Digi-Q: Transforming VLMs to Device-Control Agents via Value-Based Offline RL

Anonymous authors
Paper under double-blind review

Abstract

Most paradigms for building foundation model agents rely on prompting or fine-tuning on existing demonstrations, but this is not sufficient in dynamic environments (e.g., mobile device control). In theory, while on-policy reinforcement learning (RL) should address these limitations, this approach itself is not quite effective at leveraging existing agentic data, especially when it is of low quality. An approach to address this issue is to use offline value-based RL but realizing value-based RL for device-control agents has been elusive due to of stability and efficiency associated with running TD-learning at scale with vision-language models (VLMs). In this paper, we develop a scalable value-based RL approach called Digi-Q that makes it possible to train VLM agents with TD-learning. We situate our study in building GUI agents for Android devices. The key idea in Digi-Q is to perform TD-learning on a frozen, intermediate-layer representation of a VLM rather than training the whole VLM itself. Doing so successfully requires an initial phase of fine-tuning to prime VLM representations to feature actionable information that is critical for TD-learning. When done correctly, our approach is able to attain better performance per-unit compute FLOPS. To make maximal use of the learned Q-function, we devise a novel best-of-N policy extraction operator that imitates the best actions out of multiple candidate actions from the current policy as ranked by the value function. With no REINFORCE-style policy gradients that need careful tiuning and an efficient TD-learning approach, Digi-Q outperforms prior methods on user-scale device control tasks in Android-in-the-Wild, attaining 9.9% of relative improvement over prior best-performing offline RL method in this domain.

1 Introduction

Foundation models (OpenAI, 2024a; GeminiTeam, 2024) open up possibilities of building agents that make intelligent decisions in the real world (Liu et al., 2023). While prompting off-the-shelf language models with specific instructions is one way to get them to make decisions, this is not good enough for attaining goals and maximizing rewards that are critical in downstream tasks (Zeng et al., 2023; Chen et al., 2023). Part of the reason is the lack of sufficiently diverse decision-making data for training large models (Gur et al., 2023), but the bigger reason is that simply imitating Internet data is not good enough for training models how to act intelligently, reduce uncertainty, and achieve goals in non-stationarity real-world decision making settings (Bai et al., 2024; Ma et al., 2024).

Reinforcement learning (RL) provides a general approach for fine-tuning agents that avoids the shortcomings of imitation and prompting, by explicitly training the policy to maximize rewards (Zhou et al., 2024b; Verma et al., 2022; Snell et al., 2023; Abdulhai et al., 2023). The best performing RL methods for multi-step agentic tasks typically run basic policy gradients (Yao et al., 2023) and Monte Carlo value estimations (Bai et al., 2024; Putta et al., 2024) on on-policy data. Relying almost exclusively on (near) on-policy rollout data is not only expensive (Haarnoja et al., 2018) at real user-scales (e.g., when acting on a phone) but also sub-optimal, since using previously collected offline data can lead to substantial improvements if done correctly using value-based RL methods (Ball et al., 2023). A natural approach to remove this dependence on on-policy data is to transition to using efficient value-based offline RL methods. These methods use Q-functions, which can be trained offline, to predict future rewards from a given state and action pair, and optimize the policy against this value function. The value-based paradigm is appealing due to the substantial gains it offers in terms of sample-complexity (Haarnoja et al., 2019; Mnih et al., 2013) and historical data reuse, in a way that improves over simply mimicking successful data or relying on purely on-policy data.

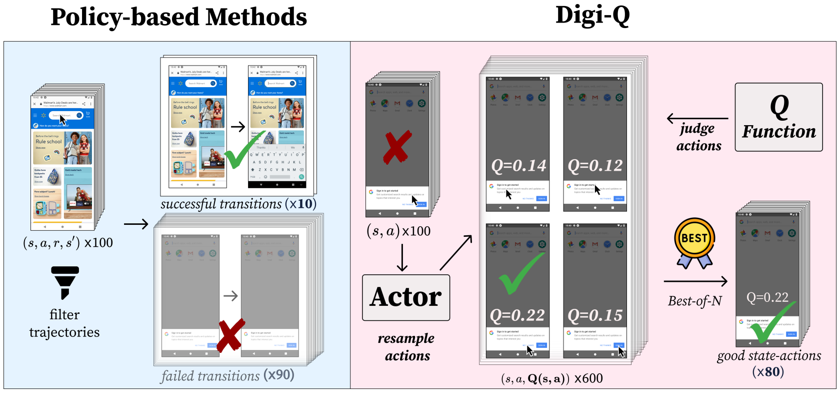


Figure 1: **Comparing Digi-Q with alternate policy-based methods.** Typically, policy-based methods need online data to improve. In contrast, Digi-Q learns a Q-function through TD-learning on offline data and queries this Q-function at multiple actions to learn a policy via the Best-of-N policy extraction approach.

Can we train VLM device-control agents at scale with value-based RL? To answer this question, we develop a value-based RL approach, Digi-Q, for reliably and efficiently training VLM agents at scale, and show its efficacy in the domain of device-control (Rawles et al., 2023). We situate our study in the offline RL problem setting, where we must train agents entirely from historical data since it provides a challenging test-bed for building value-learning algorithms, while being fast to iterate on (due to no requirement for running on-policy training against a simulator). Digi-Q trains a state-action value function (i.e., a Q-function). To train Q-functions effectively, Digi-Q handles a number of challenges posed by value-based offline RL at scale: **(i)** instability in training large models associated with running temporal-difference (TD) learning with large models (Kumar et al., 2021) and **(ii)** inefficiency of TD backups per unit amount of compute (Chebotar et al., 2023) (see Section 5.2). Digi-Q does so by performing TD-learning on a frozen intermediate layer *representation* of the VLM instead of training all parameters of the VLM. For attaining best performance though, representations from off-the-shelf models are not good enough since they often do not pay feature information crucial for predicting actions or their consequences. Therefore, Digi-Q prescribes running an initial phase of representation fine-tuning to prime representations of a VLM to be more amenable to TD-learning.

Once the Q-function is trained, Digi-Q uses it to train a VLM policy that maximizes this Q-function while staying close to the dataset since we operate in an offline RL setting. Using a learned Q-function for policy extraction offers numerous benefits as shown in Figure 1: first, while state-only Monte-Carlo value functions, typical in prior work (Bai et al., 2024; Zhai et al., 2024), can only evaluate the efficacy of a single action that was actually executed at any given state, if the learned Q-function is somewhat accurate, we can obtain reasonable estimates for future expected reward for *multiple* actions at the same state, without needing to actually roll these candidates out. This allows us to develop an "aggressive" and Best-of-N policy training objective that trains the policy to imitate the best-rated action per the Q-function. In contrast, off-the-shelf agentic RL methods need to use variants of imitation learning (Pan et al., 2024) or policy gradients that can be quite finicky, especially in the offline setting (Ahmadian et al., 2024). Overall, Digi-Q leads to effective perofrmance.

The main contribution of this work is Digi-Q, an approach to train VLM device-control agents via value-based offline RL. Digi-Q represents and trains Q-functions on top of intermediate representations from a VLM, fine-tuned to be aware of actionable information. Then Digi-Q utilizes a Best-of-N policy extraction objective to make most effective use of the Q-function in obtaining a policy. The agent produced by running Digi-Q on offline data outperforms prior approachrs that also extract policies from offline data in the problem setting of Android device control (Rawles et al., 2023) with 9.9% of relative improvement over prior best-performing prior method, even though these domains remain challenging for state-of-the-art proprietary models (Liu et al., 2024c; Bai et al., 2024). *To the best of our knowledge, this work is the first to successfully apply offline RL with TD-learning to realistic agent tasks with foundation models and show significantly improved performance.*

2 Related Work

RL for training GUI and device-control agents. Due to their reasoning, perception, and generation capabilities, LLMs and VLMs have been applied extensively to build agents to navigate web pages (Zhou et al., 2024a; Koh et al., 2024a; Deng et al., 2023; Zheng et al., 2024; He et al., 2024) and GUI interfaces (Bai et al., 2024; Yan et al., 2023; Hong et al., 2023; Rawles et al., 2023; 2024; Zhang & Zhang, 2024). In contrast to using off-the-shelf proprietary models (Zheng et al., 2024; Yan et al., 2023; Zhang et al., 2023; He et al., 2024) or fine-tuning them with a small amount of human demonstrations (Hong et al., 2023; Zhang & Zhang, 2024; Zeng et al., 2023), RL provides the advantage of optimizing task-specific reward and goal-oriented behavior, which is important in dynamic and non-stationary environments especially when human demonstrations are stale (Bai et al., 2024; Zhou et al., 2024b; Putta et al., 2024; Pan et al., 2024; Song et al., 2024). However, most successful applications of RL for real-world GUI agent tasks use less efficient RL algorithms such as (nearly) on-policy policy gradient or filtered imitation learning algorithms (Bai et al., 2024; Putta et al., 2024; Song et al., 2024; Koh et al., 2024b). In traditional RL, off-policy and offline RL algorithms that train a state-action value function (i.e., a Q-function) via temporal-difference learning (TD-learning) are known to be substantially more sample efficient and effective (Mnih et al., 2013; Haarnoja et al., 2018; Fujimoto et al., 2018; Kumar et al., 2020). To the best of our knowledge, our work is the first to scale value-based Bellman bootstrapping to convert VLMs into strong device-control agents, starting from the setting of training entirely on offline data.

