Steering Robots Safely: Using Mechanistic Interpretability to Analyze the Effects of RL Fine-Tuning in Vision-Language-Action Models

Team Members: Thomas Deng, Andrew Chen
Emails: tdeng23@stanford.edu, awche@stanford.edu

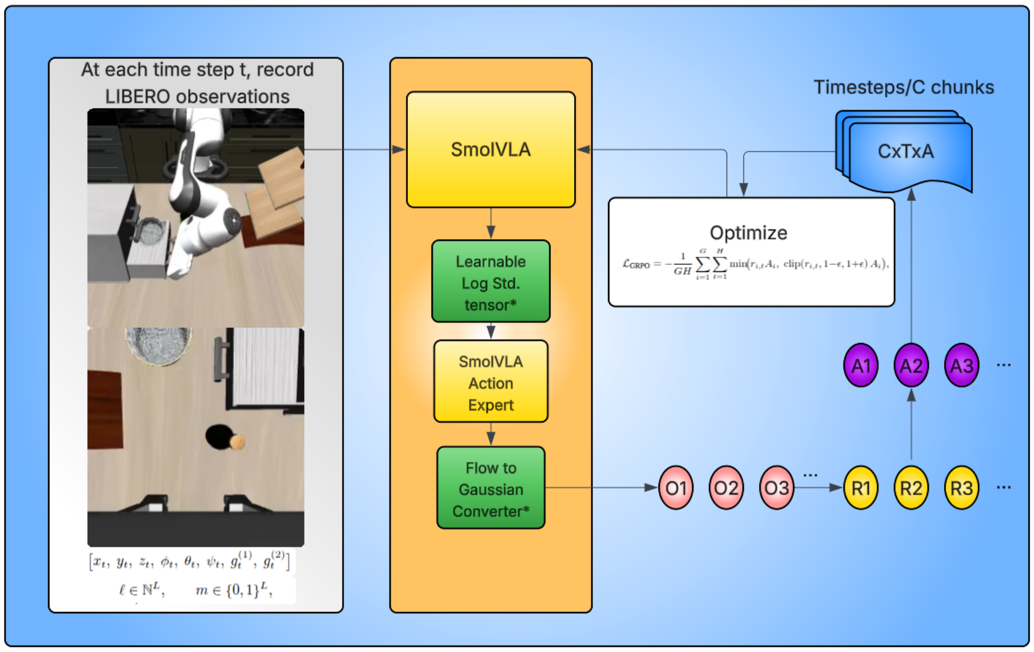
At each time step t, record LIBERO observations
$[x_t,\ y_t,\ z_t,\ \phi_t,\ \theta_t,\ \psi_t,\ g_t^{(1)},\ g_t^{(2)}]$
$\ell \in \mathbb{N}^L, \quad m \in \{0,1\}^L,$
SmolVLA
Learnable Log Std. tensor*
SmolVLA Action Expert
Flow to Gaussian Converter*
Timesteps/C chunks
CxTxA
Optimize
$\mathcal{L}_{\text{GRPO}}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

Figure 1: Steering Robots Safely Architecture, *denotes novel method

1 Introduction

A quickly developing paradigm in robotics is represented by VLA models, such as the commercial model π_0 by Intelligence et al., which combine perception, language, and control into a single multimodal framework that can zero-shot generalize to new tasks. 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 in out-of-distribution or out-of-context situations—a key issue for real-world robotics applications where safety is essential. We seek the use of Reinforcement Learning fine-tuning in order to suppress cascading failures in long-horizon tasks, which primarily occur under traditional imitation learning methods.

Since we fine-tuned SmolVLA, we generated rollouts for SmolVLA. At each timestep t, SmolVLA receives the inputs detailed in the graph above (further information can be found in section 3.1). At each timestep t, SmolVLA outputs a normalized action vector $a_t \in \mathbb{R}^9$, where each action corresponds to a 7-DoF end-effector pose update and a 2-DoF gripper action. With these inputs and outputs in mind, we implemented a sparse-reward GRPO model, that utilizes rollouts to perform group updates over the original SmolVLA policy. Our motivation is to not only build upon this reinforcement learning methods, but to also interpret it. In the case of our RFT method, we aim to fine-tune SmolVLA on the LIBERO Long suite of tasks, the LIBERO suite that VLAs universally perform the worst on, due to the presence of the titular long-horizon tasks.

