Online Reinforcement Learning of VLA Robotics Policy Models

Team Members: Thomas Deng, Jason Chon
Emails: tdeng23@stanford.edu, jchon5@stanford.edu

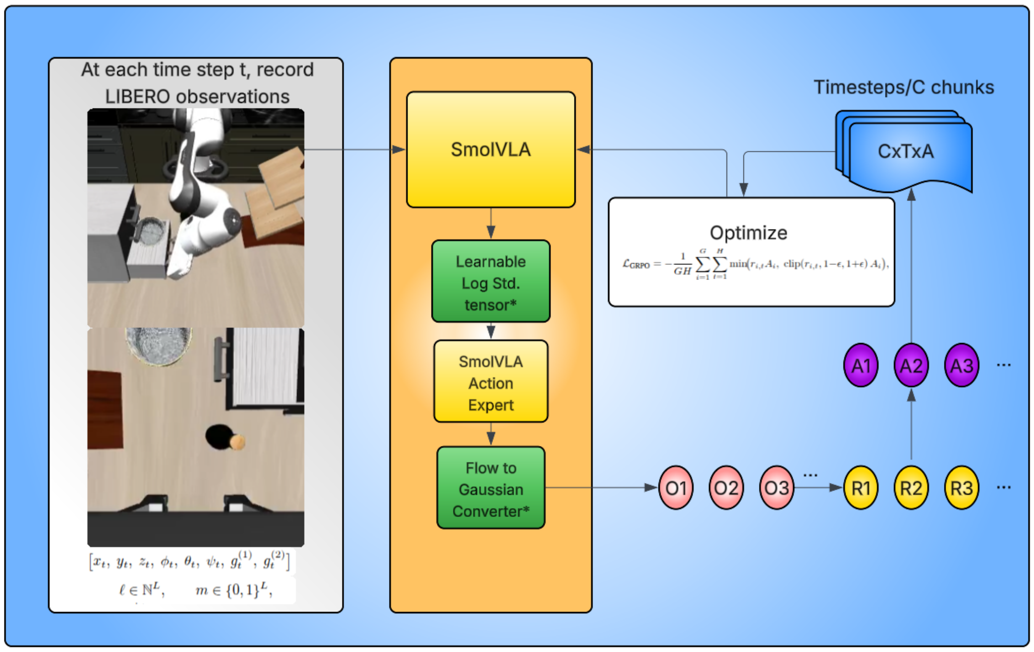
At each time step t, record LIBERO observations
$[x_t,\ y_t,\ z_t,\ \phi_t,\ \theta_t,\ \psi_t,\ g_t^{(1)},\ g_t^{(2)}]$
$\ell \in \mathbb{N}^L, \quad m \in \{0,1\}^L,$
SmolVLA
Learnable Log Std. tensor*
SmolVLA Action Expert
Flow to Gaussian Converter*
Timesteps/C chunks
CxTxA
Optimize
$\mathcal{L}_{\text{GRPO}} = -\frac{1}{GH}\sum_{i=1}^{G}\sum_{t=1}^{H}\min\big(r_{i,t}A_i,\ \text{clip}(r_{i,t},1-\epsilon,1+\epsilon)\,A_i\big),$
A1
A2
A3
...
O1
O2
O3
...
R1
R2
R3
...

Figure 1: Modified SmolVLA Architecture, *denotes modification for online RL

1 Abstract

We investigate the challenge of improving long-horizon task performance for compact vision-language-action (VLA) robotics policies via online reinforcement learning. We apply the policy-gradient algorithm Group Relative Policy Optimization (GRPO) on a pretrained VLA policy model, SmolVLA. We fine-tune on the “Long” task suite of the LIBERO simulation benchmark, a set of long-horizon, compositional manipulation tasks that the base model struggles with. An example of a "long" task would be "Put both the alphabet soup and the tomato sauce in the basekt." While our computational resources limit the amount of training, we observe that fine-tuning can substantially improve success rates on tasks where the pretrained model already had nonzero baseline capability, even though tasks it initially failed remain unsolved. This outcome highlights both the promise and the limitations RL fine-tuning for improving pretrained VLA policies. We discuss challenges around exploration, reward sparsity, and sample efficiency, and suggest directions for future work to scale up RL-based adaptation of compact VLA models.

