VLM Q-LEARNING: ALIGNING VISION-LANGUAGE MODELS FOR INTERACTIVE DECISION-MAKING

Jake Grigsby[1]*, **Yuke Zhu**[1], **Michael Ryoo**[2], **Juan Carlos Niebles**[2]
[1]The University of Texas at Austin [2]Salesforce AI Research

ABSTRACT

Recent research looks to harness the general knowledge and reasoning of large language models (LLMs) into agents that accomplish user-specified goals in interactive environments. Vision-language models (VLMs) extend LLMs to multi-modal data and provide agents with the visual reasoning necessary for new applications in areas such as computer automation. However, agent tasks emphasize skills where accessible open-weight VLMs lag behind their LLM equivalents. For example, VLMs are less capable of following an environment's strict output syntax requirements and are more focused on open-ended question answering. Overcoming these limitations requires supervised fine-tuning (SFT) on task-specific expert demonstrations. Our work approaches these challenges from an offline-to-online reinforcement learning (RL) perspective. RL lets us fine-tune VLMs to agent tasks while learning from the unsuccessful decisions of our own model or more capable (larger) models. We explore an off-policy RL solution that retains the stability and simplicity of the widely used SFT workflow while allowing our agent to self-improve and learn from low-quality datasets. We demonstrate this technique with two open-weight VLMs across three multi-modal agent domains.

1 INTRODUCTION

Diverse training datasets give large language models (LLMs) the ability to generate knowledgeable text across a wide range of topics. LLM pre-training produces a model that outputs the most likely continuation of its input text according to a distribution of training examples sourced from across the web. However, there are many applications where we would like to direct our model to accomplish a specific objective. In these tasks, our model acts as an agent in an interactive environment and makes a sequence of decisions to maximize an evaluation metric. Because the model was not trained to optimize this objective, the best decision-making strategy may be an unlikely output. Ongoing research looks to fix this alignment problem and repurpose generative models for decision-making.

LLM-agent techniques are largely enabled by the post-training process, where the model is fine-tuned on curated datasets of chat or assistant-like behavior. A natural first step is to continue training with supervised fine-tuning (SFT) on task-specific datasets. However, SFT cannot outperform the best decision-making strategy in its training data and relies on collecting successful demonstrations. LLMs' ability to adapt to new tasks based on examples in their input (Brown et al., 2020) allows this realignment to be done without further training: agent behavior can be generated by prompting an LLM with demonstrations. Additional prompting techniques improve long-term planning by eliciting self-reflection and reasoning (Yao et al., 2022b; Wei et al., 2022b).

Vision-language models (VLMs) extend LLMs to multi-modal inputs. VLMs are trained by using smaller datasets of interleaved text and image data to mix visual features into an existing LLM (Liu et al., 2024b;a). The high-level problem of realigning VLMs with decision-making objectives is similar to that of LLM agents. However, VLM research is still in its early stages, and current models lag behind their LLM equivalents in two critical areas: 1) agent action syntax and 2) long-context prompting. Agent tasks often do not involve freeform dialogue and instead require outputs with a specific syntax that the environment can interpret. The ability to plan and act in this more restricted output space is closely related to the tool-use (or function-calling) capabilities of LLMs (Qu et al., 2024). Perhaps a more long-term issue is that environment interactions

*Work done during an internship at Salesforce AI Research.

can stretch over many model outputs, making it impractical to include complete in-context examples and align the VLM via prompting. SFT can realign a base model with agent tasks, but it is only effective when we can demonstrate a strategy that meets our performance expectations.

Our work approaches the challenges of fine-tuning VLMs for agent tasks from a reinforcement learning (RL) perspective (Figure 1). RL solves the agent alignment issue by explicitly training our model to maximize success in its environment. Decision-making is often formalized as an RL problem, but RL methods have a reputation for adding complexity; simpler prompting and SFT techniques are effective for LLM-agent tasks and are more widely used in recent work. However, we note that a category of offline RL techniques (Levine et al., 2020) that filter demonstration datasets according to a learned value model (Wang et al., 2020) have important advantages in the emerging area of VLM agent training. Like most offline RL algorithms, these filtered supervised fine-tuning (FSFT) methods let our agent outperform its demonstrations. Unlike other offline techniques, FSFT is effectively identical to SFT when the dataset is of high quality and lets us seamlessly collect new data when our agent needs further improvement. This lets FSFT serve as a low-downside, high-upside replacement for SFT in VLM agent domains where SFT is widely used. Our work explores the challenges of applying VLMs to agent tasks and demonstrates the strengths of off-policy RL applied to two open-weight models across three domains.

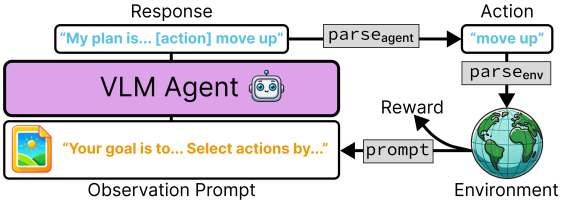

Figure 1: **VLM-Environment Interaction.** We format text and image data from the environment into an input prompt for the VLM. The models's text reply is parsed into a concrete action executed by the environment to produce a reward and new inputs for the VLM.

2 Background

Vision-Language Models. VLMs extend the capabilities of LLMs to visual data by taking text and images as input and producing text outputs. They process multi-modal inputs by projecting the features of a vision encoder into the embedding space of an LLM — allowing the LLM to attend to the features of both modalities. While there have been many VLM models released in recent months, our work will focus on publicly available (open-weight) models that are small enough to be fine-tuned with accessible GPU resources. Table 1 introduces three models used in our experiments: PaliGemma (Beyer et al., 2024), xGen-MM (Xue et al., 2024), and MoonDream2 (Korrapati, 2024).

	MoonDream2	PaliGemma	xGen-MM
Parameters	1.9B	2.9B	4.6B
Feature Dim.	2048	2048	3072
Vocabulary	51,200	257,216	32,015

Table 1: **VLMs in Our Experiments.**

Foundation Models as Agents. We focus on agent applications where a model is evaluated based on its ability to make decisions that achieve a specific outcome. Examples include: controlling an operating system with text commands (Wu et al., 2024), writing code to answer data analysis questions (Hu et al., 2024), navigating graphical interfaces, (Gao et al., 2023; Tao et al., 2023; Xie et al., 2023), ordering items from a web store (Yao et al., 2022a; Tao et al., 2023), carrying out administrative tasks in business software (Drouin et al., 2024), making PowerPoint slides based on a specific topic (Guo et al., 2023), and playing games (Chen et al., 2024a), among many others. These tasks are characterized by *multi-turn interactions*, where the model iteratively reacts to changes in the environment until it is successful. Success is measured by explicit task-specific objectives and not by the likelihood of the model's response or subjective human preferences. This creates a fundamental mismatch between the objectives the model was trained to maximize and the objective we are evaluating.

We can realign foundation models with agent tasks by biasing their outputs with careful prompting. Agent tasks require high-level planning over a sequence of outcomes. Prompting techniques for agent applications combine in-context learning (Brown et al., 2020) with step-by-step reasoning and self-reflection over previous outcomes (Wei et al., 2022b; Yao et al., 2022b; Yang et al., 2023; Zheng et al., 2024). Prompting is particularly effective when working with large proprietary models that support sequence lengths long enough to let these methods grow arbitrarily complicated by making multiple API calls to correct mistakes, retrieve relevant information, and plan for the future (Topsakal & Akinci, 2023; Xiao et al., 2023; Lutz et al., 2024; Sridhar et al., 2023).

Tool Use and Function Calling. In order for an external environment like a browser or game to carry out an agent's request, it must be able to interpret the model's intention from open-ended text. Therefore, our model needs to generate text according to a task-specific syntax that is described as part of its input prompt (Hsieh et al., 2023). Action syntax creates a barrier to entry where models fail because they cannot produce valid outputs despite responses that suggest they have some understanding of the task. This problem is closely related to the "tool-use" (or function-calling) capabilities of LLMs (Ling et al., 2023; Xu et al., 2023). Tool-use refers to the ability of an LLM to call external APIs and is an important feature of chat assistants that retrieve information on current events, perform calculations, or prompt specialized image generation models. Tool-use accuracy in accessible open-weight LLMs once lagged behind proprietary models but has been improved by fine-tuning on specialized datasets with standardized syntax (Patil et al., 2023; Qin et al., 2023).