From an algorithmic standpoint, the closest work to ours that trains agents with Q-functions is ArCHer (Zhou et al., 2024b), which builds a hierarchical framework for developing RL algorithms for agents and largely presents results on simplified environments (Yao et al., 2023). While we do use the hierarchical actor-critic abstraction in ArCHer to formalize our approach, the methodology for running RL at scale is substantially different from Zhou et al. (2024b) along the use of frozen VLM representations and a policy extraction approach based on best-of-N policy extraction. Our experiments in Section 5 show that Digi-Q is much more effective (about a 20% improvement as shown in Section 5.3) than the policy-gradient algorithm used by Zhou et al. (2024b). This justifies the benefits of seemingly simple, yet an effective design of our approach. Other works (Zhai et al., 2024) train VLMs with on-policy PPO (Schulman et al., 2017; Chen et al., 2024). Finally, Chen et al. (2024) runs RL on top of frozen VLM representations as well, although unlike us they do not fine-tune the VLM to make these representations more amenable for fitting value functions. We find that this representation fine-tuning phase in Digi-Q is critical to obtaining a good Q-function. Moreover, while this prior work largely instantiates existing deep RL algorithms instead of devising a more scalable and effective value-based RL algorithm that we aim to do.

Challenges of value-based RL with foundation models. Despite the efficiency and data reuse benefits of value-based RL algorithms, they can be unstable and computationally inefficient if not treated carefully, particularly the case for large foundation models with billions of parameters. This instability stems from two aspects: **(1)** prior work has often found it hard and unstable to train value functions via Bellman backups and TD-learning (Kumar et al., 2021; 2022; Chebotar et al., 2023), which is challenging at scale. To address this, Chebotar et al. (2023) had to employ a combination of conservative regularization (Kumar et al., 2020) and regularization with n-step returns (Hessel et al., 2018) resulting in a complex approach; **(2)** policy extraction from trained Q-functions often utilizes policy gradient approaches with a "negative gradient" term (Tajwar et al., 2024) that can be unstable with offline data. This has largely resulted in the community focusing on on-policy or filtered imitation learning methods. However, Park et al. (2024) show that supervised regression methods such as AWR (Peng et al., 2019) can lead to slow convergence and poor asymptotic performance. To address challenge **(1)**, Digi-Q runs TD-learning on top of frozen VLM representations, but after a fine-tuning phase to make them more amenable to representing Q-functions and to address **(2)**, we introduce a novel Best-of-N based policy extraction loss.

3 Preliminaries and Problem Setup

We build value-based RL algorithms for transforming VLMs into agents, focusing on the offline RL problem setting. We situate ourselves in the domain of building agents that can perform pixel-based interactions on virtual devices, following similar protocols as past work (Bai et al., 2024). In this section, we will discuss the setup for this problem, followed by reviewing terminology, notation, and background information that would be useful in developing our approach in the next section.

3.1 Problem Setup: Android Device Control

We scope our study in the domain of pixel-based Android Device Control (Bai et al., 2024; Zhang et al., 2023; Rawles et al., 2023). Each episode in this domain starts with a fully-functioning Android emulator reset to the home screen and a task is randomly drawn from a task pool. The agent needs to complete the task through pixel-based interactions with the device as illustrated in Figure 1. The actions that the agent can take are primitive pixel-level commands such as clicking at a coordinates and typing text. Concretely, given a screenshot of a phone, we want the agent to output a string command such as “click (0.8, 0.2)”to be executed in the environment at the current step, where 0.8 and 0.2 are 0-1 normalized x-y coordinates in the screen. This domain is known to be more general and challenging than web navigation alone or link-based device control (Bai et al., 2024), and present many real-world challenges of device stochasticity such as unpredictable distractors like pop-ups and technical glitches like incomplete website loading. Following Pan et al. (2024); Bai et al. (2024), the agents are evaluated via binary 0/1 rewards from a proprietary model (GeminiTeam, 2024) that makes a verdict of whether the specific task has been completed at each step. More importantly, we want to train device-conrtol agents, in an ***offline*** setting where we are given a static dataset $\mathcal{D}$ storing historical past interaction data that this agent must learn from.

3.2 Reinforcement Learning Definitions

To design a value-based RL approach for training device-control agents, we subscribe to the hierarchical MDP framework from Zhou et al. (2024b). This hierarchical MDP consists of two MDPs: **(1)** a high-level MDP where each step is an interaction with the external environment, and **(2)** a low-level MDP embedded inside each action in the high-level MDP, where each step is an independent natural language token. Terminology wise, we define the **state** s_t in the high-level MDP as the log of the interaction history of the agent with the environment thus far concatenated to the current observation. Each **action** a_t in the high-level MDP is a sequence of tokens that are directly applied to interact with the environment such as “type box [2]: wikipedia of chocolate”. Each action a_t^h in the low-level MDP is an individual token output while each state in the low-level MDP contains the high-level state s_t and all the action tokens $a_t^{1:h-1}$ output before the current token.

The Q-function for a given policy π is the expected cumulative reward of a particular action at the current step, and then following policy π thereafter: $Q^\pi(s_h, a_h) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_{h+t}, a_{h+t})\right]$. The value function of a policy π, $V^\pi(s_h)$, is defined as the expected Q-value, $Q^\pi(s_h, a_h)$, where actions a_h are sampled from the policy π. The advantage function $A^\pi(s_h, a_h)$ corresponds to the relative benefit of taking action a_h in state s_h, and is computed as the difference between the Q-value and the value of the state under the policy: $A^\pi(s_h, a_h) = Q^\pi(s_h, a_h) - V^\pi(s_h)$. The goal of RL is to train a policy that can produce token sequences that maximize the cumulative rewards $\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t)$, over the course of a rollout. Our offline value-based approach will model an independent action-value Q-function Q and a state-only value function V parameterized by parameters θ and ψ respectively while the policy is parameterized by ϕ. Both Q- and V- functions are instantiated by a small MLP layer on top of a VLM backbone. We use $\theta_{\text{VLM}}, \psi_{\text{VLM}}$ to represent the parameters of the VLM backbone and similarly $\theta_{\text{MLP}}, \psi_{\text{MLP}}$ for parameters of the MLP head. We will denote the last layer representations of these VLM backbones as $f_{\theta_{\text{VLM}}}(s, a)$ and $f_{\psi_{\text{VLM}}}(s)$.

3.3 Background: ArCHer Framework for Deriving Off-Policy RL Algorithms

The core idea behind the ArCHer framework (Zhou et al., 2024b) is to pose training of foundation model agents as a hierarchical RL problem. While this can give rise to many RL algorithms, Zhou et al. (2024b) show that a convenient way to instantiate this approach is to learn a value function in the high-level MDP and a policy in the low-level token MDP. The high-level critic and low-level actor are then optimized against each other similarly to standard actor-critic RL.

$$J_Q(\theta) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[(Q_\theta(s,a) - r - \gamma V_{\bar{\psi}}(s'))^2\right]. \tag{1}$$

$$J_V(\psi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi_\phi(\cdot|s)}\left[(V_\psi(s) - Q_{\bar{\theta}}(s,a))^2\right]\right]. \tag{2}$$

$\bar{\theta}$ and $\bar{\phi}$ are the delayed target networks (Mnih et al., 2013) for stability and they are updated as an exponential moving average of θ and ϕ. The instantiated algorithm from Zhou et al. (2024b) supports policy extractions through REINFORCE policy gradient:

$$J_\phi(\pi) = \mathbb{E}_{s_c\sim\mathcal{D}, a_t^{1:L}\sim\pi(\cdot|s_c)}\left[\sum_{i=1}^{L} A(s_c, a_t^{1:L}) \log \pi_\phi(a_t^i|s_c, a_t^{1:i-1})\right]. \tag{3}$$

While our approach will utilize a similar framework to conceptualize the method, the choice of actor and critic updates employed are substantially different. The practical approach in Zhou et al. (2024b) did not work very well in our experiments with VLMs.

4 Digi-Q: Training VLM Policies with Offline Value-Based RL

To obtain an effective offline RL method, Digi-Q addresses a number of challenges with value-based RL at large scale. First, to avoid pathological behavior of TD-backups with large models (Zhou et al., 2024b; Snell et al., 2023; Abdulhai et al., 2023; Chebotar et al., 2023) and to avoid the computational costs associated with training an entire billion-parameter VLM with TD-learning, we train Q-functions on top of frozen VLM representations. Since VLMs are not trained on substantial quantities of decision-making data, off-the-shelf VLMs largely do not accurately represent *actionable* elements of an input scene. To address this, Digi-Q fine-tunes VLM representations before TD-learning.