To better analyze and eventually control how reinforcement learning reshapes such large multimodal policies, we apply mechanistic interpretability to SmolVLA Templeton et al. (2024). In particular, we build on the precedent established by Häon et al. (2025), who demonstrated that VLA models such as π_0 and OpenVLA contain semantically interpretable and steerable value vectors within their feed-forward (FFN) submodules. However, the effects of reinforcement fine-tuning on these internal representations remain largely unexplored.

Our decision in choosing SmolVLA presents a unique architectural challenge, as π_0 and OpenVLA are token-based VLA models that generate discrete action tokens, while SmolVLA employs a fundamentally different second-generation VLA architecture based on a continuous flow-matching action expert. This architectural difference makes SmolVLA particularly unique for reinforcement fine-tuning: before we can meaningfully study how GRPO updates reshape its internal representations, we must first establish whether the base policy itself contains semantically meaningful and causally steerable directions that can lead to safe steering in RL fine-tuned VLA models with flow-matching models.

This project spans two classes, CS229 and CS238. For CS229, our work was primarily focused on developing the theory and implementing the new methodologies of our GRPO algorithm and incorporating mechanistic interpretability and activation steering. For CS238, the focus was on building infrastructure for performing GRPO RL, in addition to hyperparameter tuning and optimization of our RL algorithm.

2 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 smaller models such as SmolVLA by adopting a sparse reward structure that produces feedback only on clear successes or failures, thereby reducing computation while still enabling effective reinforcement fine-tuning.

We aim to build on these models that leverage actor-critic architectures that could perform under hardware constraints, as in GRPO Shao et al. (2024), this 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 learned value function by computing group advantages. Our first goal is therefore to extend this line of work by applying actor-critic reinforcement learning to fine-tune SmolVLA on out-of-distribution robotic tasks long-horizon tasks, benchmarking in simulation with LIBERO. Yet, as these models become increasingly complex and are fine-tuned with reinforcement learning, a key question arises: how do such updates alter the internal representations and mechanisms of VLA policies? To investigate this, we draw inspiration from Häon et al. (2025), who introduced mechanistic interpretability to VLA systems by identifying semantically meaningful activation directions (e.g., "fast," "slow") within feed-forward layers that can be manipulated to steer robot behavior at inference time.

By combining reinforcement learning fine-tuning, we aim to assess whether VLA policies with flow-matching models provide stable, semantically meaningful control directions that can support safe and transparent reinforcement fine-tuning–laying the groundwork for future studies of RL-induced representational drift.

3 Dataset and Features

3.1 Reinforcement Learning

Since in reinforcement learning, since GRPO is an online learning method, are training examples come in the form of simulated rollouts. To convert raw LIBERO observations into inputs compatible with SmolVLA, we created novel infrastructure that adapted existing VLA infrastructure for our sole purpose of performing rollouts. We preprocess each timestep 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:

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

where μ_{state} and σ_{state} are extracted from the SmolVLA normalizer. 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 $\ell \in \mathbb{N}^L, m \in \{0, 1\}^L$. These 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. To apply GRPO to a model originally trained with a flow-matching action expert, we implemented a novel flow-matching output to probability distribution method inspired by to convert the model's continuous action predictions into a valid probability distribution (see Appendix 7.1). At each timestep, the pretrained SmolVLA action head produces a normalized action $\hat{a}_t$, which we treat as the mean μ_t of a Gaussian policy. A learnable log-standard-deviation vector $\log \sigma_t$ is shared across actions and updated during GRPO. To sample an action, we draw

$u_t \sim \mathcal{N}(\mu_t, \sigma_t^2)$ and apply a tanh squashing transform to ensure actions remain in $[-1, 1]$. Following standard practice for Gaussian policies, we compute the corrected log-probability of the action, which is required for the GRPO probability ratio (see Appendix 7.1).

Furthermore, since at each timestep we accumulate new probability distributions that we must take compute backpropagation for, one of the bottlenecks of our initial training method of inputting a was that we were limited to only 24 gigabytes of GPU RAM on one GPU. This meant that when we input a batch of G × timesteps × actions into the ˜450 million parameter model, GPU memory would run out very quickly. We resolved this issue by chunking our bataches into chunks of C, where C would divide the number of timesteps. G × C × actions. Thus, we would be saving memory by magnitudes of $\frac{\text{timesteps}}{C}$.