Fine-Tuning Agents. The syntax of agent environments is less standardized than tool-use, and task inputs may be very different from those seen during training. When prompting alone is not enough to create a successful agent, performance can be improved by fine-tuning on task-specific demonstrations (Zeng et al., 2023; Hong et al., 2024; Feng et al., 2023; Song et al., 2024; Zhang et al., 2024a; Yin et al., 2023). Continuing training can be expensive, but we can safely assume that the resulting agent will match the performance of the demonstrator(s) when given sufficient training data. However, we cannot expect our agent to learn a better strategy than appeared in the fine-tuning demonstrations because it is trained to replicate the decisions in the dataset.

Off-Policy RL. RL formalizes the process of learning optimal decision-making strategies in interactive environments. The environment begins in a state s_0 that is represented to the agent by an observation o_0. The agent's policy π takes this observation as input and samples an action $a_0 \sim \pi(o_0)$ from the space of possible actions $\mathcal{A}$, leading to a new state s_1 and reward r_1 determined by the environment's transition function ($T(s_0, a_0) = s_1$) and reward function ($R(s_0, a_0) = r_1$), respectively. This process repeats after the agent receives its next observation o_1. RL uses data to discover a policy that maximizes the agent's return — or cumulative reward from the current state: $V^\pi(s_t) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} r_i\right]$, where $\gamma \in [0, 1)$ is a discount factor.

Off-policy RL methods can learn from any experience in the environment — regardless of whether the actions of the current policy generated that experience. An example is Q-Learning (Watkins & Dayan, 1992; Mnih et al., 2015), which learns a function Q to estimate the return that will follow a particular action a: $Q^\pi(s_t, a_t) = R(s_t, a_t) + V^\pi(s_{t+1})$. If we can learn an accurate estimate of Q, we can improve the policy by selecting actions that maximize Q. Actor-critic methods train a critic model to approximate Q and a second (often entirely separate) actor model to maximize the critic's predictions (Lillicrap, 2015). *Offline* RL agents learn a policy from a fixed dataset (Levine et al., 2020). It is common to view offline RL as an extreme case of off-policy RL where we assume that our dataset was generated by a mixture of unknown policies and that we cannot collect additional data. However, vanilla Q-Learning methods struggle to handle the fully offline setting because they repeatedly maximize their current Q-values but cannot collect new experiences that would reveal their predictions are overestimated (Kumar et al., 2019). Successful modifications avoid directly maximizing Q (Kostrikov et al., 2021) or regularize estimates to be more conservative (Kumar et al., 2020; Agarwal et al., 2020). We derive our method from a family of filtered (or weighted) behavior cloning algorithms where a value estimate is used to mask (or downweight) sub-optimal demonstrations (Nair et al., 2020; Peng et al., 2019; Nair et al., 2018; Chen et al., 2020; Wang et al., 2018); the variant used here will be most similar to CRR (Wang et al., 2020). These methods are capable of outperforming the quality of their dataset by learning to prefer alternative actions when they lead to higher Q-values (Kumar et al., 2022; Ghugare et al., 2024). *Online* RL refers to the more standard setting where we are allowed to interact with the environment to collect additional data. *On-policy* updates improve a policy based on its own recent experience (Williams, 1992; Schulman, 2015), and are generally less sample efficient.

3 VLM Agents as RL Policies

We approach agent interactions from an RL perspective, where the VLM will become an RL policy mapping text and image observations to text actions. Text is represented by tokens from the VLM's vocabulary, $\mathcal{V}$. The external task the agent interacts with — such as the browser or game it is playing — can be thought of as the environment. The environment interprets text actions to produce the next observation and reward. Because VLMs are sequence-to-sequence models, there are two

ways to define the action space $\mathcal{A}$ (Zhou et al., 2024; Abdulhai et al., 2023). One option is to view each individual output token as a separate action ($\mathcal{A} = \mathcal{V}$). This **"token-based view"** most closely aligns with how sequence models are trained. During evaluation and data collection, it is simpler to take a "**turn-based view**" where each timestep is a full turn of dialogue and actions are complete replies with some maximum length r ($\mathcal{A} = \mathcal{V}^r$). On each turn t, the environment provides an observation of text (o^{text}) and image (o^{img}) data, and we query the VLM to generate a complete response a_t. The open-ended text response is parsed by the environment to produce the true action that it uses to update its state. Depending on the environment, the "true" action space $\mathcal{A}'$ may be text, code, or a set of discrete choices. We refer to this mapping from text actions to true actions as $\texttt{parse}_{\text{env}} : \mathcal{A} \to \mathcal{A}'$. We assume this behavior is outside the agent's control and that it is the agent's responsibility to produce text actions that $\texttt{parse}_{\text{env}}$ can interpret. Figure 1 summarizes the turn-based interaction between a VLM agent and its environment. Our goal is to use RL to update the VLM's parameters to maximize rewards.

4 Related Work

RL has many applications in foundation model fine-tuning and agent tasks. A summary of related work in Reinforcement Learning from Human Feedback and applications of foundation models without open-ended text-based actions is deferred to Appendix A.1.

Our VLM agent method shares technical motivations with work on fine-tuning LLMs for multi-turn dialogue. WebRL (Qi et al., 2024) applies a constrained on-policy RL update to prevent the fine-tuned policy from diverging from an initial task-specific SFT policy. When combined with a variety of other curriculum and dataset filtering techniques, the WebRL update allows open-source models to significantly outperform proprietary LLMs in WebArena (Zhou et al., 2023). ILQL (Snell et al., 2022) adjusts the outputs of a base LLM during inference based on the predictions of a second critic LLM trained by offline RL. ArCHer (Zhou et al., 2024) is an online RL update that highlights the "turn-based" vs. "token-based" action discrepancy (Section 3) and creates a hierarchical approach that splits learning across both timescales. In contrast, we will train RL solely from the token-based format because it allows for a simple substitution of SFT. ILQL and ArCHer are actor-critic updates; like most actor-critics, they create training signal for the main policy model we are fine-tuning by training other models to learn value predictions (Q^π and V^π) or a reference policy based on SFT. The cost of training multiple models is rarely an issue at the scale of standard RL research domains but is a significant concern in VLM applications where models contain billions of parameters. Our method will only involve a single VLM.

Recent work has applied online RL to VLMs in multi-turn vision tasks (Fereidouni & Siddique, 2024). The most closely related work to our own is RL4VLM (Zhai et al., 2024). RL4VLM fine-tunes the LLaVA VLM (Liu et al., 2024b) in a two-stage process. During the first stage, the base model is aligned to the syntax of the environment by SFT on demonstrations generated by chain-of-thought prompting (Wei et al., 2022b). The second stage improves the success rate of the SFT model with on-policy RL via PPO (Schulman et al., 2017). Our method will replace this two-stage pipeline with a single stage by using an RL update that makes the SFT stage redundant.

5 Method

Our method follows the popular framework of performing SFT on task-specific data that is generated by basic prompting. However, we allow the agent to improve over its dataset by filtering out tokens that we estimate will degrade the agent's performance. This filtering process is enabled by converting VLM agent fine-tuning to an off-policy actor-critic RL problem.

5.1 Task Setup and Prompting

The environment provides an observation of text (o^{text}) and image (o^{img}) data. A `prompt` method reformats information from the environment into a full input prompt, including a description of the agent's objective. We include documentation of the actions available to the agent alongside an example of the syntax expected by $\texttt{parse}_{\text{env}}$ (Section 3). We can generalize our method by introducing a second parser: $\texttt{parse}_{\text{agent}}$ (Figure 1). $\texttt{parse}_{\text{agent}}$ enables arbitrary output formats including chain-of-thought-style reasoning (Wei et al., 2022b; Yao et al., 2022b). We will allow

for reasoning (or "thought") outputs by setting $\texttt{parse}_{\text{agent}}$ to return only the text after the string "`[action]`". This choice is arbitrary and could be replaced by a JSON output (as in RL4VLM), for example. Our VLM agents' first challenge will be to output text actions a that are valid inputs to $\texttt{parse}_{\text{env}}(\texttt{parse}_{\text{agent}}(a))$. When the agent fails to do so, it will be given an error message and prompted to try again. VLMs that cannot break out of this loop create a sparse reward problem by failing to reach useful reward signals for training. We minimize this problem by introducing penalties for invalid syntax on top of the environment's native reward function (Appendix A.3.1). Appendix A.2.2 provides further discussion on action parsing and its connection to RL exploration.

