoUI and GPT-4V.

D.2. Error Recovery

We observe that DigiRL is able to recover from its own mistakes. As shown in Figure 10, we find that DigiRL explores ways to get back to the original screen in order to perform a search. As a comparison, AutoUI fails to reset to the original screen and gets stuck at the diverged screen. Under the hood, we find DigiRL trying to maximize the state value, which usually induces it to reset to the original screen (that has a large value to success).

D.3. Reasoning failure of GPT-4V

The performance of GPT-4V failed on AiTW tasks predominantly due to not being able to carry out control actions as it plans on a high level, and then not being able to recover from these mistakes. Moreover, one of the main reasons why it is not able to recover from a mistake is that it might hallucinate and make itself believe that it is a wrong app or website. Indeed, GPT-4V constructs a plan of further actions when provided a task from either Web Shopping or General dataset of AiTW. Then, when it makes a misclick and fails to successfully proceed in an intermediate step, it might think that it actually solved that intermediate step and is in the correct app or website to execute further actions, causing the overall trajectory to fail. An example of this is provided in Figure 11. Here, we ask the model to search for an item in a webshopping website, in particular in "newegg.com". However, the model fails to proceed to that website due to not being able to precisely locating the search button. Then, instead of trying to go to that website again, the model thinks it is already in that webshopping website, and mistakes the search bar of Google with the search bar of "newegg.com". Hence, the rest of the trajectory also fails. Another slightly different phenomenon is illustrated in Figure 12. Here, the model is able to proceed to the correct website and search for an item, but this time it fails to tap on the search button on the website and clicks to an advertisement instead. Consequently, the model fools itself to think it successfully searched the item, and scrolls the page hoping to find that item, but it cannot do so because in reality it views the results of the advertisement. The primary reason of these failures is the challenge of grounding the control actions in GUI interfaces to realize the intermediary goals laid out by GPT-4V model's thoughts. As an example, we provide an illustration of trying to set up an alarm task in Figure 13. Here, in the last frame, it fails to execute the precise movements in the necessary amount of rounds to correctly set up the alarm to the desired time, and in the last frame we see that the action taken does not align with the thought process of the model.

E. Fine-grained failure modes

In Figure 14, we present a more fine-grained breakdown for all six failure modes provided in the user study. Those failure modes include:

- *Failure to recover from mistakes* refers to the scenario where the agent made a mistake that led it to states from which it

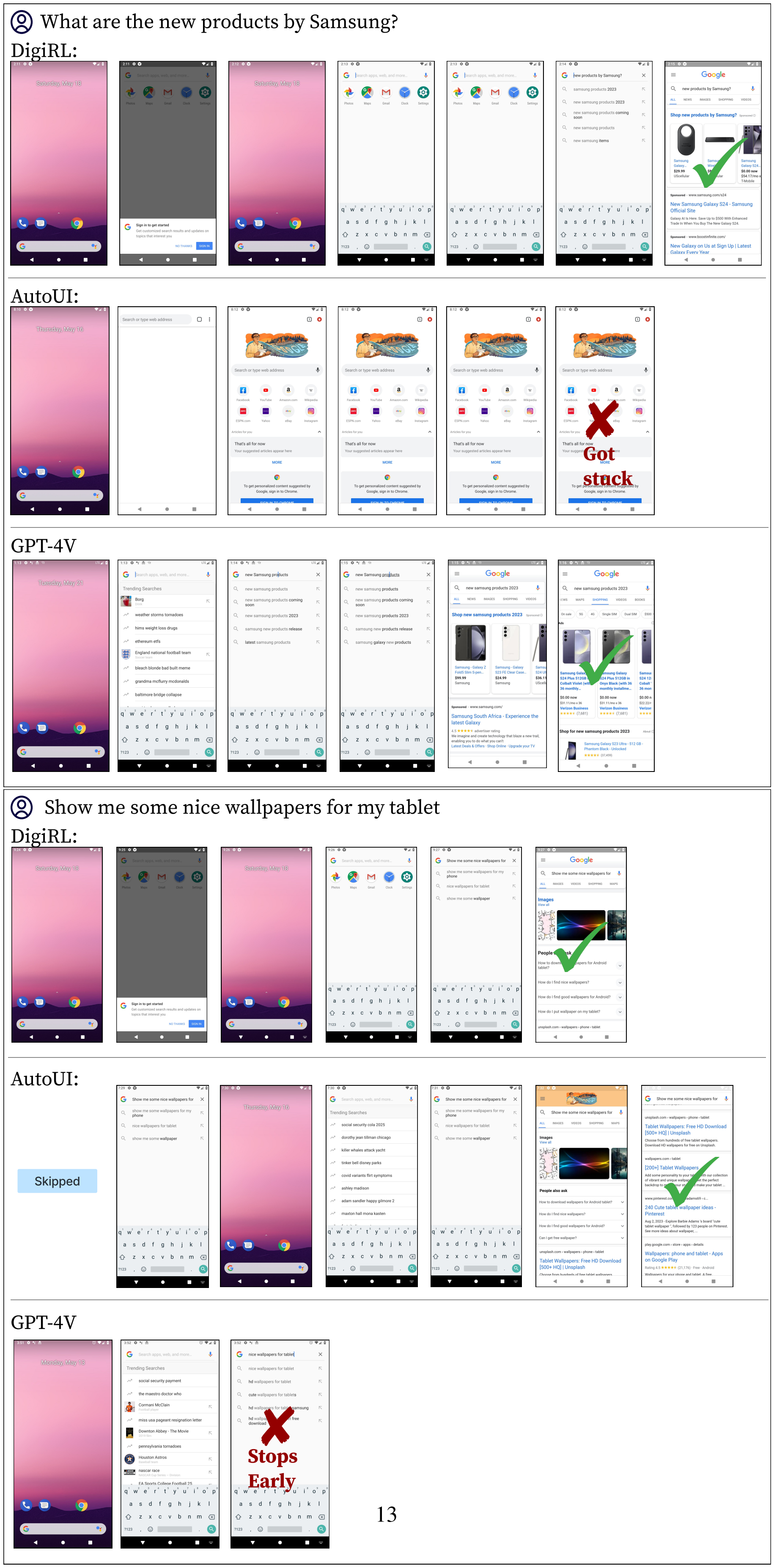


Figure 8: Agents' trajectory on two randomly sampled tasks on the General split of AitW.