AWR (Peng et al., 2019)), which updates the policy with a single action per state, training a Q-function enables simultaneous estimation of returns for multiple action candidates. These candidates can be leveraged to improve the policy more efficiently, benefiting from variance reduction, especially with a well-estimated critic. Digi-Q leverages this idea to propose a best-of-N reranking-based policy extraction operator, which is more stable and effective than policy gradient or advantage-weighted regression. This method is illustrated in Figure 2 and described below.

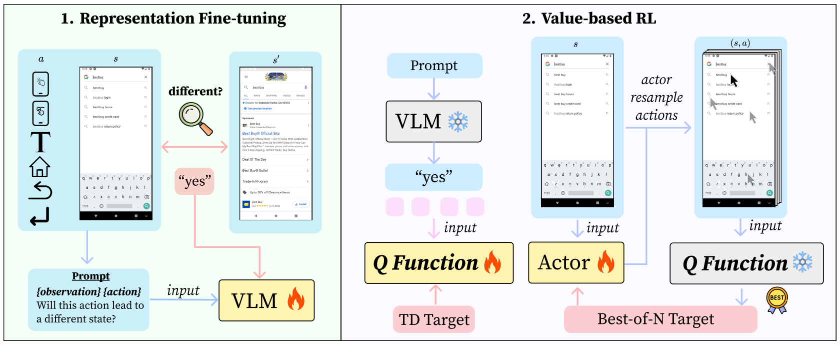


Figure 2: **Method overview.** Blue arrows represent forward data flows, while red arrows represent the flows of learning targets used for back propagation. Our method first goes through a representation fine-tuning stage to extract actionable features from the VLM. TD-learning is then performed on top of frozen VLM representations, followed by best-of-N policy extraction approach.

4.1 Training VLM Q-Functions via TD-Learning and Representation Fine-Tuning

Fine-tuning large VLMs end-to-end via TD-learning can encounter challenges, making it practical to train a separate value function on top of a frozen VLM. However, most VLMs are not designed to model actionable information in scenes or predict outcomes of actions. If VLM representations fail to capture actionable details, training a state-action Q-function $Q(s, a)$ via TD-learning risks degenerating into a state-only value function $V(s)$ (ignoring actions) or diverging due to amplified noise in Q-value estimates for out-of-distribution actions during TD-backups. Preliminary experiments show that while open-source VLMs like LLaVa-1.5 (Liu et al., 2024a) can answer scene-based questions, they often fail to predict future impacts of actions, e.g., "does this click lead to a new page on eBay?"

Thus, Digi-Q first fine-tunes representations of a VLM with an binary classification objective to enable it to pay attention to actionable features of an input scene in device control. Once fine-tuned, the representations of this VLM are used to train a Q-function represented using a small MLP on top of the frozen representation. Not only is this more stable but it cuts down computational costs since only 1% of all the VLM parameters are now trained via TD-learning.

Approach. Our representation fine-tuning objective for device control is constructed as follows: given a transition pair (s_t, a_t, s_{t+1}) drawn from a replay buffer, our objective aims to model if and how the next state s_{t+1} will change from the current state and action (s_t, a_t). Our observation is that in device control problems, a useful action should lead to a substantial visual change in the pixel values of a scene (e.g., successfully typing a search query and pressing enter on google.com should cause the page to change substantially to now show a list of search results).[1] Equipped with this insight, we construct positive and negative tuples of transitions (s_t, a_t, s_{t+1}), where the positive tuples consists of transitions change the state significantly (i.e., larger than a threshold ϵ on the ℓ_2

[1] Note that while this heuristic does not filter out actions that appear useful erroneously due to glitches in the environment (e.g., page failed to load) but this is acceptable as long as glitches do not dominate this set.

image distance) and the negative tuples are the remaining transitions. This is equivalent to assigning a binary {0, 1} label to each transition:

$$y_t = \begin{cases} 0, & d(s_t, s_{t+1}) < \epsilon \\ 1, & \text{otherwise} \end{cases}$$

Now, the VLM is trained to produce this 0-1 label y_t given a state-action tuple (s_t, a_t) using a binary cross-entropy loss on its paramters θ_{VLM}:

$$J_{\mathcal{P}}(\theta_{\text{VLM}}) = -\mathbb{E}_{s_t, a_t \sim \mathcal{D}} \left[y_i \log \mathcal{P}_{\theta_{\text{VLM}}}('\text{yes}'|s_t, a_t) + (1 - y_i) \log \mathcal{P}_{\theta_{\text{VLM}}}('\text{no}'|s_t, a_t) \right], \quad (4)$$

where $\mathcal{P}_{\theta_{\text{VLM}}}$ is the next-token distribution from the VLM backbone.

After this phase of representation fine-tuning, we freeze the parameters of VLM, and extract the embedding of the yes/no token output to serve as the input representation of (s_t, a_t) to the Q function. We now run a TD-learning objective from Equation 1 in Section 3 to train the Q-function.

Note that Equation 1 also utilizes a parameterized value function. Since the value function does not depend on the action, we are able to directly use internal representations of an off-the-shelf VLM, without requiring any phase of initial fine-tuning. On top of the frozen representations from the VLMs, our value functions $\theta_{\text{MLP}}, \psi_{\text{MLP}}$ is optimized with the TD loss as in Equation 5 and 6.

$$J_Q(\theta_{\text{MLP}}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[(Q_{\theta_{\text{MLP}}}(f_{\theta_{\text{VLM}}}(s, a)) - r - \gamma V_{\bar{\psi}_{\text{MLP}}}(f_{\bar{\psi}_{\text{VLM}}}(s')))^2 \right]. \quad (5)$$

$$J_V(\psi_{\text{MLP}}) = \mathbb{E}_{s \sim \mathcal{D}} \left[\mathbb{E}_{a \sim \pi_\phi(\cdot|s)} \left[(V_{\psi_{\text{MLP}}}(f_{\psi_{\text{VLM}}}(s)) - Q_{\bar{\theta}_{\text{MLP}}}(f_{\bar{\theta}_{\text{VLM}}}(s, a)))^2 \right] \right]. \quad (6)$$

4.2 Best-of-N Policy Extraction Against the Learned Q-Function

Given a learned Q-function, we will now use it to extract a policy in an efficient and reliable manner. Perhaps the most straightforward approach for doing so is to use REINFORCE policy gradient to optimize the learned policy, however, this approach can be quite brittle with off-policy data (Swaminathan & Joachims, 2015; Zhou et al., 2024b). Moreover, the presence of a "negative gradient" term Tajwar et al. (2024) (i.e., a term where the policy gradient multiplies the log likelihood by a negative-valued advantage) means that careful tuning of learning rates and interleaving policy and critic updates must be done to attain good performance (see Zhou et al. (2024b) Section 5.7 for a discussion of these challenges, and experiment results shown in Table 3). While advantage-weighted supervised learning (i.e., AWR (Peng et al., 2019)) avoids this instability issue, it can be conservative in its KL divergence with the behavior policy (Table 3)..

To build a stable yet aggressive policy training method, Digi-Q modifies AWR to make it more aggressive, by leveraging the insight that access to the functional form of a learned Q-function allows for estimating values for multiple N actions at any given state. After computing Q-values for multiple action candidates, we can imitate the best action. This would produce substantially more aggressive updates than single-action AWR, without needing a negative gradient. Theoretically, this is because the implicit KL constraint against the data-generating policy that makes AWR conservative, is now much less of a problem with our multiple-action approach, since this implicit constraint is enforced against the Best-of-N policy (Cobbe et al., 2021). The best-of-N policy is already better than the data collection policy for large values of N, meaning that we do not lose utility of off-policy data.

Concretely, given any state s, we sample N action token sequences (i.e., actions in the high-level MDP) from the learned policy: $a_1, \cdots, a_N \sim \pi_\beta(\cdot|s)$, where π_β is a behavior-cloned policy from the offline dataset. Now, we rank these actions according to the Q-values obtained from the critic trained previously. The policy is then trained to imitate the best of these N actions as long as this best action also attains a positive advantage. Formally, this means that the policy is optimized as.

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, a_i \sim \pi_\beta(\cdot|s_t)} \left[\sum_{i=1}^{N} \delta(a_i) \sum_{h=1}^{L} \log(a_i^h | s_t, a_i^{1:h-1}) \right], \quad (7)$$

where $\delta(a_i) = \mathbb{1}\{a_i = \arg\max_i Q(s_t, a_i) \ \ \& \ \ Q(s_t, a_i) - V(s_t) > 0\}$. This approach allows us to make fairly aggressive policy updates, while also being stable and efficient due to a log loss.