5.2 RL Fine-Tuning

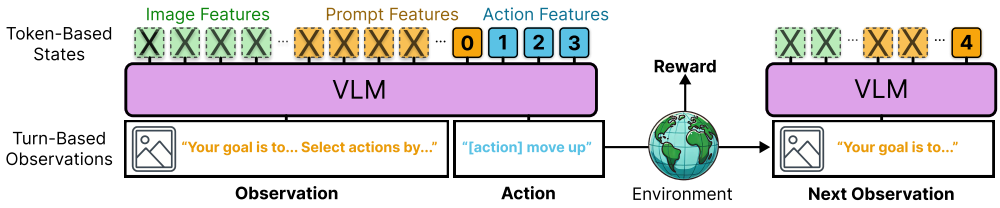

Figure 2: **Converting Turns → Tokens.** We treat each token of the agent's reply as a single action by passing consecutive turns of dialogue through the VLM and finding output representations corresponding to tokens where an action decision is made. Outputs corresponding to the prompt tokens are masked and ignored. The resulting sequence of RL input states is numbered $(0, 1, 2, 3, 4)$.

Let VLM_θ represent the VLM architecture with all its parameters θ excluding the token output layer(s) at the end of the LLM. We will use L_ϕ to refer to the feed-forward language head that maps VLM_θ's outputs to a distribution over the tokens in our vocabulary $\mathcal{V}$. Our dataset contains turn-based transitions: $((o_t^{\text{text}}, o_t^{\text{img}}), a_t, r_{t+1}, d_{t+1}, (o_{t+1}^{\text{text}}, o_{t+1}^{\text{img}}))$, where a_t is a complete reply and d_{t+1} is one if we've reached the final turn of the interaction and zero otherwise. While it is simpler to view agent-environment interaction from a turn-based perspective, our training process will convert data to a token-based action space. The end result will be a more standard RL problem where states are the output of the VLM, and our policy decides between $|\mathcal{V}|$ discrete actions.

Converting from Turns to Tokens. During inference, the VLM generated a_t by tokenizing $(o_t^{\text{text}}, o_t^{\text{img}})$ into a multi-modal sequence and autoregressively continuing that sequence until a stop condition was met. Let $\tilde{a}$ be the tokenized action string with length l. The true input sequence to the VLM is $\tilde{a}$ concatenated to the end of a tokenized $(o^{\text{img}}, o^{\text{text}})$. VLM_θ outputs a sequence of d-dimensional vector representations where the last $l+1$ indices correspond to the action decisions we want to optimize. We will summarize the process of retrieving the representations used to generate action tokens with the notation $\text{VLM}_\theta(o^{\text{img}}, o^{\text{text}}, a) = h \in \mathbb{R}^{(l+1)\times d}$. Querying the VLM without the action text ($\text{VLM}_\theta(o^{\text{img}}, o^{\text{text}})$) returns the d-dimensional representation (of length one) that is used to begin the action reply. We can think of $\text{VLM}_\theta(o^{\text{img}_t}, o_t^{\text{text}}, a_t) = h_t$ as a sequence of RL states. The policy begins with h_t^0, selects a token action $\tilde{a}_t^0$, sees h_t^1, and receives no reward. At the end of the turn, the policy selects a token that terminates generation and returns control to the environment. From the policy's perspective the next observation is the first action query of the next turn, or $\text{VLM}_\theta(o_{t+1}^{\text{img}}, o_{t+1}^{\text{text}})$, with a reward and termination signal from the environment r_{t+1} and d_{t+1}. Fol-

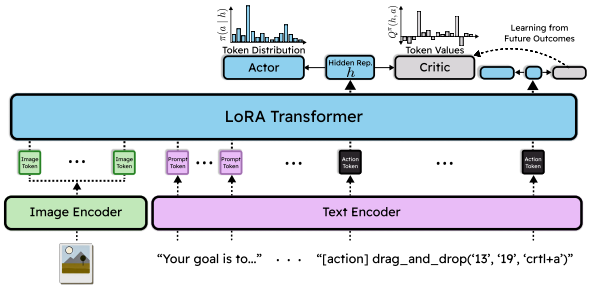

Figure 3: **VLM Actor-Critic.** We visualize a standard decoder-only VLM. Our method adds a second output head (critic) to estimate the future return achieved by selecting each token in the vocabulary. The critic filters the dataset of the language modeling head (actor).

lowing this logic, we can convert a turn-based transition $((o_t^{\text{text}}, o_t^{\text{img}}), a_t, r_{t+1}, d_{t+1}, (o_{t+1}^{\text{text}}, o_{t+1}^{\text{img}}))$ to token-based RL transitions by computing:

$$h = \text{VLM}_\theta(o_t^{\text{img}}, o_t^{\text{text}}, a_t) \qquad (1) \qquad\qquad h' = \not{\nabla}\text{VLM}_\theta(o_{t+1}^{\text{img}}, o_{t+1}^{\text{text}}) \qquad (2)$$

Where $\not{\nabla}$ is a stop-gradient that puts the VLM in inference mode. The expanded batch of RL transitions is: $(h^0, \tilde{a}_t^0, r^1 = 0, d^1 = 0, h^1), (h^1, \tilde{a}_t^1, r^2 = 0, d^2 = 0, h^2), \ldots, (h^l, \tilde{a}_t^l, r^{l+1} = r_{t+1}, d^{l+1} = d_{t+1}, h^{l+1} = h')$. This process is visualized in Figure 2. After the conversion, we can apply RL exactly as we would in any other setting, and gradients will flow back to the base VLM.

Language Loss. SFT is the most popular approach to VLM agent fine-tuning. In multi-turn SFT, we compute the next-token prediction error causally along the entire sequence and mask the loss so that we are only maximizing the likelihood of the agent's replies. In our notation, the SFT loss for turn i would be written:

$$\mathcal{L}_{\text{SFT}}(i) = -\frac{1}{l}\sum_{j=0}^{l} \log L_\phi(\tilde{a}_i^j \mid h_i^j) \tag{3}$$

RL would call Equation 3 behavior cloning, and its limitation is that it imitates every decision in the dataset and, therefore, cannot improve beyond its performance. We need to *filter* the dataset with some function $f(h, a) \in \{0, 1\}$ to avoid imitating sub-optimal decisions:

$$\mathcal{L}_{\text{FSFT}}(i) = -\frac{1}{l}\sum_{j=0}^{l} f(h_i^j, \tilde{a}_i^j) \log L_\phi(\tilde{a}_i^j \mid h_i^j) \tag{4}$$

The filter creates a second mask for action tokens that should not be imitated. While we could create a heuristic filter by masking low-return trajectories, we can let RL outperform heuristics by learning a filter from data. We create a second output head, Q_ψ, which takes the VLM representations as input and produces $|\mathcal{V}|$ outputs — much like the language head L_ϕ. Let $Q_\psi(h)[k]$ be the output corresponding to token k. Assuming Q_ψ outputs correct Q-values (Section 2), the filter can become:

$$f(h_i^j, \tilde{a}_i^j) = Q_\psi(h_i^j)[\tilde{a}_i^j] > Q_\psi(h_i^j)^\intercal L_\phi(h_i^j) + \beta \tag{5}$$

In RL terminology, the filter approves actions that have an *advantage* of at least β: $f(s, a) = Q^\pi(s, a) - V^\pi(s) > \beta$. We will refer to Equation 4 with the filter in Equation 5 as Advantage Filtered SFT (AFSFT).

Value Loss. Our filter will only be useful if we can train Q_ψ to output accurate Q-values. We train Q_ψ with a one-step temporal difference (TD) loss where we bootstrap an estimate of $Q(s, a)$: $y^i = r^{i+1} + \gamma(1 - d^{i+1})\not{\nabla}(Q_\psi(h^{i+1})^\intercal L_\phi(h^{i+1}))$. For stability, y uses a moving average of the critic parameters ψ (Lillicrap, 2015). We update our current predictions based on the improved estimate:

$$\mathcal{L}_{\text{TD}}(i) = \frac{1}{l}\sum_{j=0}^{l} (Q_\psi(h_i^j)[\tilde{a}_i^j] - y_i^j)^2 \tag{6}$$

Training. We can now put these RL components together to update our VLM. Figure 3 provides a high-level overview. Each training step samples a batch of turn-based transitions from our dataset $\mathcal{D}$. We convert VLM agent data to a sequential RL problem with one forward pass of the VLM on turn i (Equation 1) and one inference-only pass on the following turn $i + 1$ (Equation 2). We then compute the critic loss (Equation 6) and actor loss (Equations 4 and 5), and take a gradient step on the joint loss:

$$\mathcal{L}_{\text{VLMQ}} = \mathop{\mathbb{E}}_{i \sim \mathcal{D}} \left[\mathcal{L}_{\text{FSFT}}(i) + \lambda \mathcal{L}_{\text{TD}}(i)\right] \tag{7}$$

Note that the use of stop-gradients in TD targets y and the filter f mean that the actor (language head) L_ϕ only optimizes $\mathcal{L}_{\text{FSFT}}$ while the critic Q_ψ only optimizes $\mathcal{L}_{\text{TD}}$. However, the VLM trains on both objectives simultaneously and balances their values with a hyperparameter λ. We use standard techniques to save GPU memory during optimization, including training in mixed precision with an 8-bit optimizer (Dettmers et al., 2021). We also use LoRA to adapt the VLM's behavior with a small number of trainable parameters (Hu et al., 2022). However — unlike the other efficiency techniques — LoRA plays an important conceptual role and would likely still be useful even if

hardware resources were not a concern. Optimizing the VLM on task-specific data with $\mathcal{L}_{\text{VLMQ}}$ will almost surely degrade its performance on other tasks. In theory, $\mathcal{L}_{\text{FSFT}}$ lets us retain general behavior by imitating reward-free SFT tokens from a standard instruction-tuning dataset alongside RL training, but this would be expensive in practice. LoRA makes this unnecessary by letting us discard the weight adapters and recover the base model at any time during deployment.