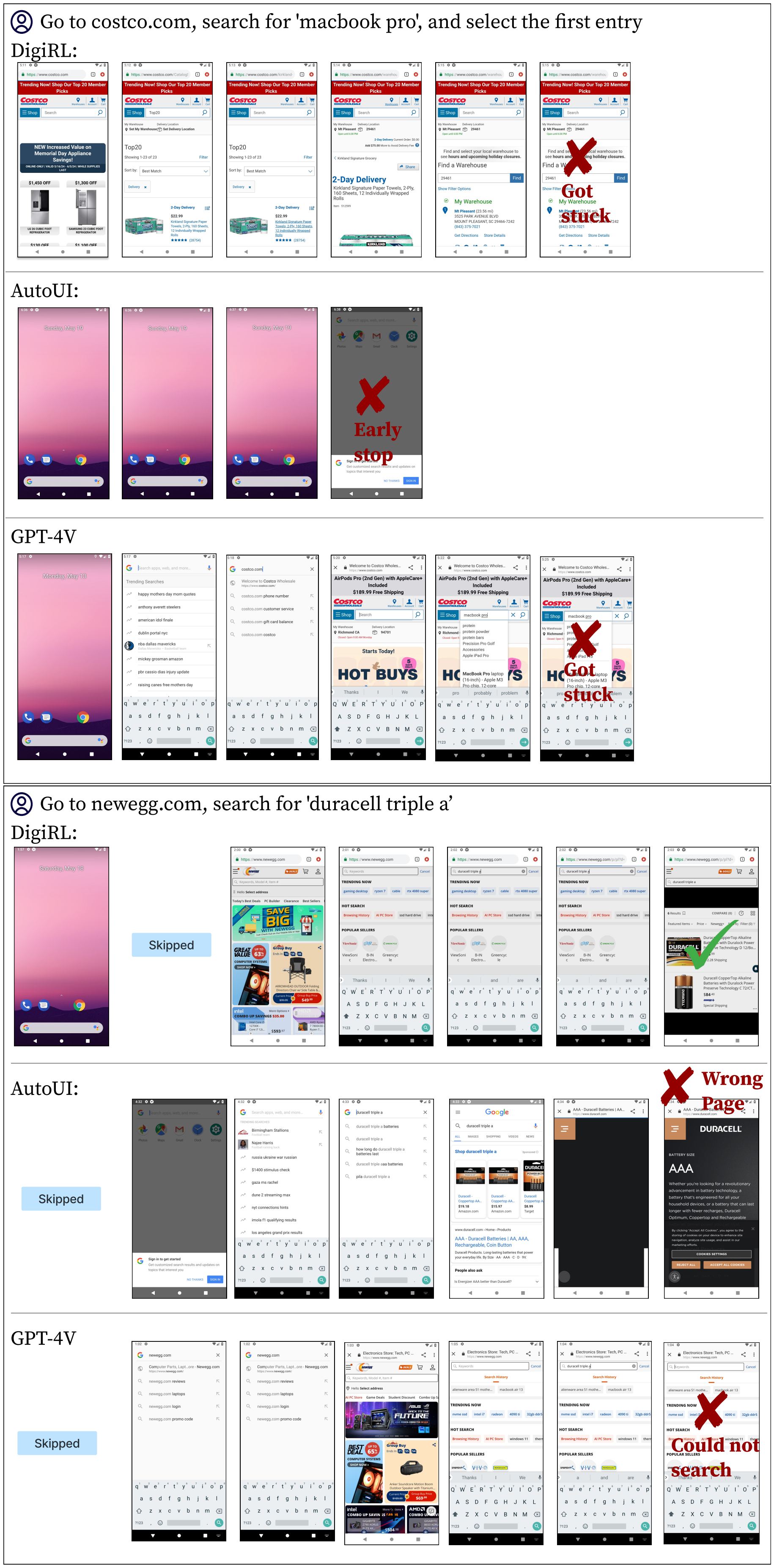


Figure 9: Agents' trajectory on two randomly sampled tasks on the WebShop split of AitW.