4.3 Putting it Together: Algorithm and Implementation Details

A pseudocode of our overall algorithm is shown in Appendix A. After initially fine-tuning the VLM through the unsupervised pre-training scheme in Section 4.1, Digi-Q first trains Q and V-functions

			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	25.0	25.0	31.3	38.5
		AutoUI	27.7	22.9	20.7	25.0
	Offline	Filtered BC	51.0 ± 0.9	54.5 ± 1.3	37.2 ± 4.7	43.8 ± 1.7
		DigiRL	53.5 ± 2.7	59.0 ± 4.7	43.1 ± 3.6	47.6 ± 4.2
		Digi-Q (Ours)	**61.8 ± 1.0**	**68.1 ± 1.3**	**49.0 ± 0.9**	**49.7 ± 3.5**

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. To be consistent with prior work (Bai et al., 2024), results are evaluated with the autonomous evaluator with the first 96 instructions in the train and test set.

before performing gradient updates on the actor, where the VLM backbone for the V-function is kept frozen from the pre-trained checkpoint. The usage of V-functions follows from Zhou et al. (2024b); Snell et al. (2023) to improve training stability and make calculations of high-level advantages easier. The actor is represented on top of a separate VLM and is trained end-to-end, unlike the use of frozen features for the critic. For our experiments, we sample $N = 16$ actions for computing the Best-of-N style policy learning objective in Equation 7: while the choice of N can differ from domain to domain, our runs find that $N = 16$ is a good choice for device control problems. We use LLaVa-1.5 (Liu et al., 2024a) for the backbone VLM for our Q- and V- functions.

5 Experimental Evaluation

The goal of our experiments is to evaluate Digi-Q in training effective device control agents. We aim to answer: **(1)** How does Digi-Q compare to state-of-the-art algorithms for user-scale device control tasks? **(2)** Can Digi-Q effectively learn from past interaction data? **(3)** Does Digi-Q offer a favorable computation-performance trade-off compared to end-to-end TD-learning? Additionally, we conduct ablations to assess the impact of components like representation fine-tuning and the Best-of-N re-ranking approach for policy extraction.

5.1 Main Performance Results

Comparisons. We compare Digi-Q with prior methods for building device control agents. First, we compare with several prompting-based methods that extend off-the-shelf proprietary VLMs such as GPT-4V (OpenAI, 2024b) and Gemini 1.5 Pro (GeminiTeam, 2024) with the Set-of-Marks (Yang et al., 2023) and a chain-of-thought mechanism for producing actions. We also compare with existing VLMs trained via imitation learning for device control: CogAgent (Hong et al., 2023), a 18B model and AutoUI-1B (Zhang & Zhang, 2023), that our policy is based on. Finally, we compare to the current state-of-the-art approach, Digi-RL, which uses advantage-filtered behavioral cloning for training the agent instead of off-policy value-based RL. We evaluate our results on Android-in-the-Wild (AitW) dataset with offline dataset containing 1296 trajectories from pre-trained AutoUI checkpoint, following Bai et al. (2024). More details on the offline dataset can be found in Appendix B.1.

Results. Our main results are presented in Table 1. We find that Digi-Q outperforms all prompting-based methods substantially (45.9% absolute improvement on average compared to the best prompting-based approach AppAgent with GPT-4V) and improves over the previous state-of-the-art, Digi-RL by 9.9% relatively averaged on General and Web Shopping subset, and 19.2% relatively over Filtered-BC, a simple but strong baseline. By visualizing the agent's rollouts on test examples, as we will show in Section 5.4, we find that training with offline value-based RL enhances the capability of RL to perform dynamic programming sub-optimal data to learn a better policy in the environment.

5.2 Compute Efficiency Comparison

A common concern with deploying TD-learning methods to train large-scale foundation models is their compute inefficiency (Abdulhai et al., 2023; Chebotar et al., 2023). Therefore, we attempted to understand the compute-performance tradeoffs associated with Digi-Q by comparing it against end-to-end TD-learning on VLMs without using any representation fine-tuning or frozen pre-trained representations. We plot the performance-compute tradeoff curve for Digi-Q on the web-shopping subset of the AiTW dataset in Figure 6. We found it a bit hard to fine-tune an entire VLM with TD-learning, which required iteration on hyperparameters such as learning rate and soft update rates for target networks. Due to the compute-intensive nature, we use a 3B VLM (PaLiGemma (Beyer et al., 2024)) for these runs instead of our 7B VLM (Liu et al., 2024b), and evaluate the performance of the critic as measured by the correlation between advantage predictions and ground-truth notion of human judgement on a held-out set of trajectories. In particular, we find that end-to-end TD-learning exhibits a much worse performance-compute frontier, to the extent that beyond a point more training FLOPS hurts performance. We conjecture that this behavior is likely a result of well-known pathologies of training large models with TD learning (Kumar et al., 2022), though we leave it for future work to fully understand these pathologies in our context. In contrast, while Digi-Q invests an initial amount of computation for representation fine-tuning, its accuracy quickly rises up and results in much better frontiers, with no instability. The calculation of the FLOPS is in Appendix B.2.

5.3 Ablation Studies

Next, we will perform a series of controlled experiments to understand the reasons behind the efficacy of Digi-Q. In particular, we will attempt to understand **(1)** the effect of representation fine-tuning (Stage I) for seeding the VLM representations for subsequent Q-function training, **(2)** the behavior of Best-of-N reranking style policy extraction operator compared to AWR (used by DigiRL (Bai et al., 2024)) and standard REINFORCE-style policy gradients (Williams, 1992), and **(3)** the benefits of TD-learning over the more conventional approach of supervised regression to Monte-Carlo return. Experimental details of the ablation studies can be found in Appendix B.3.

Representation	Performance
Behavior Policy	25.0
Digi-Q (w/ MC return)	32.3 ± 2.5
Digi-Q w/ CLIP + BERT	14.9 ± 6.0
Digi-Q Off-the-shelf VLM	22.2 ± 0.5
Digi-Q (Ours)	**42.7** ± **2.5**

Table 2: **Efficacy of our representation fine-tuning procedure** on the Web-Shopping test set in AiTW.

(1) The effect of representation fine-tuning in Digi-Q on VLM representations. We first study the effect of fine-tuning the VLM representations by training them to accurately detect actions that led to a substantial change in the scene. To do so, we compare Digi-Q with alternate approaches that train Q-functions on top of two other natural choices of representations: **(a)** not using a generative VLM (i.e., Llava-1.5), but instead using frozen CLIP (Radford et al., 2021) and BERT (Devlin et al., 2019) representations, following the protocol in Bai et al. (2024); **(b)** using an off-the-shelf VLM (Chen et al., 2024). Observe that using an off-the-shelf VLM outperforms the CLIP+BERT representations used in Bai et al. (2024): this is perhaps expected because an off-the-shelf generative VLM is still prompted with an appropriate prompt asking it to pay attention to the action, where CLIP does not offer such flexibility. That said, its performance is still below the behavior policy. As we will also qualitatively show in Section 5.4, off-the-shelf VLMs also do not pay enough attention to action information, resulting in a Q-function that degenerates to a similar solution as the state-only value function. In contrast, the representation fine-tuning procedure employed by Digi-Q leads to about 20% absolute performance improvement.

(2) The effect of best-of-N reranking based policy extraction operator. Next, we aim to understand the impact of using best-of-N reranking for policy extraction. This operator differs from traditional policy extraction methods in several ways: **(i)** the use of multiple actions **(ii)** not using a "negative gradient" (Tajwar et al., 2024) as in REINFORCE (Williams, 1992). To understand the effect of the number of actions in **(i)**, we ablate Digi-Q over $N \in \{1, 4, 8, 16, 64\}$ in Figure 3 (*right*). Observe that Digi-Q improves monotonically as N increases, indicating a clear benefit of sampling more actions and reranking them against the Q-function during training. More discussions on the ablations of number of actions resampled can be found in Appendix B.4.

Next, we answer **(ii)** by comparing Digi-Q with REINFORCE and supervised regression (AWR). Our results in Table 3 show that standard policy gradient attains the worst performance, attaining final performance below the behavior policy (17.4% vs 25.0%). We hypothesize that this is a direct consequence of the negative gradient effect, which is known to destabilize training. While AWR (used by Bai et al. (2024)) does not suffer from this issue, it is also not able to improve substantially over

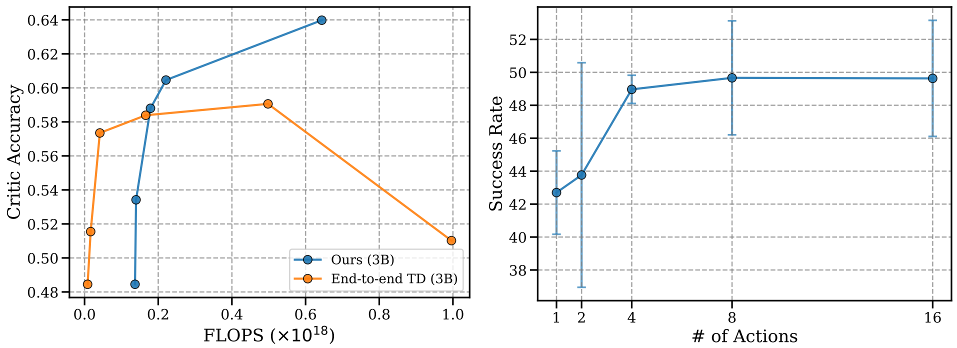


Figure 3: ***Left:*** **Offline critic evaluation accuracy as a function of compute** measured in terms of training FLOPS, compared across Digi-Q, end-to-end TD-learning on a VLM, and MC return. Observe that the critic accuracy is much better for our approach over end-to-end TD-learning as the amount of compute increases. ***Right:*** **Performance of Digi-Q when varying the number of actions N used for policy extraction.** Observe that the performance of Digi-Q improves when more actions are used for policy extraction, indicating the efficacy of our approach and the benefits of learning a Q-function.

the dataset (27.8% vs 25.0%). On the other hand, Digi-Q is able to make substantial improvements over the average success rates of rollouts in the dataset.