Extending SFT. Our base VLM optimizes a policy loss that is a safe improvement over SFT in most cases and lets us recover SFT if necessary. When the learned filter f is noisy or near its initialization, Eq. 4 effectively becomes SFT on random subsets of the dataset. When Q-value estimation is successful, Eq. 4 creates a policy that outperforms its training data by declining to imitate action token outputs thought to be sub-optimal. There are some concerns when training on expert datasets where the dataset is of uniformly high quality, as the margin of advantages used to compute Eq. 4 would be close to zero. If we know that our dataset contains optimal decisions, we could manually recover standard SFT with $\beta = -\infty$ and $\lambda = 0$. Like SFT, but unlike many foundation model RL actor-critics, $\mathcal{L}_{\text{VLMQ}}$ requires a single base VLM.

Offline-to-Online Fine-Tuning. In applications like robotics, offline pre-training is often motivated by safety or cost reduction. Many VLM agent applications involve safe and inexpensive simulated environments, but learning from existing datasets will be important for two different reasons. First, it will help us overcome the challenge of outputting correct action syntax without a separate SFT stage. We can use more capable models or other techniques to generate a diverse dataset of valid action syntax but do not need to assume those actions are high-quality semantic decisions. The token-based action format allows agents to copy syntax details from poor actions without imitating the entire semantic decision. The second benefit addresses token vocabulary ($|\mathcal{V}|$) action spaces that are orders of magnitude larger than we would typically see in discrete actor-critics (Table 1). Many token actions are not relevant to the task and will never be selected or appear in our dataset (Snell et al., 2022). The outputs of Q_ψ for these tokens will never be optimized and should effectively be treated as random numbers that do not represent meaningful value estimates. Many offline RL techniques manage this miscalibration by constraining updates to actions that appear in their (small) static dataset. Token action spaces are so large that they create a unique situation where this constraint is necessary even when we do allow for online interaction.

6 EXPERIMENTS

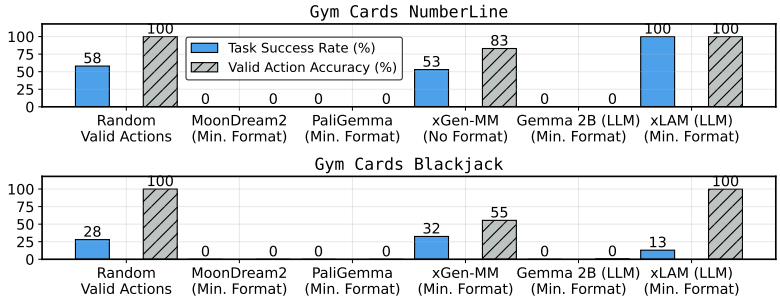

Figure 4: **Gym Cards Model Evaluations.** We compare the task success and the rate of valid model output syntax of base models in two Gym Cards environments. LLM prompts replace the image with an equivalent text description.

We begin by highlighting the difficulty of prompting base VLMs to succeed in simple agent-like tasks. Gym Cards (Zhai et al., 2024) converts toy RL decision-making tasks like the game of Blackjack into testbeds for VLM-Agent research. The true action space $\mathcal{A}'$ is a small set of discrete choices, but following (Liu et al., 2024b), we make this domain more representative of the broader problems involved in prompting the model to output specific syntax by setting $\texttt{parse}_{\text{env}}$ to match a text action with only light corrections. We choose the $\texttt{parse}_{\text{agent}}$ syntax of "`[action]`" because it is trivial to prompt the base model to pass this check by including the action string at the beginning of the reply. Additional training and action parsing details are provided in Appendix A.2. We follow a general prompting approach summarized in Section 5.1, where a text input includes instructions about the objective of the task and the expected syntax of the models' action replies (with an example). Appendix A.4 contains example prompts for our experiments.

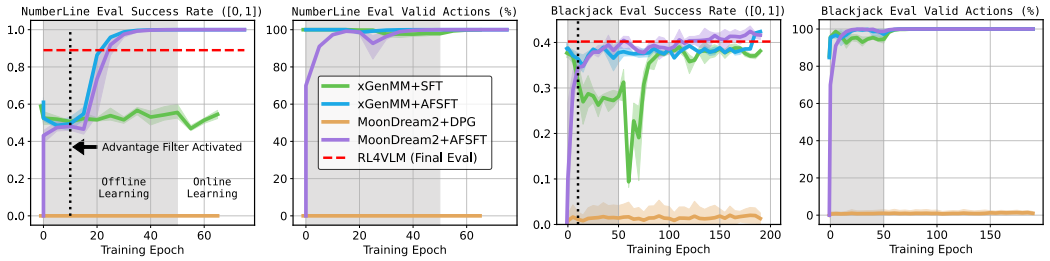

Figure 5: **Offline-to-Online Gym Cards. (Left, Center Right)** Task success rate ($[0, 1]$) compared against a reference score for a similar setup in Zhai et al. (2024). A gray background indicates offline learning on a fixed dataset. A white background indicates that online environment interaction is enabled. **(Center Left, Right)** Action syntax accuracy during evaluation.

Figure 4 compares the performance of the three base VLMs in Table 1 using only prompting and without further fine-tuning. We see that it is surprisingly difficult to get these smaller open-weight VLMs to cooperate with the expected action syntax. Action syntax is closely related to tool-use (Section 2), where larger models tend to be more accurate, and this capability is more reliable in larger LLMs. We can demonstrate this by using privileged environment data to add a text summary of the image to the observation prompt. LLMs can then be used in a nearly equivalent setup by simply ignoring the visual input. xLAM (Zhang et al., 2024b) — an LLM specifically targeted at tool-use and function-calling applications — achieves high accuracy. Off-policy RL fine-tuning will let us learn from the interactions of larger models (including LLMs with alternate text prompts). However, even when models have reasonable syntax accuracy, they might only output certain subsets of the action space or make incorrect semantic decisions that will not solve the task. Appendix A.2 Figure 8 repeats this experiment in the BabyAI (Chevalier-Boisvert et al., 2019) domain and explores the addition of "thought"-style prompting. We find all VLMs output inaccurate syntax, and only xGen-MM has a non-zero success rate.

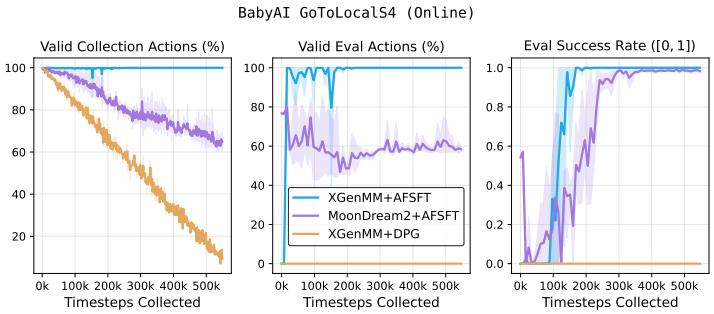

Figure 6: **Online RL in BabyAI.** An ϵ-greedy exploration schedule allows for online learning by introducing valid action syntax that can be refined by the critic filter to produce a successful policy.

