

RoboIRS: Inference-Time Intervention on Robot Policies with Internal Representation Steering Vectors

Abstract—Vision-language-action (VLA) and world-action models (WAMs) often degrade under out-of-distribution task variations despite retaining partial task capability. To recover such capability, we propose RoboIRS, an inference-time internal representation steering method that uses successful and failed rollouts to train linear classifiers, select outcome-relevant intervention locations, and derive task-specific steering directions without updating policy parameters. On 15 simulation tasks with a frozen $\pi_{0.5}$ policy, RoboIRS improves the average success rate from 44.4% to 66.2%, outperforming alternative inference-time intervention baselines while adding little inference time. We further validate RoboIRS on real-robot manipulation using the same $\pi_{0.5}$ policy and demonstrate its applicability to a world-action model Cosmos Policy, where the average success rate improves from 35.4% to 55.4%. These results show that directly steering internal robot-policy representations can improve the performance of robot policies at inference time.

I. INTRODUCTION

Vision-language-action (VLA) models and world-action models (WAMs) have recently emerged as promising architectures for robotic manipulation by integrating visual perception, language-conditioned task understanding, and action generation with future prediction in a unified policy. However, their performance often degrades on out-of-distribution (OOD) inputs, such as unseen scenes, novel object layouts, and new object appearances [1], [2]. Importantly, this degradation does not necessarily imply complete incapability: a policy may still succeed on a subset of these OOD tasks, albeit at a lower success rate. This suggests that the policy may retain some of the internal computations required for successful execution even when its behavior becomes unreliable.

A similar phenomenon has already been observed in LLMs, which may produce both truthful and untruthful responses despite being queried under similar conditions. This inconsistency suggests that the model may already contain internal representations associated with the desired behavior, even when they are not reliably expressed in its outputs [3]. Recent progress in mechanistic interpretability and activation steering for large language models provides evidence that internal representations can be causally manipulated to alter model behavior. Here, internal representations refer to intermediate output that encode information during model computation; in transformer-based models, these may include residual-stream activations passed between layers, as well as activations associated with individual neurons, attention heads, or other localized model components [4]. Prior work has shown that high-level behaviors such as truthfulness, refusal, sentiment, and task execution can be localized in hidden representations and modified through activation-space interventions. Li et al. [3] steer LLMs toward truthful answers by intervening on truth-related activations, while Turner et al. [5] and Rimskey et

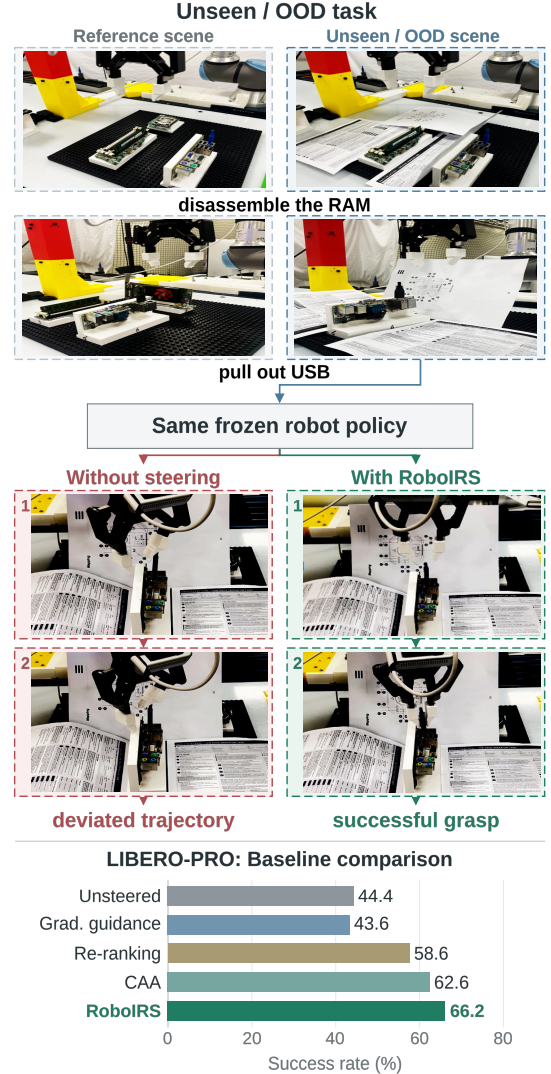


Fig. 1: Real-robot setup under visual distribution shift (top) and baseline comparison (bottom).

al. [6] show that contrastive activation directions can control generation behavior without retraining the model. Zou et al. [7] frame such methods as representation engineering, where internal activations are treated as controllable objects for modifying model behavior.