3.2 Mechanistic Interpretability

Mechanistic interpretability was analyzed in context of the SmolVLA-LIBERO pretrained model, as located on the HuggingFace distribution hub Cadene et al. (2024). To visualize and quantify steering, the steered SmolVLA policy was tested on one specific task in the LIBERO Long suite (Task 3), consistent with our RL fine-tuning set up, as well as the steering methodology described by Häon et al. (2025). For quantitative evaluation of steering effects, we focus on the 3D end-effector position $\mathbf{p}_t = (x_t, y_t, z_t)$ and use changes in this position over time to measure robotic motion magnitude and speed. We use the pretrained SmolVLA policy in evaluation mode, to ensure deterministic activations, and do not apply any additional data augmentation or normalization.

4 Methods

4.1 Group Relative Policy Optimization (GRPO) Rollouts

To fine-tune SmolVLA with reinforcement learning, we employ Group Relative Policy Optimization (GRPO), a lightweight variant of PPO designed for sequence models and large policies. GRPO operates by generating G rollouts using the current policy π_θ, computing group-wise advantages using π_{old}, and applying a clipped policy-gradient update. The rollout procedure is as follows. In the case of our implementation, each batch of groups is given the same task prompt. To enable stochastic rollouts, we treat the flow-matching output as the mean of a Gaussian policy and sample an unsquashed action

$$u_t \sim \mathcal{N}(\mu_t, \sigma^2), \qquad a_t = \tanh(u_t),$$

where μ_t is the flow-matching prediction and σ is a learned scale parameter. The log-probability of a_t is computed using the standard squashed Gaussian correction. A rollout of horizon H consists of the sequence $\big(o_0, a_0, r_0,\ o_1, a_1, r_1,\ \ldots,\ o_{H-1}, a_{H-1}, r_{H-1}\big)$,

and we collect G such trajectories in parallel, forming a *group*. Since our tasks use sparse binary rewards, the return of each trajectory is simply 1 if the trajectory succeeds and 0 otherwise. 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

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

mirroring PPO but using trajectory-level advantages instead of timestep-level estimates. After each update, θ_{old} is replaced with the updated policy parameters and new rollout groups are generated. This procedure integrates seamlessly with SmolVLA's chunked action generation and allows us to fine-tune the model using only sparse binary feedback, without the need for dense reward shaping or a learned critic.

4.2 Mechanistic Interpretability

4.2.1 FFN Value Vector Extraction

To analyze neuron-level semantics within SmolVLM2, we follow the value-vector framework introduced by Häon et al. (2025). In each transformer block, the feed-forward network (FFN) contains a down-projection matrix $W_{\text{down}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$. Each column of $W_{\text{down}}^{\top}$ defines a value vector corresponding to an individual hidden neuron. For a given layer ℓ, we extract all value vectors $v_i^{(\ell)} \in \mathbb{R}^{d_{\text{model}}}$ and project them into token space using the model's unembedding matrix $U \in \mathbb{R}^{|\mathcal{V}| \times d_{\text{model}}}$:

$$z_i = U v_i^{(\ell)}$$

where $|\mathcal{V}| = 49280$ defines the vocabulary space for SmolVLM2, and $d_{model} = 960$ as the dimensionality of the model's hidden embeddings. The resulting logit vector z_i defines a distribution over vocabulary tokens, allowing each neuron to be associated with the top-k tokens it most strongly activates.

4.2.2 Semantic Embedding

Grouping by logit value, we first extract the top-k tokens ($k = 30$) for each FFN value vector using its projected logit distribution $z_i \in \mathbb{R}^{|\mathcal{V}|}$. These k tokens are used to form semantic embeddings $e_i \in \mathbb{R}^{d_{\text{model}}}$:

$$e_i = \sum_{j=1}^{k} \text{softmax}(z_i)_j \, U_{T_i(j)},$$

which are softmax-weighted averages of the corresponding token embedding vectors. Here, index i refers to the FFN neuron (value vector) index within a single layer. We then ℓ_2-normalize these semantic embeddings in order to allow for cosine similarity calculations between neurons.