We can improve performance by taking these same VLM-agents and fine-tuning them with $\mathcal{L}_{\text{VLMQ}}$ (Eq. 7). Figure 5 demonstrates the technique in two Gym Cards tasks. We initialize a dataset $\mathcal{D}$ to the actions generated by a random policy; this data makes sub-optimal (random) decisions that are properly formatted as text. We also mix in data collected by prompting base VLMs — which are both inaccurate and low-performance. More detail about the initial offline datasets used in our experiments is provided in Appendix A.3.2. For demonstration purposes, the advantage threshold is initialized to $\beta = -\infty$ (Eq. 5) such that the language head (RL actor) is optimizing SFT on this sub-optimal dataset while beginning to optimize the critic value regression loss (Eq. 6). This quickly brings both xGenMM and MoonDream2 to a similar performance level. After a few hundred gradient steps, we schedule the advantage threshold to reach $\beta = 0$; the critic is now filtering sub-optimal tokens within each turn's reply, and the agent's performance improves to match or exceed the RL4VLM (Zhai et al., 2024) reference score. Because $\mathcal{L}_{\text{VLMQ}}$ is equally applicable online and offline, we are free to use our current RL agent to collect additional data. In this case, the agents have already converged to high performance but maintain this level as the buffer fills with fresh data. In RL, it is not uncommon to see a dip in performance during the transition between offline and online learning due to distribution shift and related optimization challenges (Zhang et al., 2024c), but Eq. 4 avoids many of these obstacles, and we do not observe instability.

Figure 5 includes results from an alternative actor update that replaces Eq. 4 with a more standard online objective that directly maximizes Q-values according to the current critic: $\mathcal{L}_{\text{DPG}}(i) = -\frac{1}{l}\sum_{j=0}^{l} L_\phi(h_i^j)^\intercal \nabla Q_\psi(h_i^j)$. While this objective is effective in discrete actor-critics in more typ-

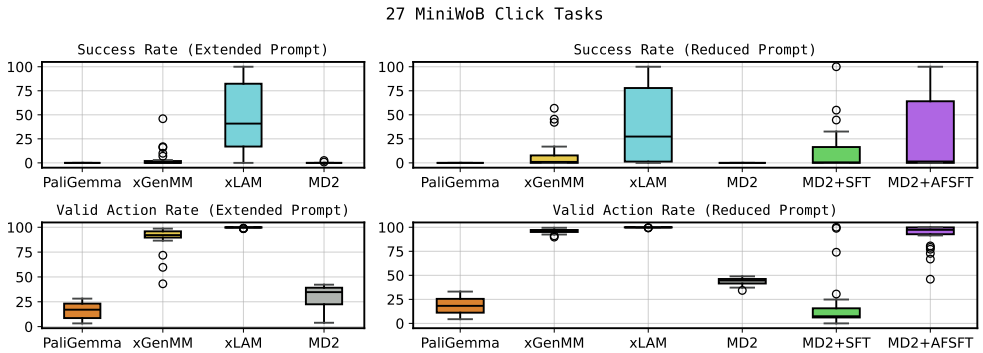

Figure 7: **Offline RL in BrowserGym MiniWoB.** Box plots indicate the median and interquartile range across 27 "click" tasks. Models are evaluated with and without an "extended prompt" that includes the webpage DOM and language descriptions of the code-like action space (Appendix A.4). Individual task results are listed in Tables 5 and 6. "MD2" abbreviates MoonDream2.

ical RL settings, it is not effective here due to miscalibrated Q-values in the critic network's output space resulting from large token vocabularies.

Training an RL policy from a fully offline dataset more closely resembles the setup of SFT agent fine-tuning, where the ability to collect online data to improve performance is a convenient feature. However, we could also fine-tune the VLM entirely online using a buffer of its own experience. Figure 6 demonstrates online learning in a BabyAI task. BabyAI is another domain where the true action space is a small set of discrete choices, which allows for a simple exploration strategy where we can replace the model's response with a random (valid) decision with probability ϵ that decays over time. Because the dataset then contains some valid actions that avoid the syntax reward penalty, the AFSFT update learns to favor them over the model's own (inaccurate) outputs, leading to a spike in action accuracy. From there, the Q filter can prefer actions that lead to task success over other valid but sub-optimal decisions, and the VLM's success rate rises to nearly 100%.

In domains where the true action space is open-ended text or state-dependent code, it is more difficult to generate valid exploration actions. We evaluate this with the BrowserGym (Drouin et al., 2024) variant of the MiniWoB++ benchmark (Shi et al., 2017; Liu et al., 2018). MiniWoB is a suite of locally hosted browser tasks for RL agents. BrowserGym creates a standardized interface for web tasks with code-like action syntax (Figure 11). We use 3 VLMs and the xLAM LLM to collect a dataset of 340k actions across 27 different "click" navigation tasks. Figure 7 summarizes these models' action accuracy and task success rate. We compare the performance of a more detailed prompt that includes HTML text in addition to a browser screenshot against a simplified variant that leads to more affordable input lengths. Our dataset is partially generated by filling the agent's action reply with the beginning of a randomly selected BrowserGym function call — adding many inaccurate or irrelevant browser commands to the dataset and creating a challenging offline RL problem. We then fine-tune the worst performing VLM (MoonDream2) using both SFT (Eq. 3) and AFSFT (Eq. 7). Figure 7 highlights $\mathcal{L}_{\text{VLMQ}}$'s ability to sift through noisy datasets in order to recover a MoonDream2 policy that more closely resembles the larger xLAM model.

7 Conclusion

The standard approach to training VLM agents typically involves using a larger, more capable model to generate a dataset, which is then used to train a smaller model via supervised fine-tuning (SFT). In this work, we show that advantage-filtered supervised fine-tuning (AFSFT) — an offline reinforcement learning technique that masks actions predicted to reduce downstream performance — can effectively replace SFT in scenarios where datasets are suboptimal or action formats are unreliable. AFSFT enables a smooth transition between learning from static offline data and interacting with the environment online, while requiring only a minimal architectural change: the addition of a secondary token output head atop a fine-tunable base model.

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. *arXiv preprint arXiv:2311.18232*, 2023.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International conference on machine learning*, pp. 104–114. PMLR, 2020.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, 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.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.

Junzhe Chen, Xuming Hu, Shuodi Liu, Shiyu Huang, Wei-Wei Tu, Zhaofeng He, and Lijie Wen. Llmarena: Assessing capabilities of large language models in dynamic multi-agent environments. *arXiv preprint arXiv:2402.16499*, 2024a.

William Chen, Oier Mees, Aviral Kumar, and Sergey Levine. Vision-language models provide promptable representations for reinforcement learning. *arXiv preprint arXiv:2402.02651*, 2024b.

Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, and Keith Ross. Bail: Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:18353–18363, 2020.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

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

Yicheng Feng, Yuxuan Wang, Jiazheng Liu, Sipeng Zheng, and Zongqing Lu. Llama rider: Spurring large language models to explore the open world. *arXiv preprint arXiv:2310.08922*, 2023.

Moghis Fereidouni and AB Siddique. Search beyond queries: Training smaller language models for web interactions via reinforcement learning. *arXiv preprint arXiv:2404.10887*, 2024.

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

Raj Ghugare, Matthieu Geist, Glen Berseth, and Benjamin Eysenbach. Closing the gap between td learning and supervised learning–a generalisation point of view. *arXiv preprint arXiv:2401.11237*, 2024.

Yiduo Guo, Zekai Zhang, Yaobo Liang, Dongyan Zhao, and Duan Nan. Pptc benchmark: Evaluating large language models for powerpoint task completion. *arXiv preprint arXiv:2311.01767*, 2023.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*, 2024.

Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. A survey of reinforcement learning from human feedback. *arXiv preprint arXiv:2312.14925*, 2023.

Vikhyat Korrapati. Moondream2: A tiny vision language model. `https://huggingface.co/vikhyatk/moondream2`, 2024. DOI: `10.57967/hf/3219`.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in neural information processing systems*, 32, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. When should we prefer offline reinforcement learning over behavioral cloning? *arXiv preprint arXiv:2204.05618*, 2022.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Yuan Ling, Fanyou Wu, Shujing Dong, Yarong Feng, George Karypis, and Chandan K Reddy. International workshop on multimodal learning-2023 theme: Multimodal learning with foundation models. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5868–5869, 2023.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 26296–26306, 2024a.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024b.

Michael Lutz, Arth Bohra, Manvel Saroyan, Artem Harutyunyan, and Giovanni Campagna. Wilbur: Adaptive in-context learning for robust and accurate web agents. *arXiv preprint arXiv:2404.05902*, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

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.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Xinyue Yang, Jiadai Sun, Yu Yang, Shuntian Yao, Tianjie Zhang, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2023.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N0I2RtD8je.

John Schulman. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Charlie Victor Snell, Ilya Kostrikov, Yi Su, Sherry Yang, and Sergey Levine. Offline rl for natural language generation with implicit language q learning. In *The Eleventh International Conference on Learning Representations*, 2022.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*, 2024.