2 Introduction

A quickly developing paradigm in robotics is represented by Vision Language Action models, such as the commercial model π_0 by Intelligence et al., which combine perception, language, and control into a single multimodal robotics policy model. Furthermore, the use of smaller foundation models such as SmolVLA allows these powerful models to exist on smaller and cheaper robots, further motivating the need to improve their performance. Yet despite their success, many generalist VLA policies fail to behave correctly on complex and longer tasks. Our goal is to address this problem, training a model in a manner that suppresses cascading failures on long-horizon tasks.

Our approach is to use online reinforcement learning, more specifically Group Relative Policy Optimization (GRPO), to fine-tune a small pretrained model, SmolVLA. We fine-tune the model on the LIBERO Long task suite, a set of long-horizon tasks the pretrained model performs poorly on. We first set up the LIBERO simulation environment, modified the SmolVLA architecture for this new training task, and implemented a GRPO training loop.

We chose to use online RL specifically because it addresses potential limitations in the pretraining process behind SmolVLA and all robotics foundation models. These models are pretrained using imitation learning, a form of supervised training that utilizes expert demonstrations of how to accomplish tasks. While this method often achieves good baseline performance, models trained using imitation learning often suffer under model drift: because the model never learns how to correct errors, during inference small errors quickly accumulate into catastrophic failures. Online RL provides the opportunity for policy models to produce and learn from both good and bad trajectories while providing a training setup more similar to test time requirements. This especially matters for the long-horizon tasks we are interested in, where there is much more opportunity for error accumulation.

For Thomas, this project spans two classes, CS229 and CS238. For CS229, work was primarily focused on tangentially related mechanistic interpretability and activation steering for robotics research. For CS238, the focus was on implementing GRPO RL for this task: setting up the simulation environment, model, and training loops then actually training a model.

The code for this project is located at this repository: https://github.com/thomasdeng2027/safe-robot-steering/

3 Related Work

Since VLA models are trained primarily using behavior cloning, an imitation learning method, these models suffer from error accumulation under distribution shift. Small deviations from expert demonstrations compound over time, resulting in cascading failures in long-horizon tasks. Li et al. (2025) In order to resolve this issue, recent work has investigated reinforcement fine-tuning (RFT), such as VLA-RFT Li et al. (2025). These techniques enable VLAs to explore, recover from suboptimal states, and adapt to unforeseen circumstances by introducing a reinforcement learning loop, often supported by world models or simulators such as LIBERO Liu et al. (2023). Because few benchmarked reward models exist for these tasks, recent methods like VLA-RFT and VLA-RL Li et al. (2025) generate their own reward signals using learned world models or similar systems. Although these approaches yield strong performance, the reward models themselves can be expensive to train and deploy.

In this work, we aim to extend the capabilities of pretrained models in a computationally affordable manner, allowing our techniques to approachable instead of prohibitively difficult and expensive to implement. To this end, we decided to fine-tune SmolVLA, a small but capable pretrained model designed specifically to be efficient and accessible Shukor et al. (2025). Additionally, as opposed to memory-intensive actor-critic reinforcement learning methods, we chose to leverage GRPO Shao et al. (2024). This policy gradient reinforcement learning method improves upon performance of former state-of-the-art methods such as Proximal Policy Optimization (PPO) Schulman et al. (2017) while removing the need to train a separate network to estimate the value function.

4 Background

The robotics simulation environment we use is the simulation environment used in LIBERO Liu et al. (2023), a widely adopted, open benchmark for vision-language-action policy models. LIBERO provides a procedurally generated set of manipulations and rearrangement tasks (130 total when combining all its suites), each defined via a task description language, initial object configurations, and goal predicates. Under the hood, LIBERO uses a physics simulator (e.g., MuJoCo + robosuite) to simulate object dynamics, robot motion, and rendering. While not originally designed for online RL, we repurpose its already existing benchmarking infrastructure to create an online loop for gathering trajectories.

The policy model we base our work on, SmolVLA Shukor et al. (2025), is a compact Vision-Language-Action model containing a pretrained vision-language encoder (VLM) fused with a lightweight action-expert transformer, trained for continuous control via flow matching. SmolVLA depends on multimodal inputs (vision, language, proprioception) to output continuous robot actions.