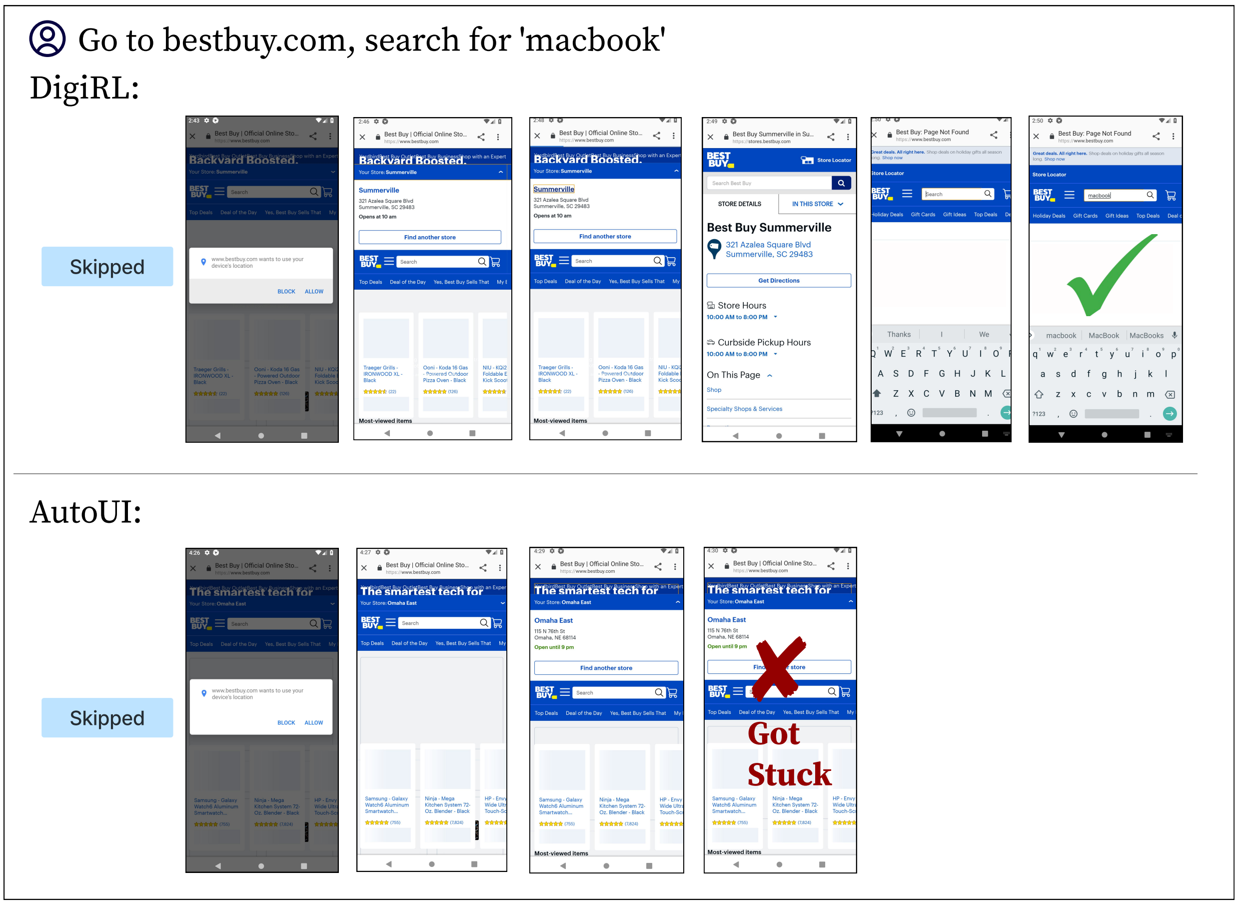


Figure 10: Error recovery cases. In `bestbuy.com`, we systematically find DigiRL able to recover from its own mistakes, while AutoUI fails to do so.

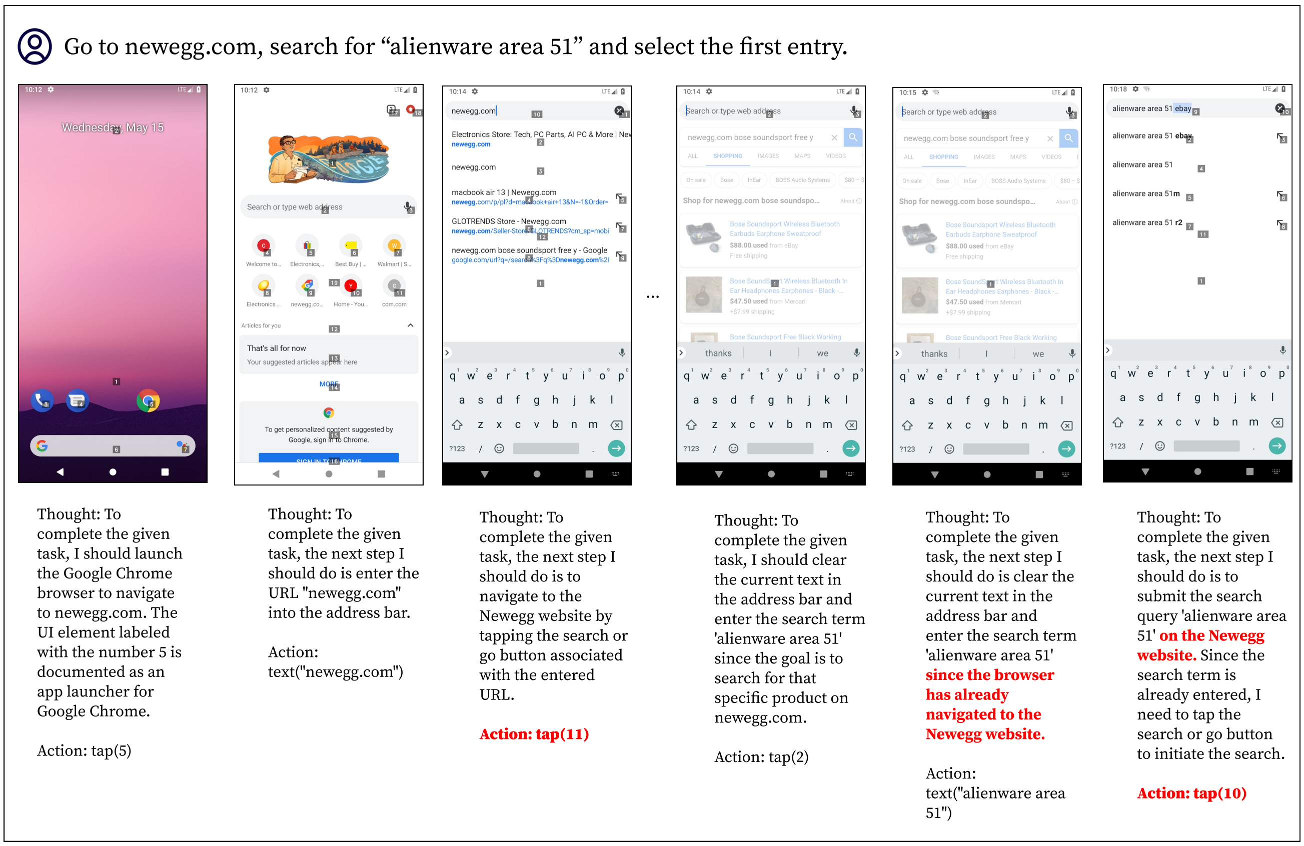
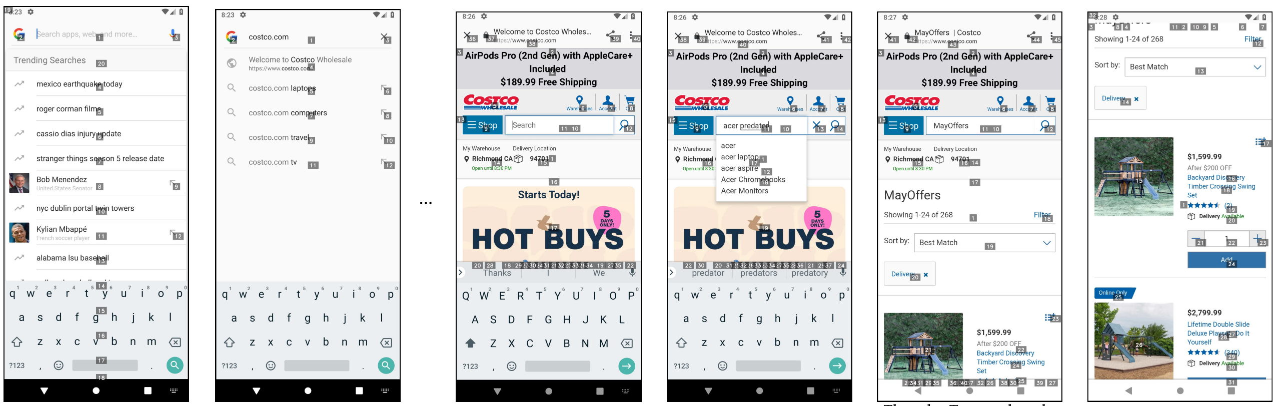