4.2.3 Neuron Clustering

To generate clusters of semantically related FFN neurons, we apply k-nearest neighbors (KNN) in the neuron semantic embedding space using the `scikit-learn` library.. Given a specific neuron i, we compute cosine distances between e_i and all other neuron embeddings $\{e_j\}_{j=1}^{d_{\text{ff}}}$ and retrieve the k nearest neighbors using the KNN algorithm. We apply this procedure to neurons whose top-k tokens contain control-related lexemes (e.g., *fast*, *slow*) in order to form clusters of neurons aligned with speed-related control concepts. These neurons within these clusters are targeted during activation steering.

4.2.4 Activation Steering

To steer, we apply a forward hook at the FFN down-projection layer. We overwrite all the activations for an associated cluster S, replacing them with α prior to the down-projection:

$$\tilde{f}_{\theta}^{(i)}(x) = \begin{cases} \alpha, & \text{if } i \in \mathcal{S}, \\ [f_{\theta}(x)]_i\,, & \text{otherwise,} \end{cases}$$

This intervention is applied at layer 16, unless otherwise stated, the layer whose hidden embeddings are the direct inputs to the SmolVLA action expert. This ensures that steering directly affects the features used for continuous action generation.

4.2.5 Semantic Identification of Control Directions

We focus on two control concepts, *fast* and *slow*. Using the workflow described in 4.2.1 to 4.2.3, we then list value vectors whose top-30 activated tokens contain the words "fast" and "slow". A value vector was manually searched and selected if least 4 out of 30 tokens displayed semantic correlations.

Vector	**Top Tokens**
126	location, sooner, least, cious, status, easiest, shortest, Multiple, quicker, Least, faster, quick, ... , rated
2253	aways, "|')", Dover, way, reset, hypothetical, thus, toget, Ø^{3}, slowly, rovers, boarding, urethane, ... , drawal

Figure 2: Example FFN value vectors and their top tokens for *fast* (126) and *slow* (2253) interventions.

We see the existence of a "fast" value vector, providing tokens such as ['sooner', 'faster', 'quick', 'quicker', 'easiest', 'least', 'shortest'...], while the "slow" value vector lacks semantic meaning and themes. Thus, we only formed a cluster around the "fast" FFN value vector and applied the steering method as described in 4.2.4.

5 Results and Discussion

5.1 Reinforcement Learning

We tune a small set of hyperparameters to enable GRPO fine-tuning under strict memory constraints. 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. The number of policy update passes per batch is reduced to `UPDATE_EPOCHS = 1`, which significantly lowers GPU memory requirements while still allowing effective policy improvement. To further reduce memory during backpropagation, we process action chunks in segments of size `UPDATE_CHUNK_SIZE = 10`, 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 of `EULER_STEP_NOISE_STD = 0.3`, which improves exploration (see appendix). We evaluated the pretrained SmolVLA policy on the LIBERO-10 benchmark (5 episodes per task, 8 selected tasks; 40 rollouts). The model achieved 6 successes out of 40 episodes (**15%** 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.

5.2 Reinforcement Learning Attempts

Model 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.

Model 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.

Model 3: Higher Exploration, Reduced Memory Footprint Using a smaller chunk size and increased exploration, `UPDATE_EPOCHS=1`, `UPDATE_CHUNK_SIZE=10`, `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.

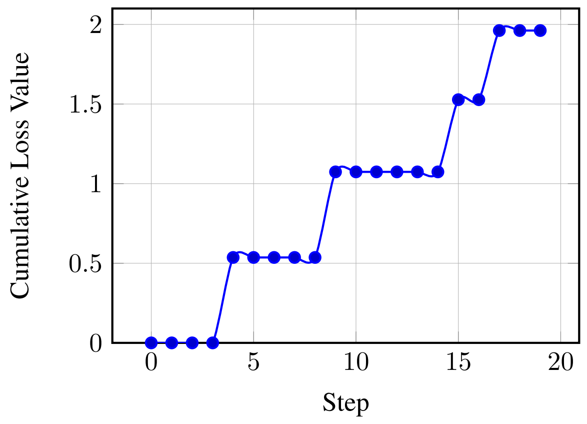

Figure 3: Cumulative Value over Training Steps.