Motivated by the findings in LLM, can we similarly steer the internal representations of a robot policy at inference time to improve its performance on OOD tasks? Recent work has already extended mechanistic interpretability to vision-language-action (VLA) models. Grant et al. [8] and Swann et al. [9] use sparse autoencoders to identify interpretable

residual-stream features associated with robot behaviors, but do not demonstrate how these features can improve downstream policy performance. Häon et al. [10] and Buurmeijer et al. [11] go further by intervening on internal representations to steer behaviors such as motion speed and end-effector height; however, their evaluations are limited to a small set of predefined behavioral attributes rather than general task performance. Hong et al. [12] instead use an LQR controller to steer the diffusion transformer’s hidden activations such that the resulting activations lead to robust robot behaviors and require constructed contrastive residual-stream pairs from matched nominal and perturbed observations to provide target activation levels.

In this work, we aim to improve the performance of a frozen robot policy under distribution shift through inference-time steering of its internal representations. Rather than retraining the policy, we use these outcome-labeled activations to identify representations associated with successful and failed rollouts and intervene on the policy at inference time using a steering vector. The goal is to bias an already capable but unreliable policy toward internal states that are more predictive of successful action selection. Inspired by [3], we first train a linear classifier to classify the internal representations and then use the classifier weight as steering vectors to change the internal representations such that the predicted actions are more favorable of successful rollouts. We evaluate the proposed approach on two representative robot policies: the vision-language-action model $\pi_{0.5}$ [13] and the world-action model Cosmos Policy [14]. We conduct experiments in both simulation and real-world robotic settings to assess whether inference-time internal representation steering improves task success rates under distribution shift.

The main contributions of this work are as follows.

- We propose RoboIRS, which adapts the internal representation-steering paradigm from LLMs to robot policies and enables inference-time intervention on frozen policies using steering vectors derived from the internal representations of successful and failed rollouts.
- We conduct simulation and real-robot experiments showing that RoboIRS improves the average SR of a fully trained $\pi_{0.5}$ policy under distribution shift from 44.4% to 66.2%, outperforms alternative inference-time intervention baselines, improves real-robot manipulation performance, and is also effective on Cosmos Policy despite its different training recipe and model architecture, increasing the average SR from 35.4% to 55.4%.

II. RELATED WORK

A. Mechanistic Interpretability and Steering in LLMs

Mechanistic interpretability increasingly studies how high-level concepts are represented in neural activations and whether these representations can be causally manipulated [4], [15]. Many intervention methods build on the observation that properties such as truthfulness and sentiment can be represented by linear directions or low-dimensional subspaces [16], [17], [18]. Activation-steering methods exploit this structure by constructing directions from contrastive examples

or demonstrations and injecting them into hidden states to control behavior without retraining [5], [6], [19], [20]. Such interventions have been applied to truthfulness, hallucination, and refusal behavior [3], [21], [22], [23].

B. Mechanistic Interpretability and Internal Representation Steering of Robot Policy Models

Recent robotics work has begun to apply mechanistic interpretability techniques to characterize the internal representations of VLA policies, with several studies further showing that the discovered features can serve as causal handles for behavior steering. Khan et al. [24] use sparse autoencoders to identify action-related latent directions from contrastive action pairs and show that these directions can be manipulated to steer VLA action outputs without retraining. Grant et al. [8] conduct a cross-architecture mechanistic study using activation injection, sparse autoencoders, and linear probes to analyze how visual, language, and action-generation pathways contribute to VLA behavior. Swann et al. [9] use sparse autoencoders to identify interpretable features in VLA hidden states, distinguishing transferable motion primitives from episode-specific memorization, and causally validate selected features through steering experiments.

Several VLA papers move from feature discovery to explicit control of robot behaviors. Buurmeijer et al. [11] formalize feature observability and controllability in VLA models and use lightweight linear interventions to steer encoded features during closed-loop execution. Häon et al. [10] identify sparse semantic directions in transformer activations and use those directions to control policy behavior without retraining. Mitra et al. [25] identify task-relevant attention heads and selectively update them, framing policy adaptation as targeted modification of relevant internal mechanisms. Hong et al. [12] take a control-based approach to representation steering, constructing contrastive directions from paired nominal and perturbed inputs and using them to define robustness-relevant latent feature subspaces and target feature setpoints. RoboIRS instead derives its intervention directly from successful and failed rollouts without requiring matched nominal and perturbed observation pairs.

C. Inference-Time Intervention of Robot Policies

Another line of work improves robot policies at inference time through action selection or generative guidance. V-GPS [26] samples actions from a frozen policy and re-ranks them with an offline-RL value function, while RoboMonkey [27], MG-Select [28], and RoVer [29] select or refine VLA action candidates using learned verifiers, intrinsic confidence, or process rewards. Related methods intervene directly during action generation: value-guided diffusion [30] biases diffusion trajectories with value gradients. Guided Action Flow [31] keeps the VLA frozen and uses gradients from a critic trained on successful and failed rollouts to steer flow-matching action generation toward higher-return actions. PPGuide [32] trains a performance predictor and uses its guidance signal to steer pre-trained diffusion policies away from predicted failure modes during inference.