Figure 11: Failure of GPT-4V, with its thoughts and link-based actions given. A typical cause of failure is that it cannot tap on the correct "search" button after entering a query and mistakenly tapped onto the "x" symbol in the search bar as the "search" button. Here the goal is: Go to newegg.com, search for "alienware area 51" and select the first entry. As seen in red emboldened actions, it fails to press search button and deletes the query instead. Also, as seen in red highlighted parts in thoughts, it thinks it is in "newegg.com" website even though it is not.

Go to costco.com, search for “acer predator”, and select the first entry.



Thought: To complete the given task, the next step I should do is type "costco.com" into the search bar since I have already initiated the process to navigate to costco.com by tapping on the Google search bar.

Action: text("costco.com")

Thought: To complete the given task, the next step I should do is navigate to the Costco website. Since the address "costco.com" is already entered in the address bar, I need to initiate the web search or load the page by tapping the search button on the keyboard.

Action: tap(4)

...

Thought: To complete the given task, the next step I should do is enter the search term 'acer predator' into the search bar using the text input function.

Action: text("acer predator")

Thought: To complete the given task, the next step I should do is to initiate the search by tapping the search button on the keyboard.

Action: **tap(24)**

Thought: To complete the given task, I should scroll through the product listings to find the 'acer predator' item. Since the first entry is what we need to select, **I should swipe up on the element labeled '26' to view more products.**

Action: **swipe(26, "up", "medium")**

Thought: To complete the given task, **I should continue to swipe up on the product listings to find the 'acer predator' item.**

Action: **swipe(21, "up", "medium")**

Figure 12: Failure of GPT-4V, with its thoughts and link-based actions given. This time the reason for failure is misclick on the wrong button. The task is “Go to costco.com, search for “acer predator”, and select the first entry”. Notice that up until the fourth frame in this Figure, the trajectory goes correct. But then it clicks on the generic advertisements on the Costco.com website, and it cannot recover back. It continues to scroll the page and takes wrong actions thereafter.