Running the third model on the LIBERO Long task suite, we get the following results.

5.3 Quantitative Results

Task Suite	Task ID	Baseline (%)	Fine-Tuned (%)
LIBERO Long	0	**0.0**	**0.0**
LIBERO Long	1	**0.0**	**0.0**
LIBERO Long	2	**60**	**40**
LIBERO Long	3	**40**	**60**
LIBERO Long	4	**0.0**	**0.0**
LIBERO Long	5	**20**	**0.0**
LIBERO Long	6	**0.0**	**0.0**
LIBERO Long	7	**0.0**	**0.0**

Table 1: Baseline and fine-tuned success rates for LIBERO-Long tasks (Task IDs 0–7).

The primary metric we are measuring is **accuracy**, and the baseline achieved accuracy of 0.15 and the fine-tuned model achieved accuracy of 0.125, representing an 16.67% decrease.

The primary causes for the lack of drastic improvement was the compute constraints and the low amount of updates. With only 4 actual updates performed over the course of 12 hours, the model was unable to make drastic improvements. Our data did not overfit on the training set since we were using randomly simulated roll outs that encouraged exploration, however, it also seemed to under fit simply because we did not train enough. In order to improve this, we would ideally run on a faster GPU and incorporate reward modeling to perform actual updates at every update instead of only when there are successes.

5.4 Steering Simulations

The *fast*-steered SmolVLA policy is tasked a single task in the LIBERO Long suite. For reproducibility, three rollouts are carried out for the task. Two hyperparameters are explored for their influence on steering behavior: (i) the steering activation strength α and (ii) the size of the semantically clustered neuron group.

The main metric used to evaluate the effect of semantic steering is the **mean end-effector (EE) displacement (MEED) per timestep**, which averages the displacement between action steps, or equivalently robot speed, calculated in cm. The end-effector refers to the device attached to the final joint of the robot, the arm executing manipulation actions. We examine two hyperparameters and their effects on MEED.

Cluster Size: Keeping the activation strength fixed at $\alpha = 10$, we do not see significant differences in MEED due to cluster size. For the baseline policy, the difference in the metric was around 0.060 cm, while for the fast-steered policy with $\alpha = 2$, the metric difference was around 0.059 cm.

Activation Strength: A series of activation strengths ($\alpha = 2, 4, 6, 8, 10$) were tested with a constant cluster size of 10. The MEED value is averaged over the 3 rollouts.

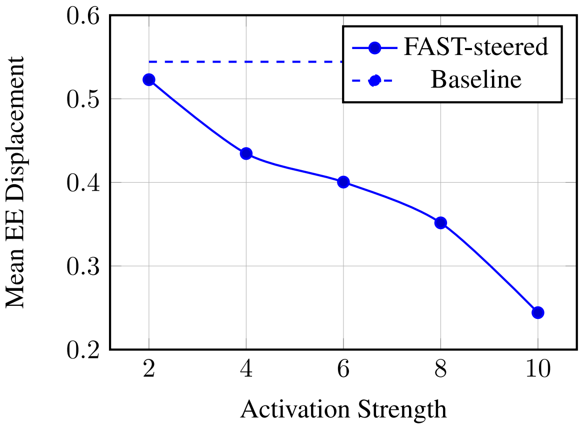

Figure 4: Mean end-effector displacement (cm) under FAST steering at varying activation strengths.

We see that as activation strength increases for the fast-steered SmolVLA policy, that **mean EE displacement decreases**. This contrasts with the findings from Häon et al. (2025), who showed that for fast-steered OpenVLA and π_0 policies, that MEED increases. If we correlate this metric with robot speed, Häon's findings imply a positive correlation between activating the "fast" value vectors and observing increases in robot speed, while we see a **strict negative correlation between activation strength and robot speed**. Additionally, at $\alpha = 10$, the policy consistently **failed to complete the task** within the maximum allowed action horizon, which was confirmed by final image states and rollout videos.