Abishek Sridhar, Robert Lo, Frank F Xu, Hao Zhu, and Shuyan Zhou. Hierarchical prompting assists large language model on web navigation. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Andrew Szot, Max Schwarzer, Harsh Agrawal, Bogdan Mazoure, Rin Metcalf, Walter Talbott, Natalie Mackraz, R Devon Hjelm, and Alexander T Toshev. Large language models as generalizable policies for embodied tasks. In *The Twelfth International Conference on Learning Representations*, 2023.

Heyi Tao, Sethuraman TV, Michal Shlapentokh-Rothman, Derek Hoiem, and Heng Ji. Webwise: Web interface control and sequential exploration with large language models. *arXiv preprint arXiv:2310.16042*, 2023.

Oguzhan Topsakal and Tahir Cetin Akinci. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In *International Conference on Applied Engineering and Natural Sciences*, volume 1, pp. 1050–1056, 2023.

Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude, 2016. URL https://arxiv.org/abs/1602.07714.

Qing Wang, Jiechao Xiong, Lei Han, Han Liu, Tong Zhang, et al. Exponentially weighted imitation learning for batched historical data. *Advances in Neural Information Processing Systems*, 31, 2018.

Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. *arXiv preprint arXiv:2402.03681*, 2024.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33:7768–7778, 2020.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022a. URL https://arxiv.org/abs/2206.07682.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022b.

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

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-copilot: Towards generalist computer agents with self-improvement. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. URL https://openreview.net/forum?id=3WWFrg8UjJ.

Yuchen Xiao, Yanchao Sun, Mengda Xu, Udari Madhushani, Jared Vann, Deepeka Garg, and Sumitra Ganesh. O3d: Offline data-driven discovery and distillation for sequential decision-making with large language models. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, et al. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*, 2023.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*, 2023.

Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. Language agents with reinforcement learning for strategic play in the werewolf game. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=usUPvQH3XK.

Le Xue, Manli Shu, Anas Awadalla, Jun Wang, An Yan, Senthil Purushwalkam, Honglu Zhou, Viraj Prabhu, Yutong Dai, Michael S Ryoo, et al. xgen-mm (blip-3): A family of open large multimodal models. *arXiv preprint arXiv:2408.08872*, 2024.

Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Lumos: Learning agents with unified data, modular design, and open-source llms. *arXiv preprint arXiv:2311.05657*, 2023.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.

Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, et al. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *arXiv preprint arXiv:2405.10292*, 2024.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*, 2024a.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*, 2024b.

Yinmin Zhang, Jie Liu, Chuming Li, Yazhe Niu, Yaodong Yang, Yu Liu, and Wanli Ouyang. A perspective of q-value estimation on offline-to-online reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 16908–16916, 2024c.

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

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. In *Forty-first International Conference on Machine Learning*, 2024.

Banghua Zhu, Hiteshi Sharma, Felipe Vieira Frujeri, Shi Dong, Chenguang Zhu, Michael I Jordan, and Jiantao Jiao. Fine-tuning language models with advantage-induced policy alignment. *arXiv preprint arXiv:2306.02231*, 2023.

A Appendix

A.1 Additional Related Work

RL is used in the post-training process to realign a base model with human feedback (RLHF) and enforce safety measures (Ouyang et al., 2022; Kaufmann et al., 2023; Bai et al., 2022). RLHF typically uses online (on-policy) updates based on constrained policy gradient techniques (Zhu et al., 2023; Ahmadian et al., 2024) that prevent the fine-tuned policy from over-optimizing a learned reward model of human preferences (Lambert et al., 2024). Unlike the more general "agent" setting, RLHF interactions typically last for one turn — a simplification that allows for alternate updates like DPO (Rafailov et al., 2023). VLMs can also be used to create reward functions for training smaller RL policies from scratch (Rocamonde et al., 2024; Wang et al., 2024). Our work updates the VLM to output decisions directly, which is useful for text-based actions because learning to output language from a randomly initialized policy via RL is sample inefficient. Another category of methods fine-tunes foundation models as policies while avoiding the action syntax challenge that comes with text actions (Section 2). When the environment has a small discrete set of valid actions, we can iterate over our options and compare their likelihood according to the LLM (Carta et al., 2023). We could also discard text outputs and use the hidden state of the base model to train a new action output layer from scratch (Chen et al., 2024b; Szot et al., 2023; Xu et al., 2024).

A.2 Additional Implementation Details

A.2.1 Training

Table 2 lists key hyperparameters and version control details for fine-tuning the MoonDream2 and xGen-MM models (with SFT or RL). Our LoRA (Hu et al., 2022) settings are based on recommendations from SFT fine-tuning pipelines by the models' authors.

	MoonDream2	**xGen-MM**
Version Notes	revision 2024-05-20	xgen-mm-phi-3-mini-instruct-r-v1
Precision	bf16	bf16
LoRA		
r	8	8
α	16	16
Dropout	.05	.05
Param Groups	"mixer.Wqkv"	["qkv_proj", "q_proj", "gate_up_proj", "down_proj"]
Tunable Params	1,572,864	12,582,912

Table 2: **VLM Model Training Details.**

Table 3 provides a full listing of the hyperparameters involved in collecting environment data and optimizing the VLM on the actor-critic RL loss (Eq. 7).

A.2.2 Action Prompting and Parsing

Action Parsing. The goal of our method is to be a flexible replacement for SFT updates in an agent setting. Therefore, we need our RL update to be compatible with arbitrary prompting schemes, especially chain-of-thought (Wei et al., 2022b) or ReACT-style (Yao et al., 2022b) reasoning. We

	Gym Cards	BabyAI	MiniWoB
Exploration / Language Sampling			
Language Temperature During Exploration	.4	.4	.5
Max Reply Tokens	24	24	32
Expl. ϵ Schedule (Start $\rightarrow$ Stop), [Over Online Epochs]	(.35 $\rightarrow$ .05), [50]	(1.0 $\rightarrow$ .1), [300]	N/A (Offline)
Explore Action Prompt	random	random	N/A
Exploit / Eval Action Prompt	agent	agent	agent
Optimization			
Minibatch Size Per GPU (A100 40GB)	3	3	1
Parallel GPUs	2	2	1
Gradient Accumulation	4	4	8
Learning Rate (VLM)	1e-4	1e-4	1e-4
Learning Rate (Critic MLP)	6e-4	8e-4	7e-4
Grad Norm Clip	2	2	2
VLM Linear LR Warmup Steps	250	500	300
Critic Linear LR Warmup Steps	100	50	100
Learning Schedule			
Buffer Min Size Before Training (Per GPU)	N/A (Offline Demos)	5k	N/A (Offline Demos)
Max Buffer Size (Per GPU)	50k	100k	N/A (Offline)
Parallel Environments (Per GPU)	6	6	8
Start Collecting on Epoch	50	0	∞
Timesteps Collected Per Env Per Epoch	96	150	0
Grad Updates Per Epoch	512	512	3072
Total Epochs	75 (NumberLine) 200 (Blackjack)	600	50
TD Learning			
Target Critic τ	.008	.008	.009
Critic Loss Weight λ	1.0	8.0	1
Critic MLP Hid. Dim.	d_{model}	d_{model}	400
γ	.995	.995	.995
PopArt β	5e-4	5e-4	5e-4
PopArt Init ν	100	100	100

Table 3: **Training Hyperparameters.** PopArt (van Hasselt et al., 2016) is an RL implementation detail that helps normalize Q-value predictions and reduces tuning of the critic loss weight (λ) and learning rate.

can allow for this by assuming a method $\texttt{parse}_{\text{agent}}$ (Fig. 1) can modify the model's text output before it is passed to the environment. In this work, we take a simple approach and remove all text up to and including the special tag "`[action]`". This allows the model room to reason in open-ended text before selecting its action. We use "`[action]`" instead of more common json agent formats because we are evaluating small-scale models that struggle to output the requested syntax, and we can easily prompt the agent with an "`[action]`" tag.

In order for the environment simulator to execute the agent's actions, it needs to be able to parse the intended action from open-ended language outputs ($\texttt{parse}_{\text{env}}$ in Figure 1). In domains like the BrowserGym version of MiniWoB (and many real-world applications of closed-source LLM/VLMs), actions are interpreted as function calls with state-dependent arguments. Even though the underlying decision-making problem of Gym Cards and BabyAI can be represented by discrete actions, we follow the lead of Zhai et al. (2024) in simulating an action syntax barrier by requiring the model to output a string from a set of discrete choices in the prompt (Figure 9). In Gym Cards, we require an exact match after removing any whitespace and markdown tags. In BabyAI, we are slightly more lenient and parse the first of any match in the output string.