To adapt SmolVLA for reinforcement fine-tuning, we formalize the control problem as a Partially Observable Markov Decision Process. The simulator's full state, which includes object poses, velocities, and environment layout, defines the state space $\mathcal{S}$. The agent's partial observation $\mathcal{O}$ comprises two rendered RGB images, robot proprioceptive state, and a linguistic task instruction. The images are captured from two cameras: one in the environment facing the robot and one attached to the robot's wrist mount. The action space $\mathcal{A}$ are continuous control vectors $\mathbf{a} \in \mathbb{R}^7$ produced by SmolVLA where each entry a_i is a value in $[-1, 1]$ representing the velocity the ith joint should take for a timestep. The reward function $\mathcal{R}$ is a binary sparse reward handled automatically by LIBERO, where a trajectory is rewarded +1 if the final environment state produced by the trajectory matches a task-specific goal configuration, eg a specific object is located on top of another.

Transition dynamics are handled by LIBERO's physics simulation. While there is no explicitly defined observation or belief function, these are implicitly a part of the policy model's decision making process, as the model must have learned some notion of how to infer information from its partial observations of the environment in order to perform well.

5 Methods

5.1 Modified SmolVLA Architecture

5.1.1 Converting Deterministic Flow-Matching into a Stochastic Policy

By default, SmolVLA is a deterministic policy:

$$\pi_\theta(I_t^{front}, I_t^{wrist}, s_t, \ell) = \mathbf{a}_t$$

That is, given an observation at some timestep t consisting of image tensors I_t^{front}, I_t^{wrist}, robot state encoding s_t, and task description ℓ which remains constant across all timesteps of the rollout, the model deterministically outputs an action vector $\mathbf{a}_t$.

GRPO, as a policy gradient method, requires a stochastic policy. There are many possibilities for how to convert the deterministic SmolVLA into a stochastic policy, and we explored several options. The method we ultimately settled upon prioritizes simplicity.

We treat the output of the deterministic policy as the parameterized mean of a diagonal Gaussian distribution. That is:

$$\pi_\theta(I_t^{front}, I_t^{wrist}, s_t, \ell) = \mu_\theta^t$$

This then allows us to sample an action probabilistically at each time step:

$$\mathbf{a}_t \sim \mathcal{N}(\mu_\theta^t, \sigma_\theta)$$

where $\sigma_\theta = e^{\log \sigma_\theta}$ and $\log \sigma_\theta$ is a learnable vector shared across the entire rollout. We choose for the model to learn log standard deviation for numerical stability. Additionally, we use PyTorch's reparameterized sampling to maintain differentiability as sampling by default is a non-differentiable operation.

To minimize the risk of catastrophic forgetting and to take advantage of the pretrained model's capabilities, we initialize $\log \sigma_\theta$ to a low value of -2 to prevent wild deviation from the pretrained behavior.

Following Haarnoja et al. (2018), once an action vector is sampled, we apply $\tanh$ entry wise to the elements to ensure actions remain in $[-1, 1]$ which is required for our simulation environment. We then can apply a tanh correction to calculate the original sampled action vector's probability density:

$$\log \pi(a \mid s) = \log \mathcal{N}(u \mid \mu, \sigma^2) - \sum_{i=1}^{d} \log \left(1 - \tanh(u_i)^2\right).$$

where a denotes the $\tanh$-squashed action and u denotes the originally sampled action.

5.1.2 Encouraging Exploration

GRPO, as an online RL method, is aided by exploration so that the model can learn about the effects of a variety of actions. To this end, we tried a variety of methods to control the amount of exploration the model takes. For simplicity, we decided on the following method.

By default, the output of SmolVLA is generated via flow matching Shukor et al. (2025). During this process, given an observation, an action is generated by repeatedly denoising an initially noisy action vector:

$$x_{t-1} = x_t - dt \cdot v_\theta(x_t, t)$$

where v_θ is the vector field learned by the model's action expert.

To encourage additional exploration by the model in the generation of its trajectories, we introduce a small amount of noise into each of these denoising steps. Specifically, at each timestep the predicted vector field is perturbed by Gaussian noise before performing the update:

$$x_{t-1} = x_t - dt \cdot \tilde{v}_\theta(x_t, t)$$

where

$$\tilde{v}_\theta(x_t, t) = v_\theta(x_t, t) + \epsilon$$

and

$$\epsilon \sim \mathcal{N}(0, \sigma_d)$$

with σ_d being a hyperparameter.