Next we attempt to understand how "aggressive" the updates made by different approaches are since one concern with AWR-style updates in prior work is the extent to which they are conservative. We wish to understand if our best-of-N reranking based policy extraction approach also admits conservative updates. To do so, we measured the KL-divergence between actions from the dataset and the fine-tuned policies produced by Digi-Q, AWR, and REINFORCE in Table 3. Note that Digi-Q incurs a larger KL-divergence value unlike AWR that incurs the smallest deviation and is most conservative. On the contrary, REINFORCE attains substantially larger divergence values and behaves unstably (see Appendix C.4 for some example rollouts).

Actor Objective	**Performance**	**KL v.s. Behavior Policy**
Behavior Policy	25.0	0
REINFORCE	17.4 ± 1.8	8.69
AWR	27.8 ± 0.5	1.52
Digi-Q (Ours)	**49.7** ± **3.5**	2.46

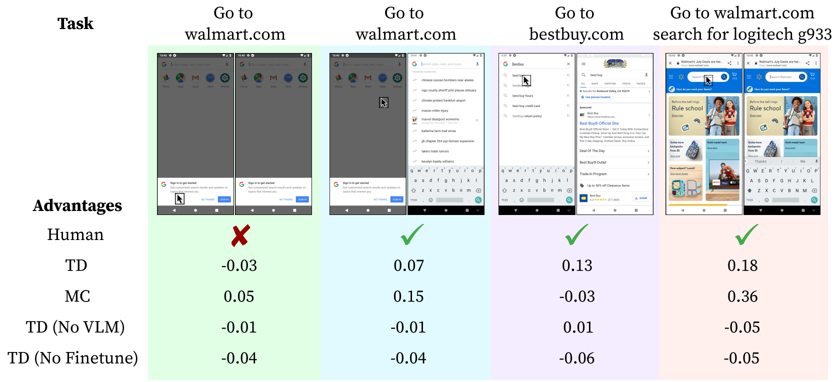
Table 3: **(1) Performance and (2) Token-level KL-divergence value between the learned policy and the dataset** when using different policy extraction methods on Web Shopping test set. We utilize the same critic for all the methods, and only train the policy differently.

(3) The effect of TD-learning as opposed to MC. To understand the importance of TD-learning for training the critic over Monte-Carlo (MC) regression that previous work is based on, we run an ablation of Digi-Q, which uses MC regression. Observe in Table 2, that this version underperforms Digi-Q by 10% (42.7% compared to 32.3%). As we show in Section 5.4, value functions from MC regression exhibit high variance, which inhibits them from producing good policies.

5.4 Qualitative Visualizations

Qualitative comparisons between different value function learning approaches. To qualitatively understand the quality of the Q-function learned, in Figure 4, we visualize advantage estimates $A(s, a) = Q_\theta(s, a) - V_\phi(s)$ computed from Q-functions produced by four methods: **(1)** Digi-Q (with representation fine-tuning and TD-learning), **(2)** Monte-Carlo regression, **(3)** Digi-Q but using CLIP+BERT representations from Bai et al. (2024); and **(4)** Digi-Q without representation fine-tuning. We contrast advantages against human judgements of whether the actions mark progress towards the desired task. Ideally, good actions should attain a positive advantage. We observe that advantage estimates from MC regression suffer from a high variance in advantage estimation because of the use of high-variance MC estimator. Moreover, we find that **(3)** and **(4)** converge to a degenerate $Q(s, a)$ that approximately matches a state-only value function, with no meaningful sensitivity to the action input. Thus, all these ablation variants fail to attain a good correlation with human judgement while only Digi-Q **(1)** is able to align well with human annotations.

Benefits of dynamic programming. A key advantage of value-based RL is its ability to perform dynamic programming: deriving optimal behaviors from suboptimal rollouts. As shown in **??**, Digi-Q learns optimal behaviors from suboptimal data. For example, trajectory (A) in the offline dataset



Task	Go to walmart.com	Go to walmart.com	Go to bestbuy.com	Go to walmart.com search for logitech g933
Advantages				
Human	✗	✓	✓	✓
TD	-0.03	0.07	0.13	0.18
MC	0.05	0.15	-0.03	0.36
TD (No VLM)	-0.01	-0.01	0.01	-0.05
TD (No Finetune)	-0.04	-0.04	-0.06	-0.05

Figure 4: **Qualitative examples showing the advantage estimations of several transitions of TD (ours), Monte-Carlo, and TD without VLM representation.** Advantage estimations using TD-learnt value functions top of VLM representation better align with human judgements compared to MC and TD without using VLM.

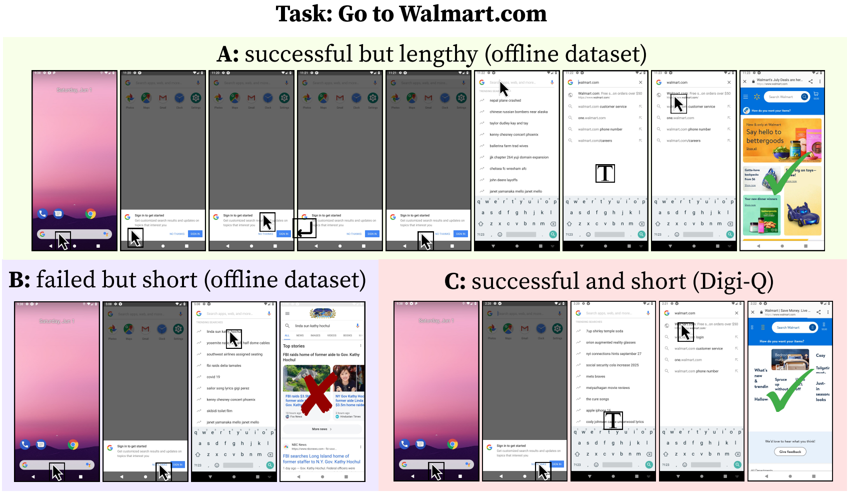

Task: Go to Walmart.com
A: successful but lengthy (offline dataset)
B: failed but short (offline dataset)
C: successful and short (Digi-Q)

Figure 5: **Trajectory examples showing benefits of Q-functions.** Our method can combine the best of a successful but lengthy (*A*) trajectory and a failed but short trajectory (*B*), to produce successful and short trajectories (*C*).

completes the task but includes redundant actions, while trajectory (B) is efficient but fails to complete the task. Digi-Q combines (A) and (B) to produce trajectory (C), which efficiently completes the task. This demonstrates Digi-Q's capability to learn an optimal policy from suboptimal data—something imitation alone cannot achieve.

6 Conclusion and Future Work

We presented Digi-Q, an effective value-based offline RL method tailored specifically for training real-world device-control agents at scale. At the core of our method is a representation fine-tuning procedure that induces actionable features from VLM useful for later TD-learning and a best-of-N policy extraction method that makes the best use of the learnt Q function from TD learning. Because of practical constraints such as compute budgets, our experiments can only focus on one domain of GUI agent tasks on Android tasks and three seeds of trials for each setting. While we have made our best efforts to make comparisons fair, such as repeated experiments and re-running baselines from prior works, the exact number may still be different from prior works because of the non-stationary nature of real-world websites and Android softwares. While we primarily focus on GUI agent tasks on Anroid devices, our methodology is general, compute efficient, and leads to substantial improvement in performance. We believe that these ideas and approach should transfer to new tasks as well and applying Digi-Q to new domains in an interesting avenue for future work. That said, extending Digi-Q to online RL setting will require a more sophisticated system design to speed up the experiment iterations. Nonetheless, ideas from Kalashnikov et al. (2018) could provide a good starting point to build systems for TD-learning during real-world interaction.

Reproducibility Statement

To facilitate reproducibility of our work, we will open-source the model checkpoints and training infrastructure. We have included a discussion of our choices of hyperparameters in all experiments in Appendix F and the prompts that we used for the autonomous evaluator in Appendix E. We will also release the data that we use to train our model and the environment configuration (e.g. emulator device specifications).

Ethics Statement

We develop methods that enable device control agent to operate on a fully-functioning Android device in an open-ended way. It is possible that irresponsible deployment of such device control agents can result in privacy leaks and vulnerability to malicious attacks (e.g. when the agent clicks on some malicious website links). While the development of a device control agent can significantly benefit productivity and increase accessibility, it is important that proper precautionary mechanisms should be in place to prevent such risks.