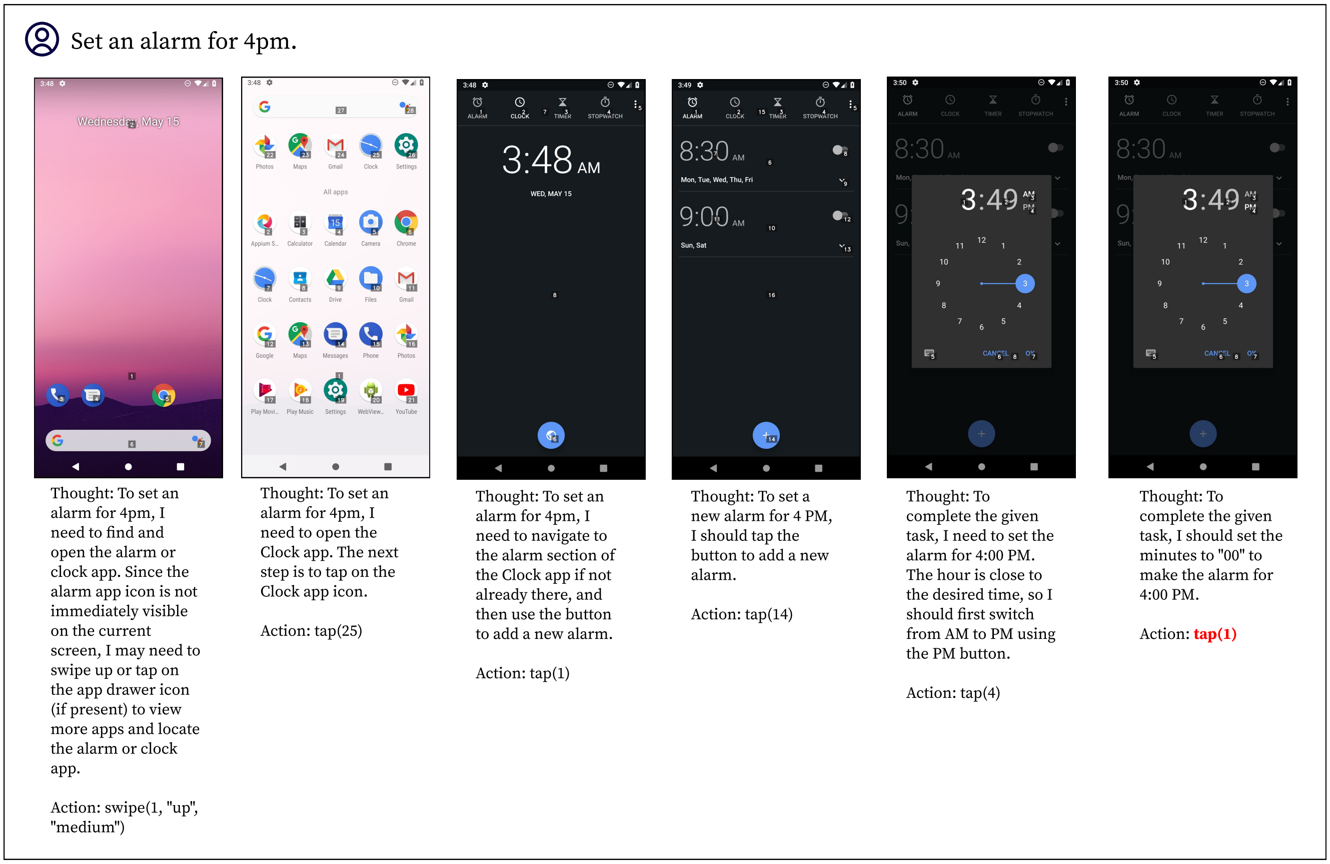


Figure 13: Failure of GPT-4V, with an example task on the AiTW general test set. The task is "Set an alarm for 4pm". Here, GPT-4V is able to successfully navigate to the clock app, and the alarm settings of that app. However, it cannot take the correct precise actions to set the alarm quickly enough, and it fails due to maximum rounds reached. In the last round, notice that the action of tap(1) contradict with its own thought process of setting minutes to "00".

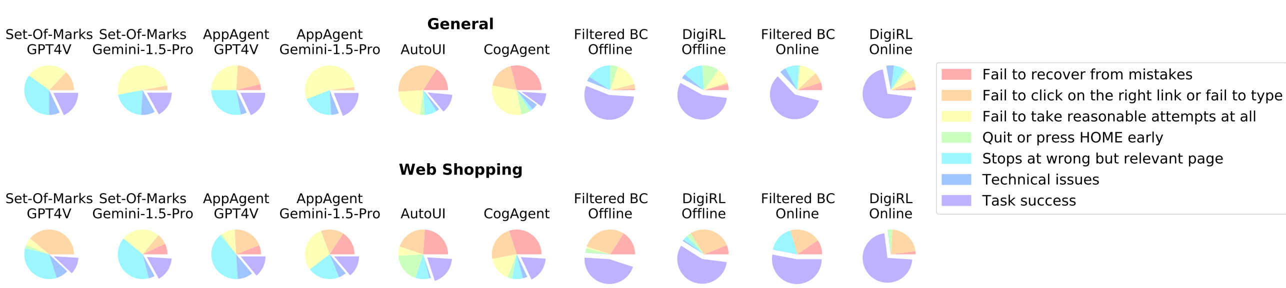


Figure 14: Failure modes decomposition for each policy model for both General and Web Shopping subsets.

failed to quickly recover and resume the task, such as a wrong google search page.

- *Failure to click on the right link or failure to click* refers to the failure mode where the agent either fails to locate the element that it tries to click on and keeps clicking on the nearby region, or fails to start typing in the string when it is supposed to do so.
- *Failure to take reasonable attempts at all* refers to the failure mode where there is no clear reason that the agent fails to complete the task and does not seem to be on the right track throughout the trajectory.
- *Quit or press HOME early* refers to the failure mode where the agent decided to finish the task or press HOME to start over before the task is actually finished.
- *Stops at wrong but relevant page* refers to the failure mode where the agent arrives at a wrong page and mistakenly thinks that it had completed the task. For example, the agent finds a macbook on costco.com while the instruction asked it to find a macbook on ebay.com.
- *Technical issues* refer to the failure mode that either the task is impossible (e.g. the tasks asks to open Amazon app but this app is not installed) or the agent is temporarily blocked from a certain website due to frequent visits.

The translation between fine-grained failure modes and coarse-grained failure modes is presented in Table 4.

Fine-Grained Failure	Coarse-Grained Failure
Fail to recover from mistakes	Fail to recover from mistakes
Fail to click on the right link or fail to type	Get stuck midway
Fail to take reasonable attempts at all	Get stuck midway
Quit or Press HOME early	Arrive at wrong goal
Stops at wrong but relevant page	Arrive at wrong goal
Technical Issues	None

Table 4: Examples of task descriptions in the AiTW Webshopping task set.

F. Experiment machines

Our main experiments are conducted on VM instances from Google Cloud Platform. Each VM instance comes with 1x Tesla T4 GPU and 16x Intel(R) Xeon(R) CPU.

G. Setup for parallel environment

Running multiple emulators in parallel can be challenging due to the inefficiency in thread synchronization and frequent fault propagation when one emulator runs into an unknown error. To address this challenge, we set up a server-client system where all emulator processes are running in independent server processes. Each emulator process communicates with the main training process through different UIAutomotor servers. The main training process sends high-level instructions to UIAutomotor servers (such as reset and step), while UIAutomotor servers parse high-level instructions into low-level UI commands (such as typing a character and tapping at a coordinate) and such UI commands are executed by the emulator processes. When an exception is thrown in the emulator, the UIAutomotor examines if it is recoverable (e.g. an UI command takes too long to execute in the emulator) and reset the emulator process if it is not. When an exception is thrown in the UIAutomotor server, the main training process stops and resets the UIAutomotor server to ensure data correctness.