This converts the deterministic flow-matching process into one where potentially drastically different final actions can be produced given the same input to the model. This stochasticity serves a different purpose than the one introduced by making the model output Gaussian distributions. Because this occurs within the action-expert, in theory high choices of σ_d can meaningfully induce exploration by forcing the action expert to denoise action vectors along different denoising trajectories. This is as opposed to setting $\log \sigma_\theta$ to an artificially high value which we found in practice would either lead to total collapse or unnecessarily jittery movements.

5.2 Online Simulation Loop

Our simulation environment is LIBERO which produces observations containing information like images from cameras and updates the simulated environment given action vectors. To convert raw LIBERO observations into inputs compatible with SmolVLA, we process each timestep's observation into five components: two RGB images, an 8-dimensional robot state vector, tokenized language instructions, and an attention mask. LIBERO provides two images, a front-facing camera view (`agentview_image`) and a wrist-mounted camera view (`robot0_eye_in_hand_image`), each originally in $H \times W \times 3$ format. We flip each image horizontally and vertically to match the orientation expected by the pretrained SmolVLA vision encoder, convert them to PyTorch tensors, normalize pixel values to the $[0, 1]$ range, and transpose them into channel-first format, yielding

$$I_t^{\text{front}},\ I_t^{\text{wrist}} \in \mathbb{R}^{3 \times H \times W}.$$

In addition to images, LIBERO provides robot state information consisting of the end-effector position (x, y, z), a quaternion orientation (q_w, q_x, q_y, q_z), and gripper finger positions (g_1, g_2). We convert the quaternion to an axis–angle representation (ϕ, θ, ψ) and assemble an 8D state vector

$$s_t = [x, y, z, \phi, \theta, \psi, g_1, g_2] \in \mathbb{R}^8.$$

To ensure compatibility with the pretrained SmolVLA distribution, we normalize this vector using dataset-level statistics from the pretraining dataset:

$$s_t^{\text{norm}} = \frac{s_t - \mu_{\text{state}}}{\sigma_{\text{state}} + \epsilon},$$

where μ_{state} and σ_{state} are defined by the SmolVLA pretraining dataset. Each episode also contains a natural-language task description, which we preprocess by appending a newline (expected by the tokenizer) and tokenizing with the SmolVLM2 tokenizer to obtain a sequence of token indices and a corresponding attention mask $\ell \in \mathbb{N}^L, m \in \{0, 1\}^L$. Sequences are padded or truncated to the maximum token length specified by the SmolVLA configuration. Finally, all processed components are assembled into a single structured input matching the format required by the SmolVLA policy, $(I_t^{\text{front}},\ I_t^{\text{wrist}},\ s_t^{\text{norm}},\ \ell, m)$, ensuring consistency with the expectations of the pretrained model and allowing direct integration with our GRPO reinforcement learning pipeline.

This set up allows the policy to act in a true online fashion. At each rollout timestep, the policy is given the environmental observations, an action is sampled from its outputted distribution, and that action is used to update the environment and retrieve the next timestep's observation.

5.3 Group Relative Policy Optimization (GRPO)

To fine-tune SmolVLA with reinforcement learning, we employ Group Relative Policy Optimization (GRPO), a policy gradient method and more lightweight variant of PPO. For a given task, GRPO operates by generating G rollouts using the current policy π_θ, computing group-wise advantages by normalizing rewards across the G rollouts, then applying a clipped policy-gradient update based on the ratio of rollout probability between π_θ and a frozen model π_{old}.

Each GRPO iteration, we first sample a random task within the LIBERO Long task suite and a random initialization of that task. We then run our G rollouts, determine the probability of each according to π_θ and π_{old} using their outputted distributions, and run the GRPO update.

The reward of each rollout is simply 1 if the trajectory succeeds and 0 otherwise. A rollout does not succeed if the task's goal state is not achieved within a specified maximum number of steps, a hyperparameter we set to 520.

GRPO computes a group-relative advantage for each trajectory:

$$A_i = \frac{R_i - \bar{R}}{\sigma_R + \epsilon}, \qquad \bar{R} = \frac{1}{G}\sum_{j=1}^{G} R_j,$$

where σ_R is the standard deviation across the group. This normalizes trajectory rewards and stabilizes updates while avoiding the need for a learned value function. The policy is then updated using the clipped importance ratio

$$r_i = \exp\Big(\log \pi_\theta(a_i \mid o_i) - \log \pi_{\theta_{\text{old}}}(a_i \mid o_i)\Big),$$

and the GRPO loss is

$$\mathcal{L}_{\text{GRPO}} = -\frac{1}{GH}\sum_{i=1}^{G}\sum_{t=1}^{H} \min\big(r_{i,t}A_i,\ \text{clip}(r_{i,t}, 1-\epsilon, 1+\epsilon)\, A_i\big),$$

For each task, after a certain number of backpropagation updates, defined by another hyperparameter, the new π_θ is updated to be π_{old} and the process repeats on a new task.