References

Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models, 2023. URL https://arxiv.org/abs/2311.18232.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

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, 2024. URL https://arxiv.org/abs/2406.11896.

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. *arXiv preprint arXiv:2302.02948*, 2023.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. Paligemma: A versatile 3b vlm for transfer, 2024. URL https://arxiv.org/abs/2407.07726.

Yevgen Chebotar, Quan Vuong, Alex Irpan, Karol Hausman, Fei Xia, Yao Lu, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, Keerthana Gopalakrishnan, Julian Ibarz, Ofir Nachum, Sumedh Sontakke, Grecia Salazar, Huong T Tran, Jodilyn Peralta, Clayton Tan, Deeksha Manjunath, Jaspiar Singht, Brianna Zitkovich, Tomas Jackson, Kanishka Rao, Chelsea Finn, and Sergey Levine. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions, 2023. URL https://arxiv.org/abs/2309.10150.

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

William Chen, Oier Mees, Aviral Kumar, and Sergey Levine. Vision-language models provide promptable representations for reinforcement learning, 2024. URL https://arxiv.org/abs/2402.02651.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023. URL https://arxiv.org/abs/2306.06070.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477.

GeminiTeam. Gemini: A family of highly capable multimodal models, 2024. URL https://arxiv.org/abs/2312.11805.

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding html with large language models, 2023. URL https://arxiv.org/abs/2210.03945.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019. URL https://arxiv.org/abs/1812.05905.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024. URL https://arxiv.org/abs/2401.13919.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

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. URL https://arxiv.org/abs/2312.08914.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation, 2018. URL https://arxiv.org/abs/1806.10293.

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, 2024a. URL https://arxiv.org/abs/2401.13649.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents, 2024b. URL https://arxiv.org/abs/2407.01476.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning, 2020. URL https://arxiv.org/abs/2006.04779.

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. Dr3: Value-based deep reinforcement learning requires explicit regularization. *arXiv preprint arXiv:2112.04716*, 2021.

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2024a. URL https://arxiv.org/abs/2310.03744.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024b. URL https://llava-vl.github.io/blog/2024-01-30-llava-next/.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2023. URL https://arxiv.org/abs/2308.03688.

Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, Jiadai Sun, Xinyue Yang, Yu Yang, Zehan Qi, Shuntian Yao, Xueqiao Sun, Siyi Cheng, Qinkai Zheng, Hao Yu, Hanchen Zhang, Wenyi Hong, Ming Ding, Lihang Pan, Xiaotao Gu, Aohan Zeng, Zhengxiao Du, Chan Hee Song, Yu Su, Yuxiao Dong, and Jie Tang. Visualagentbench: Towards large multimodal models as visual foundation agents, 2024c. URL https://arxiv.org/abs/2408.06327.

Xinbei Ma, Yiting Wang, Yao Yao, Tongxin Yuan, Aston Zhang, Zhuosheng Zhang, and Hai Zhao. Caution for the environment: Multimodal agents are susceptible to environmental distractions, 2024. URL https://arxiv.org/abs/2408.02544.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

OpenAI. Gpt-4 technical report, 2024a. URL https://arxiv.org/abs/2303.08774.

OpenAI. Gpt-4v(ision) technical work and authors, 2024b. URL https://openai.com/contributions/gpt-4v/.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents, 2024. URL https://arxiv.org/abs/2404.06474.

Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline rl?, 2024. URL https://arxiv.org/abs/2406.09329.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

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, 2024. URL https://arxiv.org/abs/2408.07199.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control, 2023. URL https://arxiv.org/abs/2307.10088.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2024. URL https://arxiv.org/abs/2405.14573.

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

Charlie Snell, Ilya Kostrikov, Yi Su, Mengjiao Yang, and Sergey Levine. Offline rl for natural language generation with implicit language q learning, 2023. URL https://arxiv.org/abs/2206.11871.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents, 2024. URL https://arxiv.org/abs/2403.02502.

Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback, 2015. URL https://arxiv.org/abs/1502.02362.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference fine-tuning of llms should leverage suboptimal, on-policy data, 2024. URL https://arxiv.org/abs/2404.14367.

Siddharth Verma, Justin Fu, Mengjiao Yang, and Sergey Levine. Chai: A chatbot ai for task-oriented dialogue with offline reinforcement learning, 2022. URL https://arxiv.org/abs/2204.08426.

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

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. URL https://arxiv.org/abs/2311.07562.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023. URL https://arxiv.org/abs/2310.11441.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023. URL https://arxiv.org/abs/2207.01206.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms, 2023. URL https://arxiv.org/abs/2310.12823.

Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and Sergey Levine. Fine-tuning large vision-language models as decision-making agents via reinforcement learning, 2024. URL https://arxiv.org/abs/2405.10292.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users, 2023. URL https://arxiv.org/abs/2312.13771.

Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*, 2023.

Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents, 2024. URL https://arxiv.org/abs/2309.11436.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded, 2024. URL https://arxiv.org/abs/2401.01614.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024a. URL https://arxiv.org/abs/2307.13854.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl, 2024b. URL https://arxiv.org/abs/2402.19446.

Appendices

A Details on the Algorithm

For completeness, we include a detailed pseudo-code of Digi-Q in Algorithm 1. After initializing the parameters, we perform the representation fine-tuning procedure on top of VLM to obtain actionable features for later TD-learning. Then the VLM parameters will be kept frozen and we train the Q- and V- functions using TD-learning on top of frozen VLM representations. After both value functions are trained, we perform gradient updates on the actor with best-of-N policy extraction.

Algorithm 1 Digi-Q: Practical Framework

```
Initialize parameters $\phi, \psi_{\text{MLP}}, \bar{\psi}_{\text{MLP}}, \theta_{\text{MLP}}, \bar{\theta}_{\text{MLP}}$.
Initialize replay buffer $\mathcal{D}$ (from an offline dataset).
for each VLM iteration do
    $\theta_{\text{VLM}} \leftarrow \nabla J_{\mathcal{P}}(\theta_{\text{VLM}})$                              ▷ Equation 4
end for
for each critic step do
    ## Update high-level Q and V functions by target function bootstrapping.
    $\theta_{\text{MLP}} \leftarrow \theta_{\text{MLP}} - \nabla J_{\theta_{\text{MLP}}}(Q)$                ▷ Equation 5
    $\psi_{\text{MLP}} \leftarrow \psi_{\text{MLP}} - \nabla J_{\psi_{\text{MLP}}}(V)$                  ▷ Equation 6
    ## Update target Q and V functions.
    $\bar{\theta}_{\text{MLP}} \leftarrow (1-\tau)\bar{\theta}_{\text{MLP}} + \tau\theta_{\text{MLP}}$
    $\bar{\psi}_{\text{MLP}} \leftarrow (1-\tau)\bar{\psi}_{\text{MLP}} + \tau\psi_{\text{MLP}}$
end for
## Update low-level actor with high-level critic.
for each actor step do
    $\phi \leftarrow \phi - \nabla J_{\phi}(\pi)$                                       ▷ Equation 7
end for
```

B Experimental Details

B.1 Offline Dataset Construction

We use the pre-trained AutoUI checkpoint to collect offline trajectories. Specifically, to collect each trajectory, starting from the home screen, the agent generates an action, and then the environment takes the action and transitions to the next state. It iterates until a maximum number of steps have been reached or the autonomous evaluator has decided to be a success. We collect 1296 trajectories this way for both AitW Webshop and AitW General subsets. The horizon H of the Webshop subset is set to 20, and the horizon of the General subset is set to 10, which aligns with (Bai et al., 2024). Each trajectory is composed of state-action-reward-next-state pairs (s, a, r, s'), which is also referred to as “transitions".

The actions in the offline dataset collected this way are sampled from the pre-trained AutoUI checkpoint. When training the actor offline, as we use the Best-of-N loss, we want to sample more than one action. To facilitate fast iterations, we pre-collect $K - 1$ actions for each state and add them to the offline dataset. In practice, we find $K = 64$ sufficient for $N \leq 16$, i.e. it suffices to give enough variety compared to sampling the actions when training the actor model. Note that in this case, the original action will always appear in the offline dataset.

B.2 FLOPS Calculation and Critic Accuracy

FLOPS Calculation. The 3B VLM takes 45.6×10^{12} FLOPS for *each sample* for forward plus backward process. As the end-to-end TD learning contains one VLM as part of the Q function and one VLM as the target Q function (which only do forward pass), one sample takes 68.4×10^{12} FLOPS (according to Hoffmann et al. (2022), the FLOPS incurred by the forward prrcess is approximately half of the backward process). Thus, as the longest run takes 15k samples, the last point of the end-to-end run in Figure 3 (*Left*) takes around 1×10^{18} FLOPS. Also, the first logged point takes 128 samples, so the starting point should have 8.3×10^{15} FLOPS.