This design can easily be scaled up to a multi-machine setting. As illustrated in Figure 15, one host machine equipped with GPU accelerator has a local copy of the current policy π_t, and distributes the policy to all worker machines equipped with only one GPU and multiple CPUs. Each worker machine will then collect trajectories of different tasks using π_t. After all collection processes are synchronized, the host machine gathers all the trajectories together to update the policy to π_{t+1}. This process keeps iterating until the policy converges.

The performance boost with respect to the number of worker machines is nearly linear, as demonstrated in Figure 16, where we conduct experiments that examine the scaling performance of our parallel emulator. Our distributed emulator

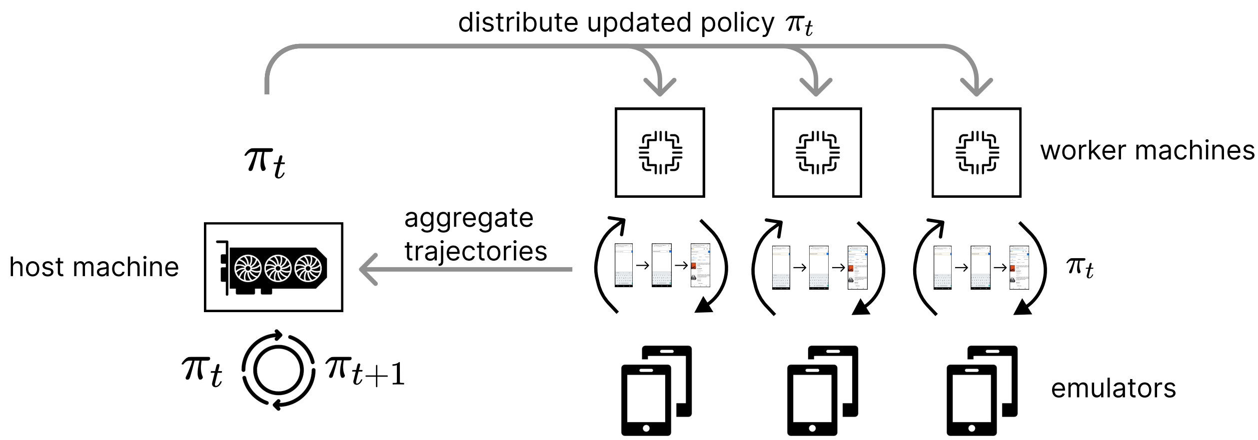


Figure 15: Multi-machine parallel emulator execution. The host machine is equipped with GPU accelerators and the worker machines are equipped only with CPUs. The policy update is executed on the worker machine and the trajectory collections are executed distributedly on the worker machines and aggregated by the host machine.

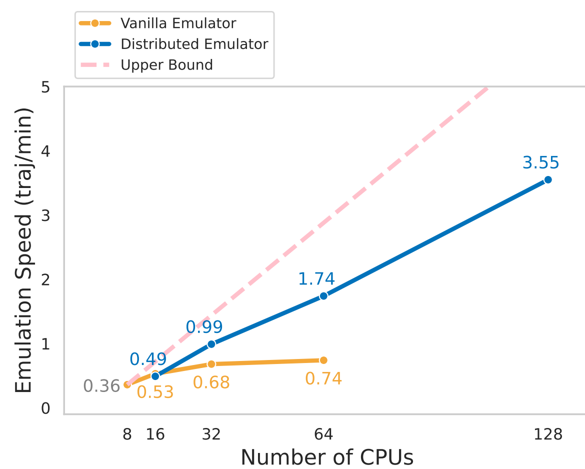


Figure 16: **Emulation speed w.r.t number of CPUs used.** The upper bound can only achieved when there is no communication and error handling cost. Our design of distributed emulator can significantly improve the efficiency of emulation compaared to the vanilla method of running all emulations over the same instance.

that runs emulations across multiple servers can reliably collect data with up to 64 parallel emulators on 128 CPUs with near-linear speedup. In contrast, a naive baseline that runs all parallel emulations on the same server achieves much inferior performance (0.74 compared to 1.74 trajs/min using 64 CPUs).

H. Autonomous evaluator details

Our autonomous evaluator gives a reward to each observation we get. The observation is composed of the current screenshot of device and the task. The evaluator gives a reward of 1 if the screenshot shows a completion of the task, and will terminate the POMDP as a result result.

The optimized prompt is shown in Figure 17 and Figure 18 for General and Web Shopping subsets respectively.

Prompt

You're an expert in evaluating whether the Screenshot successfully completes the Task.

=====Examples=====

Screenshot: {train_1.png}
Task: Open the settings.
Q: What should I expect to see on the screenshot if I've opened the settings?
A: I should expect to see I'm in the settings app. The screenshot shows the home screen of a mobile device, with various app icons displayed, including the settings app icon, but the settings app is not opened.
Status: failure

Screenshot: {train_2.png}
Task: Find hotels in washington dc
Q: What should I expect to see on the screenshot if I've searched for hotels in Washington, DC?
A: I should expect to see I'm in a search results page for hotels in Washington, DC. The screenshot shows a Google search page with the search field populated with the query "hotels in washington dc" and a list of suggested searches related to hotels in Washington, DC, but it does not show any search results for hotels in Washington, DC.
Status: failure

Screenshot: {train_3.png}
Task: What's a good restaurant in Portland?
Q: What should I expect to see on the screenshot if I've searched for a good restaurant in Portland?
A: I should expect to see I'm in a search results page for a good restaurant in Portland. The screenshot shows a Google search page with a search input field for "good restaurant in portland" and a map results preview showing business locations near Portland, like "Li Pigeon", "Portland City Grill", and "Higgins",
Status: success

... (more cases)

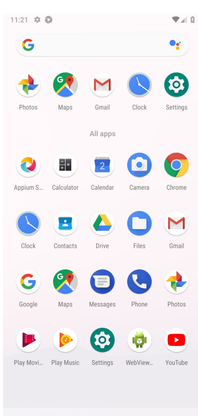
=====Your Turn=====

Screenshot: {test.png}
Task: {task_this_traj}
Respond in this format:
Q: What should I expect to see on the screenshot if I've <repeat the task>?
A: I should expect to see <first expectation, then what's in the given screenshot.>
Status: success or failure (don't return anything else)
Start with "Q:".