An important implementation detail is that we were very limited by GPU memory. We resolved this issue by chunking rollouts into chunks of C timesteps and calculating the GRPO loss function for one chunk at a time. The gradients accumulated across all the chunks would then be used in the optimization step, achieving the same result while using less memory but more time.

We choose to exclude the KL divergence penalty term sometimes used in GRPO for simplicity and because the model already performs poorly on many tasks in our target suite.

6 Results

6.1 Reinforcement Learning

We tune a small set of hyperparameters to enable GRPO fine-tuning under strict memory constraints. We ran multiple different training setups, and have results for 3 separate models. For each model, the rollout `GROUP_SIZE` is set to 4, meaning that advantages are computed over groups of four parallel trajectories; this stabilizes training while keeping memory usage manageable. We experimented with the number of `UPDATE_EPOCHS`, testing out 1, 2, and 4. The more update epochs, the more model updates per GRPO iteration. To further reduce memory during backpropagation, we process action chunks in segments of size `UPDATE_CHUNK_SIZE = 5`, ensuring that only a small portion of each trajectory is loaded into GPU memory at a time. We also inject stochasticity into the flow-matching denoising process using an Euler-step noise magnitude, testing out `EULER_STEP_NOISE_STD` values of 0.2 and 0.3, which improves exploration. We evaluated the pretrained SmolVLA policy on the LIBERO-10 benchmark (5 episodes per task, 10 selected tasks; 50 rollouts). The original model achieved 9 successes out of 50 episodes (**18%** success rate), performing well on tasks where pretrained behavior already matched the required policy but failing almost all tasks requiring new behaviors. This motivates the need for RL fine-tuning.

6.2 Reinforcement Learning Attempts

Fine-tuned Model 0.1: No Exploration With `EULER_STEP_NOISE_STD` $= 0$, the policy became fully deterministic. Advantages were nearly always zero, preventing GRPO from updating the model. No improvement over baseline was observed.

Fine-tuned Model 0.2: Moderate Exploration, High Memory Load Introducing exploration (`EULER_STEP_NOISE_STD` $= 0.2$) produced meaningful advantage signals. However, the configuration (`UPDATE_CHUNK_SIZE` $= 20$) exceeded GPU memory limits, causing out-of-memory (OOM) failures and preventing full training.

Fine-Tuned Model 1: Higher Exploration, Low update Epochs Using a smaller chunk size and increased exploration, `UPDATE_EPOCHS=1`, `UPDATE_CHUNK_SIZE=5`, `EULER_STEP_NOISE_STD=0.3`, the policy successfully completed 20 updates, 4 of which produced non-zero advantages. This demonstrates that GRPO can improve the policy when exploration is sufficient and memory constraints are controlled. Furthermore, GRPO does not require multiple update epochs, enabling faster updates. In total, these 20 updates took 12.26 hours, averaging at 0.613 hours per update step.

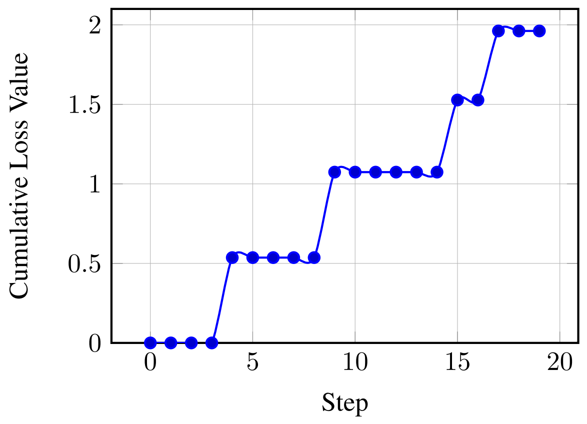

Figure 2: Cumulative Value over Training Steps for model 1.