On the other hand, in Digi-Q, we first finetune the 3B VLM, which incurs only one forward and backward process. Thus, finetuning the 3B VLM on 2000 samples takes 91.2×10^{15} FLOPS. After that, we infer the representations of these samples with the 3B VLM, which includes one forward pass. This sums up to 136.8×10^{15} FLOPs, which explains the starting point of the Digi-Q curve.

Then we only train the value head using the VLM representations. The size of the value head is 0.07B, incurring 1.1×10^{12} FLOPS for each sample. The longest run of Digi-Q takes 0.46M samples, thus incurring 506.9×10^{15} FLOPS (10×10^{17}).

Thus, the end-to-end TD learning should range from 0.0083×10^{15} to 1×10^{18} FLOPS, while Digi-Q should range from 0.137×10^{18} FLOPS to 0.644×10^{18} FLOPS, which is shown in Figure 3 (*Left*).

Critic Accuracy. We manually label 483 states with binary advantages, and normalize the advantages produced by the agents to have a mean of zero before thresholding and calculating its accuracy with human annotations.

B.3 Additional Method Details

Task set formulations. The two task sets (Webshop and General) in the AitW dataset have different horizons H (maximum number of steps allowed) in a trajectory to improve computational efficiency. Specifically, $H = 20$ for AitW Webshop and $H = 10$ for AitW General. Following tradition (Bai et al., 2024), we keep $A > 1/H$ (e.g. 0.05 for AitW Webshop) as a threshold for the actor model to learn the state-action pair.

Ablation on representation fine-tuning and TD learning as opposed to MC. In the ablation study on representation fine-tuning, for all configurations, we train the actor model with best-of-N loss where $N = 1$ to keep computation efficient. This is also the case for the ablation on the TD learning as opposed to MC ablations.

Ablation on actor loss. For the ablation study on the actor loss, we keep the same trained Q function, while we ablate only on the loss used to train the actor model. We use 30 actor epochs for the best-of-N loss and AWR loss, and 120 epochs for the REINFORCE loss as the magnitide of the raw advantage is very small. We use $N = 16$ for the best-of-N loss, while REINFORCE and AWR both uses the original action in the offlin dataset.

Value function. In practice, we find the V function significantly easier to train, and it suffices to only use the representations of the state from the vision encoder of the VLM (CLIP) to train the V functions. This simplification significantly saves time and space required, and aligns with previous work (Bai et al., 2024).

B.4 Ablation on Number of Actions Sampled

As mentioned in Appendix B.1, from an engineering aspect, collecting actions each time we sample from the offline dataset $\mathcal{D}$ is not efficient. Thus, in practice, we pre-collect $K = 64$ actions for each state, and store them in the offline dataset. As $n \in \{1, 2, 4, 8, 16\}$ is much smaller than 64, this strategy serves as a good approximation and results in good performance. We call this strategy *approximate random action sampling*.

Determinisitc action sampling, on the other hand, always sample the first n actions from the pre-collected action set. In this case, if a state is sampled multiple times in one iteration, the action set will always be the same, and thus the critic will always produce the same action for the actor to learn. We observe that the performance of determinisitc action sampling is significantly worse than approximate random action sampling, as shown in Figure 6.

Note that the $n = 1$ case for both approximate random and determinisitc action sampling uses the original action (that causes the transition) instead of sampling it from the action set. Also, for all cases where $n > 1$, we always include the original action into the sampled actions.

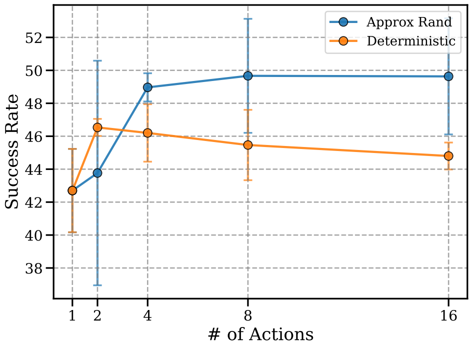
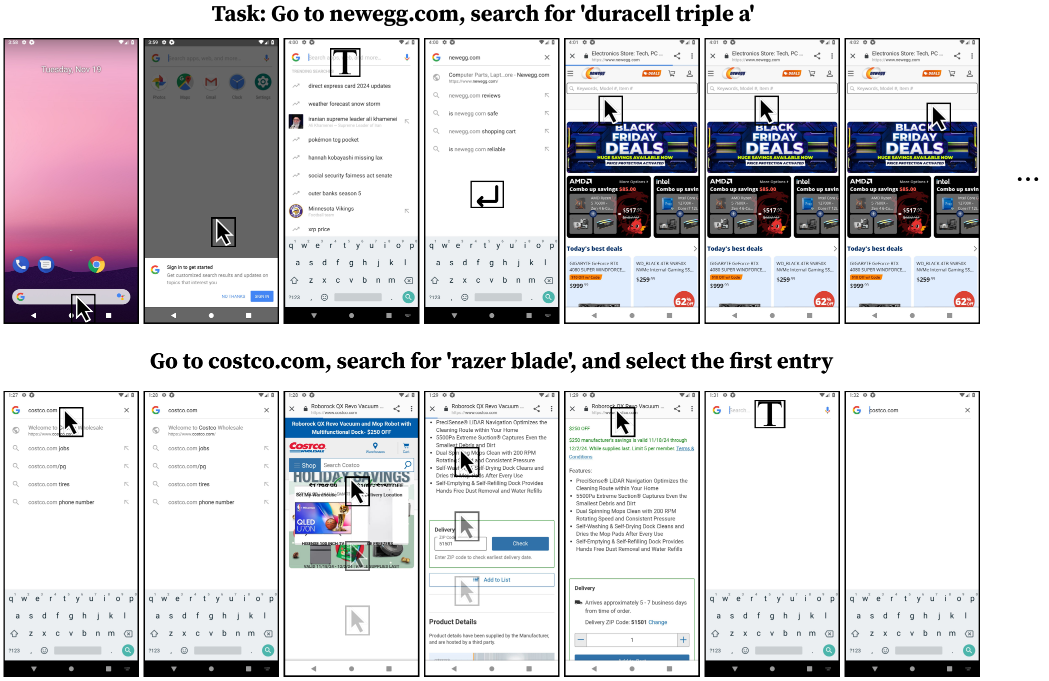


Figure 6: **Ablation study results on performance w.r.t. number of actions sampled for each step.** Errors are displayed as translucent bars along the curve.

C More Qualitative Examples

C.1 Failure Case Study

We observe some failure cases of the agent trained using Digi-Q, as shown in Figure 7. We observe that the agent successfully arrives at the shopping homepage, but fails to click the search bar after several attempts. This is probably because there is a distribution shift from the pre-training data and the non-stationary environment.

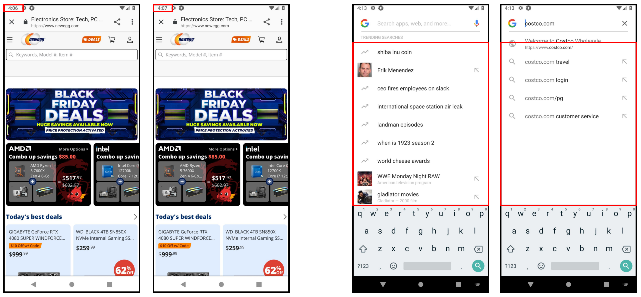

Task: Go to newegg.com, search for 'duracell triple a'

Go to costco.com, search for 'razer blade', and select the first entry

Figure 7: **Example failed trajectories.**

C.2 Example Image Differences for SFT Threshold

We illustrate two different transitions with their image difference. The first transition only has a minor difference on the top left of the screen (clock time), and has a difference of 1.6. The second transition has a major difference on the screen (search suggestions), and has a difference of 232.8.



difference=1.6 difference=232.8

Figure 8: **Example transitions and their image differences.**

C.3 Value Estimations on Sampled Actions

In this section, we show how the Q function in Digi-Q judges the values of the state-action pairs that are sampled from the actor model. In Digi-Q, we first sample N actions, then use the Q functions to estimate the action values of *each of* the actions sampled. Some real examples are shown in Figure 9.

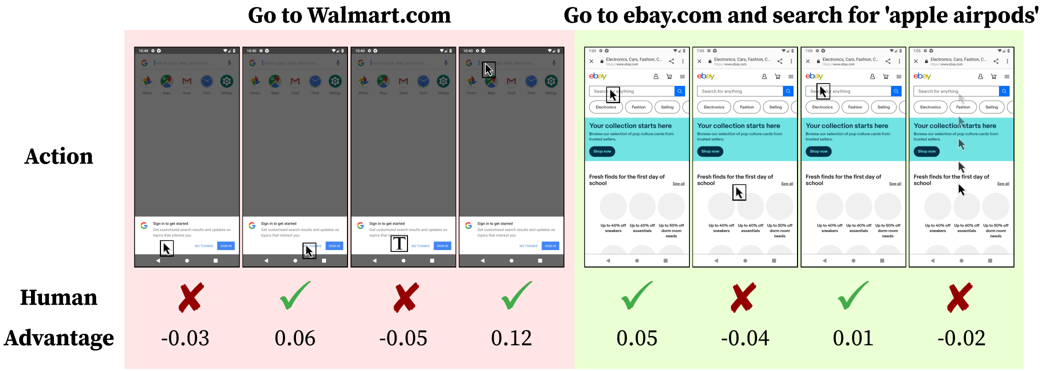


Figure 9: **Qualitative transitions showing the estimated values from our method on the sampled actions.**

We show two groups of state-action examples, each including $N = 4$ sampled actions from the actor model, for each state. Their corresponding advantages are displayed. We clearly observe that the advantages predicted by Digi-Q highly aligns with human prediction. For example, in the state of having a box preventing the agent to go to walmart.com (*Left*), the action of clicking on "NO THANKS" (action 1) should be a good action, and clicking outside the box (action 4) will also escape this message. In contrast, clicking on the box (action 1) and typing something (action 3) does not help. We find that the advantages estimated by Digi-Q highly aligns with the human annotations.