When the model produces syntax outside of the expected format, it cannot advance the simulator and collect useful experience, even if it subjectively demonstrates some understanding of the task. This dilemma is common in LLM evaluations and can sometimes be the root cause of "emergent" behavior (Wei et al., 2022a): models that are only a slight improvement in terms of training loss or human preferences can offer a dramatic improvement in automated evaluations because they cross the somewhat arbitrary threshold of being able to interpret the requested format. As discussed in Section 2, this capability has been emphasized and improved by tool-use benchmarks. However, it is an especially difficult problem in RL because invalid action syntax prevents us from reaching environment rewards and creates a sparse-reward exploration problem. The space of possible text actions is so large that default RL strategies, like randomly sampling tokens from the action space, are unlikely to find valid behavior. At the same time, the base model may be biased towards cer-

tain inaccuracies, such that sampling with higher temperature is unlikely to discover the successful examples necessary for RL to correct its mistakes.

Action Prompting. Action prompting refers to strategies that modify the `prompt` method (Figure 1) to induce more accurate action syntax or a larger variety of decisions (or both). In the main text and appendix, "No Format" or "agent" formatting refers to the natural default of letting the model respond to the observation solely by sampling from the outputs of its own language head (policy). "Minimum Format" tries to overcome the most obvious bottleneck of not outputting the "`[action]`" tag by beginning the agent's reply with a tokenized version of that string. "Random Valid Actions" or "random prompting" selects random actions from the environment's underlying action space $\mathcal{A}'$ (Section 3) and maps them to valid text. In BrowserGym, this strategy is not practical because actions require knowledge of HTML element IDs and open-ended text. Instead, we use a "partial" prompting strategy where we sample from prefixes of valid actions. More specifically, we can sample method names from the browser interface and have the agent attempt to complete the arguments for that call. These prompting strategies create a version of ϵ-greedy exploration in RL that is used during online learning and dataset collection. With probability ϵ, we can sample actions with an exploration strategy (e.g., a random valid action). Otherwise, we can use an exploitation strategy (e.g., the agent's greedy output).

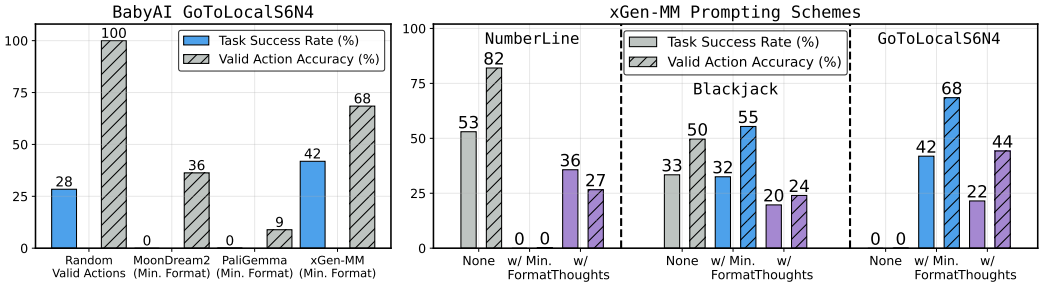

Figure 8: **Base Model Evaluations and Prompting.** (Left) We repeat the base model action syntax and success rate evaluation in a BabyAI domain. (Right) Comparing basic prompt variants with xGenM-MM. "w/ Min. Format" adds assistance with action parsing syntax. "w/ Thoughts" adds ReACT-style Yao et al. (2022b) reasoning.

A.3 Environment Details

A.3.1 Reward Functions

Our problem formulation assumes multi-turn interactions where rewards are given after every action decision. We can take advantage of this by adding reward terms beyond the reward function of the inner environment — which is often a binary indicator of success or failure at the end of the episode. We add per-timestep penalties for outputting invalid action choices in order to encourage more useful data collection (Appendix A.2.2). We also introduce time limits and penalties for running out of time in the environment or getting stuck in a cycle of several invalid actions. Modifications to the standard reward functions of each environment are detailed in Table 4.

	Gym Cards	**BabyAI**	**MiniWoB**
Inner Env Reward Multiplier	$\times 1$	$\times 100$	$\times 1$
On Invalid Action	-2	-1	-2
Timeout on Consecutive Invalid Actions	3	5	5
Timeout on Total Steps	10	64	25
On Timeout	-10	-200	-10

Table 4: **Reward Terms.** Reward functions are combinations of their environment's natural reward signal (in this case: a success/failure indicator given at the end of an episode) and per-timestep signals to encourage correct action formatting.

A.3.2 Offline Datasets

- **Gym Cards.** Each of the Blackjack and NumberLine initial (offline) datasets contains 28800 transitions. 24k transitions were generated by random valid actions while 1.6k were the outputs of MoonDream2 without any action action formatting. Both domains collected 2.4k actions from xGen-MM. Blackjack assisted xGen-MM with minimum action formatting, while in NumberLine, we could reach higher performance without action formatting (Figure 4) due to a tokenization quirk. VLM sampling used a temperature of 0.4.
- **MiniWoB.** The MiniWoB dataset contains 364320 transitions spread approximately evenly across 27 tasks (Table 5). With probability $\epsilon = .5$, a partial action was generated to encourage a diverse dataset (Appendix A.2.2). Otherwise, actions were sampled from the base VLM/LLM with temperature 0.5 (without formatting assistance). PaliGemma, xGen-MM, and MoonDream2 each created 20% of the dataset, while xLAM generated the remaining 40%. xLAM is the most accurate model in terms of valid syntax (Table 6).

A.3.3 Additional MiniWoB Results

The results of Figure 7 are listed according to the 27 individual browser tasks in Tables 5 and 6.

	PaliGemma	xGenMM	xLAM	MD2	MD2+SFT	MD2+AFSFT
click-tab	0.0	3.1	**94.8**	0.0	16.7	<u>66.5</u>
click-tab-2-easy	0.0	**1.0**	**1.0**	0.0	0.0	0.0
click-dialog-2	0.0	16.9	**84.3**	0.0	21.7	<u>66.8</u>
click-shape	0.0	0.8	**1.7**	0.0	0.0	<u>1.5</u>
click-checkboxes-transfer	0.0	0.0	**71.5**	0.0	14.9	<u>61.7</u>
click-shades	0.0	0.0	0.0	0.0	0.0	0.0
click-checkboxes	0.0	1.1	**52.2**	0.0	5.6	<u>33.8</u>
click-checkboxes-large	0.0	0.0	**5.2**	0.0	0.0	0.0
click-tab-2-hard	0.0	0.0	0.0	0.0	0.0	**0.5**
click-test-transfer	0.0	42.2	<u>90.4</u>	0.0	54.9	**91.3**
click-collapsible-2	0.0	0.0	**1.0**	0.0	0.0	0.0
click-test	0.0	56.8	**100.0**	0.0	**100.0**	**100.0**
click-menu	0.0	6.2	**31.0**	0.0	3.0	<u>23.3</u>
click-collapsible	0.0	0.0	**0.3**	0.0	0.0	0.0
click-checkboxes-soft	0.0	0.0	**33.0**	0.0	3.9	<u>29.8</u>
click-button	0.0	10.6	**95.6**	0.0	22.8	<u>91.7</u>
click-tab-2-medium	0.0	1.0	**19.7**	0.0	<u>9.2</u>	0.0
click-tab-2	0.0	0.0	**1.9**	0.0	0.0	0.0
click-option	0.0	0.0	**66.8**	0.0	0.0	0.0
click-menu-2	0.0	0.0	**8.9**	0.0	0.0	0.0
click-color	0.0	0.0	0.0	0.0	0.0	0.0
click-widget	0.0	14.5	**60.7**	0.0	16.2	<u>58.5</u>
click-dialog	0.0	4.1	**100.0**	0.0	32.6	**100.0**
click-link	0.0	9.3	0.0	0.0	0.5	**11.8**
click-scroll-list	0.0	1.0	**27.4**	0.0	0.0	0.0
click-button-sequence	0.0	<u>1.8</u>	**5.6**	0.0	0.0	0.0
click-test-2	0.0	45.5	**90.4**	0.0	44.6	<u>89.2</u>

Table 5: **BrowserGym MiniWoB Success Rates (%).** Evaluated over a sample of $5,000$ timesteps per task. **Bold** and <u>underlined</u> entries denote the best and second-best performance per task, respectively.