Fine-Tuned Model 2: High Exploration, High Update Epochs Using a smaller chunk size and increased exploration, `UPDATE_EPOCHS`=4, `UPDATE_CHUNK_SIZE`=5, `EULER_STEP_NOISE_STD`=0.3, the policy successfully completed 40 updates, 3 of which produced non-zero advantages. With `UPDATE_EPOCHS`=4, the agent repeatedly optimized on the same batch of trajectories four times per GRPO update. This meant that a single task rollout exerted disproportionate influence on the gradient. In practice, this drove the model to overfit sharply to whichever task produced a non-zero advantage, effectively amplifying noise and destabilizing the policy. As seen in the graph, the model was not able to achieve a non-zero update after the last 30 steps, indicating a collapse of the model. Instead of learning generalizable action patterns across the task suite, the policy received too many gradient steps on too narrow a distribution, causing rapid drift in parameter space and eventual performance collapse.

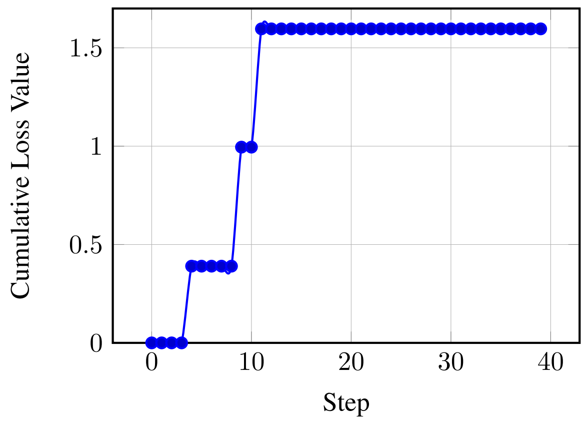

Figure 3: Cumulative loss over steps for model 2.

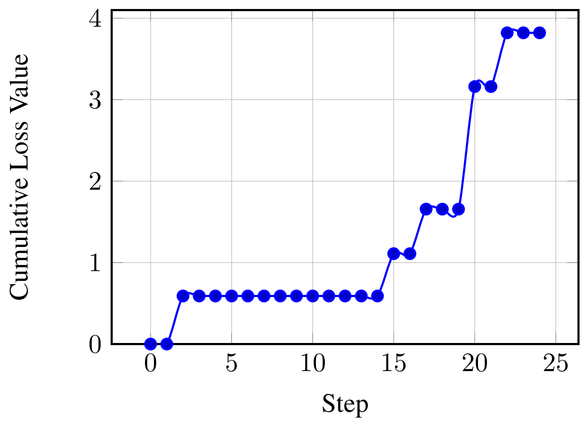

Figure 4: Cumulative loss for model 3.

Fine-Tuned Model 3: Medium Exploration, Medium Update Epochs Using a smaller chunk size and increased exploration, `UPDATE_EPOCHS`=2, `UPDATE_CHUNK_SIZE`=5, `EULER_STEP_NOISE_STD`=0.2, the policy successfully completed 25 updates, 5 of which produced non-zero advantages. After what we learned from applying 4 update epochs, we decided to decrease exploration slightly and only have 2 update epochs to ensure that policy was closer to the original model. This was successful as the model did not show any signs of plateuing even after 20 training steps. Furthermore, it allowed for much faster updates than fine-tuned model 1.

6.3 Quantitative Results

Task ID	Baseline (%)	FT 1 (%)	FT 3 (%)
0	**0.0**	**0.0**	**0.0**
1	**0.0**	**0.0**	**0.0**
2	**60**	**40**	**40**
3	**40**	**60**	**80**
4	**0.0**	**0.0**	**0.0**
5	**20**	**0.0**	**0.0**
6	**0.0**	**0.0**	**0.0**
7	**0.0**	**0.0**	**0.0**
8	**20**	**20**	**20**
9	**40**	**40**	**80**

Table 1: Baseline and fine-tuned success rates for LIBERO-Long tasks.

The primary metric we are measuring is **accuracy**, and the baseline achieved accuracy of 0.18, fine-tuned model 1 achieved accuracy of 0.16, and fine-tuned model 3 achieved accuracy of 0.22, an improvement from the baseline of 22%.

We noticed that there was not an improvement between the baseline and the first fine-tuned model. We believe that this is because we only ran the model for 20 updates with only 1 update epoch, and only 4 of those were non-zero, so this meant that the model was only updated 4 times. Because of that, we decided to increase update epochs to 2, and for our third fine-tuned model, we observed much improved performance, observing the only two 80% success rate values for a task out of all of our tests, and an overall improvement in accuracy.