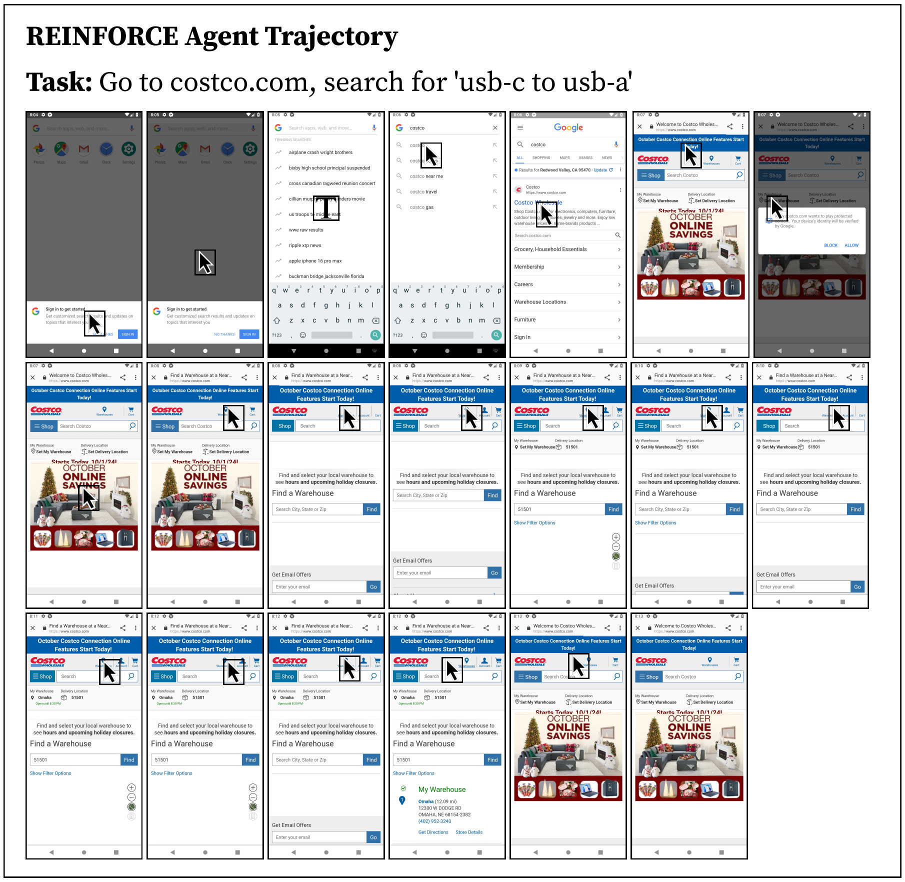
C.4 Example Trajectory of REINFORCE

We show a typical trajectory produced by the agent trained with REINFORCE in Figure 10. We observe that the agent frequently diverges from the target and is too "stubborn" to recover from errors.

In this task towards searching for an item on costco.com, the agent has successfully arrived at costco.com, but (1) it takes some bad actions and (2) cannot recover. Specifically, after the agent clicks the warehouse button, it keeps clicking on the same button for 10 times until it clicks on somewhere else.

D Compute Requirements

We show practical statistics below on how much compute it takes to train AutoUI with Digi-Q on the AitW task set, which are counted on experiments done on a machine with 8 A100 GPUs. The SFT process is standard VLM fine-tuning, which takes 20 minutes for fine-tuning a LLaVA-1.5-7b model. Getting the representations on the offline dataset takes 3 hours after vLLM acceleration. Then the


REINFORCE Agent Trajectory

Task: Go to costco.com, search for 'usb-c to usb-a'

Figure 10: **Example trajectory of the agent trained with REINFORCE.** The order is displayed from left to right, from top to bottom.

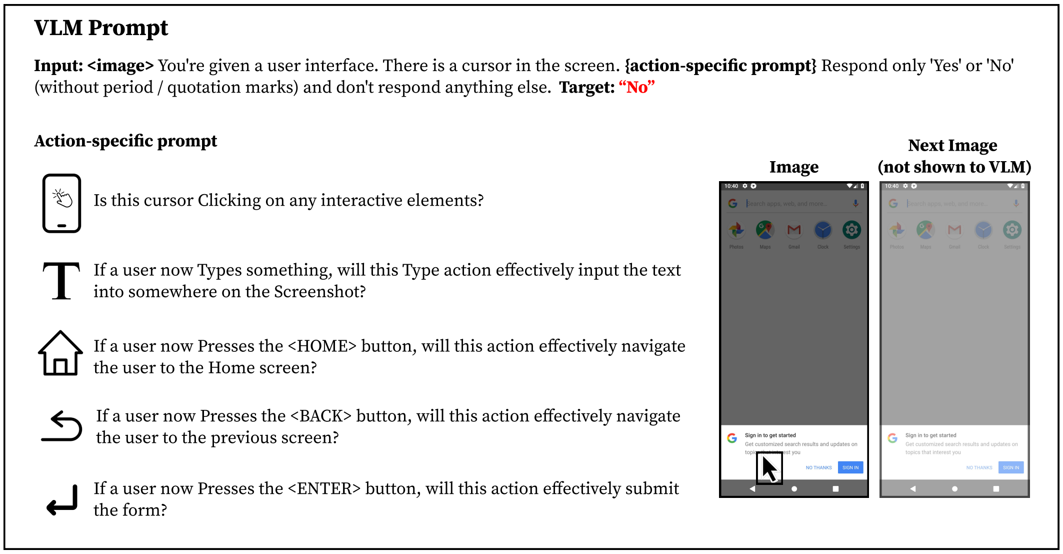
critic learning takes 20 minutes and actor learning takes 30 minutes. The whole pipeline is very well optimized (at least 4x faster than non-optimization) and will be made public after the release of the paper.

E VLM Prompts

The prompt we use for fine-tuning and inferring the VLM is shown in Figure 11. The prompt template is designed to be action-type-specific, in order to facilitate the VLM to better differentiate different types of actions, which promotes fine-grained representations within the same action type. The input to the VLM is constructed by the image and the text prompt. Note that the VLM only sees the current image (overlayed with a cursor if the action is to click), and the next image is only used to calculate whether the target should be “yes" or “no". The target is a single token to promote computational efficiency. In practice, we find that a long target sequence introduces challenges for the VLM to learn the representations.

F Hyperparameters

Hyperparameters for Digi-Q 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 4. Results for all other methods (Filtered BC and DigiRL) are kept the same as discussed in the original paper (Bai et al., 2024).



VLM Prompt

Input: <image> You're given a user interface. There is a cursor in the screen. **{action-specific prompt}** Respond only 'Yes' or 'No' (without period / quotation marks) and don't respond anything else. **Target: "No"**

Action-specific prompt

Is this cursor Clicking on any interactive elements?

If a user now Types something, will this Type action effectively input the text into somewhere on the Screenshot?

If a user now Presses the <HOME> button, will this action effectively navigate the user to the Home screen?

If a user now Presses the <BACK> button, will this action effectively navigate the user to the previous screen?

If a user now Presses the <ENTER> button, will this action effectively submit the form?

Figure 11: **Prompt template we use to fine-tune and infer the VLM.**

Table 4: Hyperparameters for All Experiments

Method	Hyperparameter	Value
Digi-Q	actor lr	1e-4
	value function lr	1e-4
	batch size	128
	maximum gradient norm	0.01
	actor updates per iteration	30
	value function updates per iteration	20
	number of iterations for offline actor updates	15, 20, **30**, 45, 60
	number of iterations for offline value function updates	30, 45, 60, 90, **120**, 150, 180, 300
VLM	model checkpoint	liuhaotian/llava-v1.5-7b
	image aspect ratio	**pad**, no
	vision encoder	openai/clip-vit-large-patch14-336
	number of training epochs	3,**5**,8,10
	per device train batch size	8, **16**, 32
	per device eval batch size	4

Table 5: Hyperparameters for Digi-Q on both General and Web Shopping subset of AitW. If multiple values are displayed, the **bolded** value represents the selected value after hyperparamemter sweeping.