6 Conclusion / Future Work

While we had intended to incorporate mechanistic interpretability to analyze how GRPO fine-tuning reshapes the internal representations of SmolVLA, this investigation remains future work due to time constraints. GRPO demonstrated promising ability to maintain model performance while incorporating updates, but deeper training and hyperparameter tuning are required to fully understand its effects in the absence of a reward model. Our RL method was chiefly constrained by rollout throughput, which prevented us from evaluating a larger baseline sample. However, the fact that sparse rewards were consistently observed across the long-horizon task suite suggests that, with faster hardware, the model would likely exhibit further improvement even under sparse update regimes.

For the *fast* control concept, SmolVLA exhibits a strong negative correlation between steering strength and robot speed. At high activation ($\alpha = 10$) the policy consistently fails to complete the task, indicating that aggressive steering destabilizes performance under a continuous-action expert. While we find that interpretable control directions do exist in SmolVLA, their behavioral effects differ from token-based VLAs, where *fast* steering induces increased motion. These results establish an important baseline for reinforcement fine-tuning, showing that continuous-action VLAs while interpretable, have causal effects that may invert or destabilize under strong intervention.

7 Appendices

7.1 Converting Flow-Matching Outputs into GRPO-Compatible Probability Distributions

A common technique used to convert outputs into compatible PPO/GRPO probability distributions is to learn a log standard deviation tensor. The pretrained flow-matching action expert v_θ produces a deterministic action prediction

$$\hat{a}_t \in \mathbb{R}^A$$

representing the denoised action at timestep t. However, GRPO requires a stochastic policy $\pi_\theta(a_t \mid o_t)$ and, crucially, the ability to compute log-probability ratios between the updated and old policies. To reconcile these requirements, we model the policy as a diagonal Gaussian centered at the flow-matching output.

We define the Gaussian policy

$$\pi_\theta(u_t \mid o_t) = \mathcal{N}(u_t \mid \mu_t, \sigma^2),$$

where

$$\mu_t = \hat{a}_t, \qquad \sigma = \exp(\log \sigma),$$

and $\log \sigma$ is a learned vector shared across all timesteps and updated during GRPO. A raw (unsquashed) action sample is drawn as

$$u_t = \mu_t + \sigma \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

Because actions operate in the bounded interval $[-1, 1]$, we apply a $\tanh$ squashing transform:

$$a_t = \tanh(u_t).$$

The log-probability of the unsquashed action under the Gaussian is

$$\log \pi_\theta(u_t \mid o_t) = -\frac{1}{2} \sum_{i=1}^{A} \left[\frac{(u_{t,i} - \mu_{t,i})^2}{\sigma_i^2} + 2 \log \sigma_i + \log(2\pi) \right].$$

You can see our implementation in `model/smolvla_policy.py` .

7.2 Probability Ratios for GRPO

Given an old policy $\pi_{\theta_{\text{old}}}$, GRPO requires the probability ratio

$$r_t = \frac{\pi_\theta(a_t \mid o_t)}{\pi_{\theta_{\text{old}}}(a_t \mid o_t)}.$$

Using the log-probabilities derived above, we compute

$$r_t = \exp\left(\log \pi_\theta(a_t \mid o_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid o_t)\right).$$

This enables us to plug the flow-matching model into the GRPO objective without modifying its architecture, while providing the stochasticity required for reinforcement learning.

7.3 Encouraging Exploration

During inference, we introduce a small amount of noise into the Euler integration steps of the flow-matching denoising process. Specifically, at each timestep the predicted vector field v_t is perturbed by Gaussian noise before performing the update $x_{t+1} = x_t + dt\, v_t$. This converts the deterministic flow-matching ODE into a stochastic differential equation, causing different integration trajectories to produce slightly different final actions. As a result, the policy becomes stochastic even though the underlying flow-matching model is deterministic. This injected Euler-step noise provides a controlled source of exploration, ensuring that GRPO can compute well-defined probability ratios and preventing the policy from collapsing to a single deterministic action during fine-tuning.

8 Contributions

Thomas worked on GRPO RL fine-tuning and RL infrastructure setup. Andrew worked on mechanistic interpretability and steering studies on SmolVLA. (Note: we worked on the same lab workspace so our commits both show up on Tomocoder123).

References

Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Jess Moss, and Thomas Wolf. 2024. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. `https://github.com/huggingface/lerobot`.

Bear Häon, Kaylene Stocking, Ian Chuang, and Claire Tomlin. 2025. Mechanistic interpretability for steering vision-language-action models.

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*.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. 2024. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*.