7 Conclusion / Future Work

7.1 Discussion

Overall, while the models did not acquire the ability to solve entirely new tasks beyond the baseline capabilities, our results demonstrate that fine-tuning SmolVLA with GRPO can still meaningfully improve task performance even at a limited scale. Although Fine-Tuned Model 1 showed no clear gains-largely due to receiving only four effective updates-the more aggressive optimization settings used in Fine-Tuned Model 3 led to measurable improvements, including the only 80% success score observed across all experiments. Correspondingly, the overall accuracy increased from the baseline value of 0.18 to 0.22 in the fine-tuned models, indicating that the policy became better at the tasks it could already partially perform. These findings suggest that, despite the constraints of small update budgets and sparse learning signals, GRPO-based fine-tuning is a viable strategy for enhancing VLA performance. With more training iterations, broader task diversity, and improved exploration, we expect this approach to generalize more effectively and unlock further gains in long-horizon robotic manipulation tasks.

7.2 Limitations, Assumptions, and Future Work

Beyond compute and time limitations that made determining optimal hyperparameters or achieving large amounts of update steps difficult, there are other potential issues in our process that leaves room for future work.

First is our choice of how to convert the deterministic pretrained model into a stochastic policy. While treating the model's outputs as the parameters for a diagonal Gaussian is simple, it comes with a fundamental limitation. The standard deviation for these distributions must be narrow for the model to produce smooth, predictable movement. However, this also leads to less stochasticity, potentially limiting the effectiveness of methods like GRPO. It is also an assumption that the action space can even be modeled as a diagonal Gaussian.

Second is how to induce meaningful exploration in the model. We experimented with injecting noise into a variety of places in the inference pipeline and observed that, in general, noise injection does not lead to truly meaningful changes in trajectory. The model's trajectories may be slightly different, but there is no change in the high-level semantic strategy it seeks to employ. This means that it is difficult to force the model to try different high-level strategies which would drastically increase the effectiveness of RL methods. This is almost certainly the reason why our fine-tuning only improved performance on tasks the pretrained model was somewhat capable at before. If the model fails on a task entirely, there will never be a sparse reward signal to learn from for that task. We assumed that our noise injection led to some meaningful exploration, and this is somewhat vindicated by the noticeable improvement in certain tasks, but cannot be certain of this.

Finally, a major issue is the sparsity of our rewards. Trajectories are given a singular binary reward at the end depending on whether the task goal was acheived or not. This can lead to many updates where all advantages are zero and unstable training dynamics due to weak training signal. Defining a denser reward function that can reward individual timesteps may significantly improve this approach's effectiveness. We assumed that this sparse reward setup would be a sufficient starting point.

Future work should look both into alternative ways of converting models like SmolVLA into policies suitable for policy gradient methods and inducing meaningful exploration in such models. These are likely issues that would take entire projects in themselves to investigate but are crucial for improving performance on tasks the pretrained model fails at completely. In general, the problem of how to teach policy models to do long-horizon tasks is an open and difficult problem in robotics.

8 Contributions

Jason worked on a lot of the conceptual and theoretical issues about how to modify models and GRPO to this specific use case. He set up the project dependencies, modified the model architecture, worked on configuring LIBERO environments for online RL, and improved the basic GRPO training loop by adding features like vectorized inference and chunked backpropagation. He dealt with changing the source code for LIBERO and Lerobot.

Thomas worked on the details of how to preprocess raw observations into outputs that SmolVLA expects. He created a lot of the infrastructure for the initial GRPO training loop. He dealt with the actual training process, including tuning hyperparameters and dealing with checkpoints and metrics logging. He evaluated both the pretrained and fine-tuned models.

Both of us worked on this writeup.

References

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. π0. 5: a vision-language-action model with open-world generalization, 2025. *URL https://arxiv. org/abs/2504.16054*, 1(2):3.

Hengtao Li, Pengxiang Ding, Runze Suo, Yihao Wang, Zirui Ge, Dongyuan Zang, Kexian Yu, Mingyang Sun, Hongyin Zhang, Donglin Wang, et al. 2025. Vla-rft: Vision-language-action reinforcement fine-tuning with verified rewards in world simulators. *arXiv preprint arXiv:2510.00406*.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. 2023. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. 2025. Smolvla: A vision-language-action model for affordable and efficient robotics.