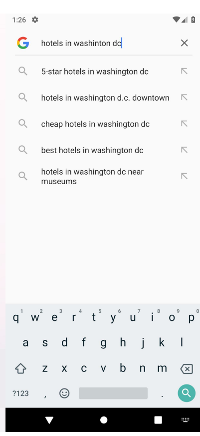
Response

Q: What should I expect to see on the screenshot if I've searched for the price of a 12' ladder at Home Depot?
A: I should expect to see the price of a 12' ladder at Home Depot; the screenshot shows a search result page for the price of a 12' ladder, with some product advertisements showing prices from Home Depot.
Status: success

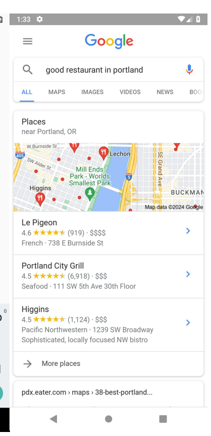
Image Sources



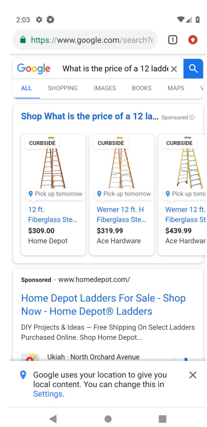
train_1.png



train_2.png



train_3.png



test.png

Figure 17: Prompt for our autonomous evaluator for tasks in AitW General subset.

Prompt

You're an expert in evaluating whether the Screenshot successfully completes the Task.

=====Examples=====
Screenshot: {train_1.png}
Task: Go to bestbuy.com
Q: What should I expect to see on the screenshot if I've gone to bestbuy.com?
A: I should expect to see I'm in the Best Buy website, which usually shows the best buy logo with some featured products and categories. The screenshot shows I'm searching for "bestbuy.com" in the Google search (with some search suggestions) instead of being in the Best Buy website.
Status: failure

Screenshot: {train_2.png}
Task: Go to ebay.com, search for "corsair k70"
Q: What should I expect to see on the screenshot if I've gone to ebay.com AND searched for "corsair k70"?
A: I should expect to see I'm in the eBay website and search results for "corsair k70". The screenshot shows I'm in the eBay ebay website with some search suggestions for "corsair k70", but it does not show search results of the product, which usually includes price and the product details.
Status: failure

Screenshot: {train_3.png}
Task: Go to ebay.com, search for "lenovo thinkpad"
Q: What should I expect to see on the screenshot if I've gone to ebay.com AND searched for "lenovo thinkpad"?
A: I should expect to see I'm in the eBay website and search results for "lenovo thinkpad". The screenshot shows I'm in the eBay website and have several search results for "lenovo thinkpad".
Status: success

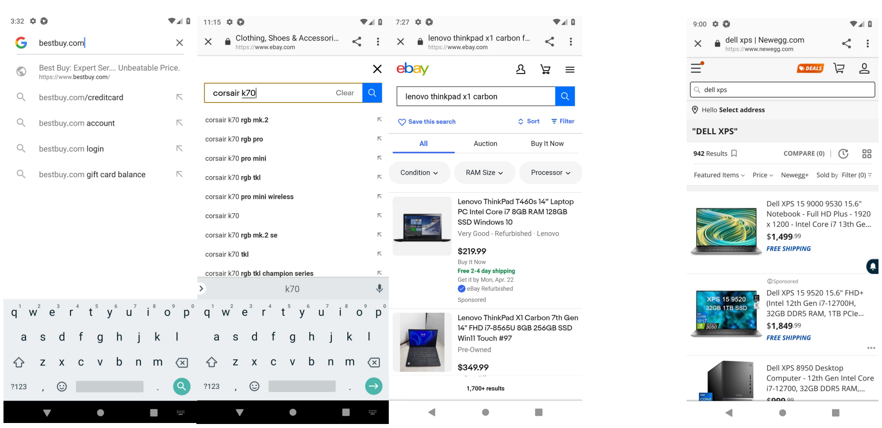
... (more cases)

=====Your Turn=====
Screenshot: {test.png}
Task: {task_this_traj}
Respond in this format:
Q: What should I expect to see on the screenshot if I've <repeat the task>?
A: I should expect to see <first expectation, then what's in the given screenshot.>
Status: success or failure (don't return anything else)
Start with "Q:".

Response

Q: What should I expect to see on the screenshot if I've searched for the price of a 12' ladder at Home Depot?
A: I should expect to see the price of a 12' ladder at Home Depot; the screenshot shows a search result page for the price of a 12' ladder, with some product advertisements showing prices from Home Depot.
Status: success

Image Sources



train_1.png train_2.png train_3.png test.png

Figure 18: Prompt for our autonomous evaluator for tasks in AitW Web Shopping subset.

I. Zero-shot Baseline Details

Figure 19 shows the prompt that we used for testing the Set-of-Marks performance for GPT-4V and Gemini 1.5 Pro. This prompt is directly taken from Yang et al. (2023).