	PaliGemma	xGenMM	xLAM	MD2	MD2+SFT	MD2+AFSFT
click-tab	13.3	97.5	99.7	39.8	7.0	**100.0**
click-tab-2-easy	16.7	95.1	**99.7**	49.0	6.6	91.5
click-dialog-2	16.0	95.7	**100.0**	44.6	6.7	99.9
click-shape	26.6	96.1	**99.3**	46.0	9.6	77.1
click-checkboxes-transfer	10.4	93.4	**100.0**	44.4	9.6	94.0
click-shades	28.4	91.0	99.9	47.8	74.0	**100.0**
click-checkboxes	11.1	92.5	**100.0**	42.9	10.2	95.6
click-checkboxes-large	4.4	95.0	**99.5**	34.3	0.0	46.0
click-tab-2-hard	5.8	89.8	**99.9**	41.3	0.3	79.1
click-test-transfer	33.1	96.6	**100.0**	39.0	22.7	96.8
click-collapsible-2	21.5	97.6	**99.6**	45.3	12.7	98.2
click-test	27.3	99.5	**100.0**	37.2	**100.0**	**100.0**
click-menu	22.5	92.9	**100.0**	46.6	6.4	99.8
click-collapsible	18.2	97.5	99.9	46.7	98.9	**100.0**
click-checkboxes-soft	7.4	96.0	**100.0**	42.5	7.0	99.5
click-button	20.0	95.3	**100.0**	44.8	6.6	95.0
click-tab-2-medium	17.1	96.6	**99.8**	48.4	5.1	97.8
click-tab-2	7.8	96.4	**99.8**	45.2	4.0	95.5
click-option	11.2	95.7	**100.0**	44.8	9.0	96.2
click-menu-2	24.4	95.9	**100.0**	46.2	5.7	72.8
click-color	27.9	97.3	**99.9**	41.8	30.6	99.5
click-widget	17.4	97.0	**100.0**	47.8	5.6	**100.0**
click-dialog	20.6	95.5	**100.0**	43.1	7.4	**100.0**
click-link	22.4	97.3	**99.9**	46.2	10.7	66.9
click-scroll-list	8.8	97.3	**99.9**	41.6	5.9	97.8
click-button-sequence	31.8	93.5	**99.9**	40.2	24.8	80.6
click-test-2	29.7	97.1	**100.0**	38.1	18.7	97.3

Table 6: **BrowserGym MiniWoB Valid Action Syntax Rates (%). Bold** and underlined entries denote the best and second-best performance per task, respectively.

A.4 Example Observation Prompts

```
Actions Taken: 0. Max Actions Allowed: 10.
You are playing a game of Blackjack.
You have long-term memory. If you want to remember something, include
     '[start remember] MESSAGE [stop remember]' in your response.

You can select an action by writing '[action] CHOICE', where CHOICE
    is one of the following choices: [ stand , hit ]. The action
    should be the final part of your answer. For example, a valid
    action would be '[action] stand'.
<|Text-Specific Info Begins|> You observe: Dealer's card: 9. Player's
     cards: 8, 8. <|Text-Specific Info Ends|>
Let's think step-by-step. Analyze the current observation, remember
    key details, and answer this question: Which action do you choose?
```

Figure 9: **Example Prompt from the Gym Cards Blackjack Domain**. `<|Text-Specific Info|>` markers denote text that is included only for LLM evaluations where the model cannot observe the corresponding image.

```
Actions Taken: 0. Max Actions Allowed: 64.
You are playing as the red triangle in a gridworld game. The triangle
     points in the direction you are facing.
You have long-term memory. If you want to remember something, include
     '[start remember] MESSAGE [stop remember]' in your response.

You select an action with'[action] CHOICE', where CHOICE is one of: [
     turn left , turn right , move forward , pick up , drop , toggle ,
     done ]. The action should be the final part of your answer. For
    example, a valid action would be '[action] turn right'.
Your goal is: go to the yellow box
Look at the image observation. Which action do you choose?
```

Figure 10: **Example prompt from the BabyAI domain.**

```
Actions Taken: 0. Max Actions Allowed: 25.
You are a browser agent. Your goal is: Switch between the tabs to find
    and click on the link "neque".
You have long-term memory. If you want to remember something, include '[
    start remember] MESSAGE [stop remember]' in your response.

Select an Action by writing '[action] `ACTION`' as the final part of your
     answer. Your `ACTION` should be a short piece of python code in the
    following format:
12 different types of actions are available.

noop(wait_ms: float = 1000)
   Examples:
      noop()

      noop(500)

scroll(delta_x: float, delta_y: float)
   Examples:
      scroll(0, 200)

      scroll(-50.2, -100.5)

fill(bid: str, value: str)
   Examples:
      fill('237', 'example value')

      fill('45', 'multi-line\nexample')

      fill('a12', 'example with "quotes"')

select_option(bid: str, options: str | list[str])
   Examples:
      select_option('a48', 'blue')

      select_option('c48', ['red', 'green', 'blue'])

click(bid: str, button: Literal['left', 'middle', 'right'] = 'left',
    modifiers: list[typing.Literal['Alt', 'Control', 'ControlOrMeta', '
    Meta', 'Shift']] = [])
   Examples:
      click('a51')

      click('b22', button='right')

      click('48', button='middle', modifiers=['Shift'])

dblclick(bid: str, button: Literal['left', 'middle', 'right'] = 'left',
    modifiers: list[typing.Literal['Alt', 'Control', 'ControlOrMeta', '
    Meta', 'Shift']] = [])
   Examples:
      dblclick('12')

      dblclick('ca42', button='right')

      dblclick('178', button='middle', modifiers=['Shift'])

hover(bid: str)
   Examples:
      hover('b8')

press(bid: str, key_comb: str)
   Examples:
      press('88', 'Backspace')

      press('a26', 'ControlOrMeta+a')
```

Figure 11: **Example prompt from the MiniWoB domain**. The goal description varies by task. **(Part 1/2)**.

```
focus(bid: str)
   Examples:
      focus('b455')

clear(bid: str)
   Examples:
      clear('996')

drag_and_drop(from_bid: str, to_bid: str)
   Examples:
      drag_and_drop('56', '498')

upload_file(bid: str, file: str | list[str])
   Examples:
      upload_file('572', 'my_receipt.pdf')

      upload_file('63', ['/home/bob/Documents/image.jpg', '/home/bob/
          Documents/file.zip'])

Only a single action can be provided at once. Example:
fill('a12', 'example with "quotes"')
 An example of a complete correct output is '[action] click('48', button
     ='middle', modifiers=['Shift'])'.


 Accessibility Tree ([bid] info, center=(x coord, y coord)):
 RootWebArea 'Click Tab Task', focused, url='file:///export/home/miniwob-
     plusplus/miniwob/html/miniwob/click-tab-2.html'
      [17] tablist '', center="(80,68)", multiselectable=False,
          orientation='horizontal'
            [18] tab 'Tab #1', center="(27,70)", expanded=True, selected=
                True, controls='tabs-1'
                  [19] link 'Tab #1', center="(27,70)", url='file:///
                      export/home/miniwob-plusplus/miniwob/html/miniwob/
                      click-tab-2.html#tabs-1'
            [20] tab 'Tab #2', center="(71,69)", expanded=False, selected
                =False
                  [21] link 'Tab #2', center="(71,70)", url='file:///
                      export/home/miniwob-plusplus/miniwob/html/miniwob/
                      click-tab-2.html#tabs-2'
            [22] tab 'Tab #3', center="(115,69)", expanded=False,
                selected=False
                  [23] link 'Tab #3', center="(115,70)", url='file:///
                      export/home/miniwob-plusplus/miniwob/html/miniwob/
                      click-tab-2.html#tabs-3'
      [24] tabpanel 'Tab #1', center="(80,137)"
            [25] paragraph '', center="(80,137)"
                  StaticText 'Enim, aliquet risus pellentesque commodo'
                  StaticText 'in'
                  StaticText 'nibh. Venenatis'
                  StaticText 'id.'
                  StaticText 'Adipiscing. Feugiat justo tellus. Tortor
                      cum'
                  StaticText 'convallis'
                  StaticText 'dolor quisque id egestas.'

<|Image-Specific Info Begins|>
Web elements in the image have been labeled with their `bid` values.
<|Image-Specific Info Ends|>
You previously output: '' with error message: None

Determine the next action to accomplish your goal. Make sure to follow
    the formatting instructions. Which action should you take?
```

Figure 12: **Example prompt from the MiniWoB domain.** The goal description varies by task. `<|Image-Specific Info|>` markers denote text that is included only for VLM evaluations where the agent can observe the corresponding browser screenshot. **(Part 2/2)**.