Prompt

"You are an agent that is trained to perform some basic tasks on a smartphone. You will be given a \nsmartphone screenshot. The interactive UI elements on the screenshot are labeled with numeric tags starting from 1. The \nnumeric tag of each interactive element is located in the center of the element.\n\nYou can call the following functions to control the smartphone:\n\n1. tap(element: int)\nThis function is used to tap an UI element shown on the smartphone screen.\n\"element\" is a numeric tag assigned to an UI element shown on the smartphone screen. \nA simple use case can be tap(5), which taps the UI element labeled with the number 5.\n\n2. text(text_input: str)\nThis function is used to insert text input in an input field/box. text_input is the string you want to insert and must \nbe wrapped with double quotation marks. A simple use case can be text(\"Hello, world!\"), which inserts the string \n\"Hello, world!\" into the input area on the smartphone screen. This function is usually callable when you see a keyboard \nshowing in the lower half of the screen.\n\n3. long_press(element: int)\nThis function is used to long press an UI element shown on the smartphone screen.\n\"element\" is a numeric tag assigned to an UI element shown on the smartphone screen.\nA simple use case can be long_press(5), which long presses the UI element labeled with the number 5.\n\n4. swipe(element: int, direction: str, dist: str)\nThis function is used to swipe an UI element shown on the smartphone screen, usually a scroll view or a slide bar.\n\"element\" is a numeric tag assigned to an UI element shown on the smartphone screen. \"direction\" is a string that \nrepresents one of the four directions: up, down, left, right. \"direction\" must be wrapped with double quotation \nmarks. \"dist\" determines the distance of the swipe and can be one of the three options: short, medium, long. You should \nchoose the appropriate distance option according to your need.\nA simple use case can be swipe(21, \"up\", \"medium\"), which swipes up the UI element labeled with the number 21 for a \nmedium distance.\n\n5. grid()\nYou should call this function when you find the element you want to interact with is not labeled with a numeric tag and \nother elements with numeric tags cannot help with the task. The function will bring up a grid overlay to divide the \nsmartphone screen into small areas and this will give you more freedom to choose any part of the screen to tap, long \npress, or swipe.

The task you need to complete is to **How much does a 2 bedroom apartment rent for in Denver?**.

Your past actions to proceed with this task are summarized as follows: **None**

Now, given the documentation and the following labeled screenshot, you need to think and call the function needed to proceed with the task. Your output should include three parts in the given format:
Observation: <Describe what you observe in the image>
Thought: <To complete the given task, what is the next step I should do>
Action: <The function call with the correct parameters to proceed with the task. When you are certain that the task is successfully done and the goal is reached as of the current observation, you should output FINISH. You cannot output anything else except a function call or FINISH \nin this field.>
Summary: <Summarize your past actions along with your latest action in one or two sentences. Do not include the numeric \ntag in your summary>\nYou can only take one action at a time, so please directly call the function."

Figure 19: Set-of-Marks prompting. The boldened inputs can be changed according to our goal. The task changes for every different task. The past actions change as we take actions (it is None now since this is the prompt for the first round).

J. Other Experiments

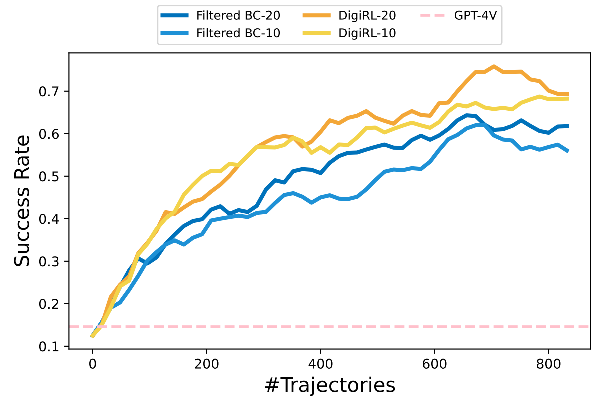


Figure 20: **Success rate with different horizon length** ($H \in \{10, 20\}$)under different methods on the AiTW Google Search task set.

J.1. Horizon Limit

We investigate the horizon limit of filtered BC and DigiRL on the AitW General subset. As most tasks can be effectively solved within 10 steps, we specify two horizon limits: a sufficient horizont $H = 10$, and a redundant horizon $H = 20$. Results show that a redundant horizon introduces significantly faster learning speed for both filtered BC and DigiRL, presumbaly because longer horizon means more opportunity to try in a single trajectory. In both horizon settings, we observe the DigiRL offers a significant speedup of around 100 trajectories over Filtered BC.

K. Hyperparameters

Hyperparameters for both Filtered BC and DigiRL are carefully tuned through binary search on the training set of General and Web Shopping subsets. The final choice of hyperparameters for both methods can be found in Table 5. As shown in the table, the only hyperparameters introduced by DigiRL are supervised training hyperparameters for the value function and instruction value function (including number of iterations and learning rate) and GAE λ.

Table 5: Hyperparameters for All Experiments

Method	Hyperparameter	Offline	Offline-to-Online
Filtered BC	actor lr	3e-3	3e-3
	batch size	128	128
	rollout trajectories	-	16
	replay buffer size	-	5000
	rollout temperature	-	1.0
	maximum gradient norm	0.01	0.01
	actor updates per iteration	20	20
	number of iterations for offline actor updates	10	10
DigiRL	actor lr	3e-3	3e-3
	value function lr	3e-3	3e-3
	instruction value function lr	3e-3	3e-3
	instruction value function lr	3e-3	3e-3
	batch size	128	128
	rollout trajectories	-	16
	replay buffer size	-	5000
	rollout temperature	-	1.0
	maximum gradient norm	0.01	0.01
	GAE λ	0.5	0.5
	actor updates per iteration	20	20
	value function updates per iteration	5	5
	instruction value function updates per iteration	-	5
	number of iterations for offline actor updates	10	10
	number of iterations for offline value function updates	20	20
	number of iterations for offline instruction value function updates	-	20

Table 6: Hyperparameters for DigiRL and Filtered BC on both General and Web Shopping subset of AitW..