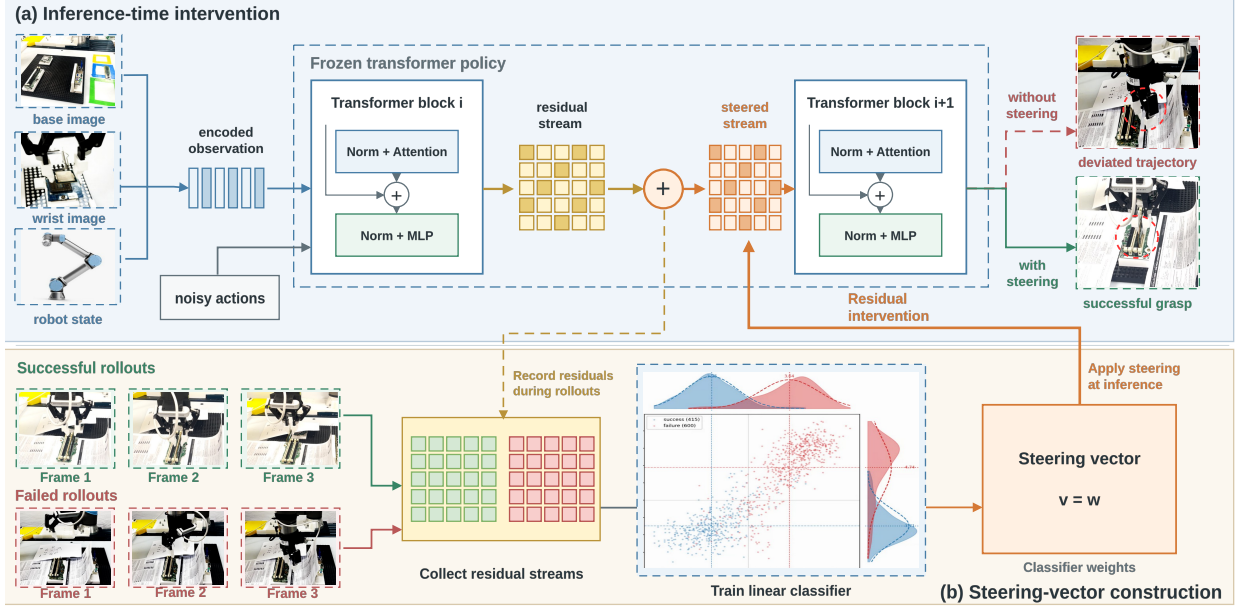


Fig. 2: Overview of the proposed RoboIRS framework. (a) RoboIRS intervenes between transformer blocks by adding a scaled steering vector to the selected residual-stream representation without updating the policy parameters. (b) To construct the steering vector, residual-stream representations are collected from successful and failed rollouts and used to train a linear classifier. The classifier weight provides the steering direction that is injected into the residual stream during inference.

III. PROPOSED APPROACH

A. Model Architecture and Internal Representation Definitions

To explain how steering is applied to internal representations, we first briefly describe the model architecture and define the notation used throughout this section. Since $\pi_{0.5}$ is the primary VLA model used in our experiments, we use it as a running example, although the formulation should apply more generally to transformer-based robot policies.

The $\pi_{0.5}$ VLA consists of a pretrained PaliGemma vision-language backbone and a smaller transformer-based action expert. The action expert, whose internal representations we steer, contains $L = 18$ transformer layers. Actions are generated using flow matching. At each environment timestep t , the policy starts from a noisy action chunk and iteratively refines it over N denoising steps. Let $\mathbf{X}_{t,n} \in \mathbb{R}^{T_a \times d_a}$ denote the noisy action chunk at denoising step n , where T_a is the action-chunk length and d_a is the dimensionality of one robot action. Each row $\mathbf{x}_{t,n,j} \in \mathbb{R}^{d_a}$ therefore corresponds to the j -th action in the chunk.

The action expert projects the action chunk into its hidden dimension d , producing T_a action tokens represented by $\mathbf{R}_{0,t,n} \in \mathbb{R}^{T_a \times d}$. For $\pi_{0.5}$, $d = 1024$. These tokens are processed through the L transformer blocks while conditioned on the VLM context, after which the action expert predicts the flow velocity used to update the action chunk.

Let $\mathbf{R}_{\ell,t,n} \in \mathbb{R}^{T_a \times d}$ denote the residual stream entering transformer layer ℓ . Omitting t and n for clarity, each transformer block updates the residual stream as

$$\mathbf{R}'_{\ell} = \mathbf{R}_{\ell} + \text{Attention}(\text{Norm}(\mathbf{R}_{\ell})), \quad (1)$$

followed by

$$\mathbf{R}_{\ell+1} = \mathbf{R}'_{\ell} + \text{MLP}(\text{Norm}(\mathbf{R}'_{\ell})). \quad (2)$$

The representation of action token j is denoted by $\mathbf{r}_{\ell,t,n,j} \in \mathbb{R}^d$, corresponding to the j -th row of $\mathbf{R}_{\ell,t,n}$. These token-level residual-stream representations are the intervention sites used for steering.

B. Steering Vector

Data Collection. Constructing the steering vectors requires residual streams from both successful and failed rollouts. For rollout i with T_i environment timesteps, we record $\mathbf{R}_{\ell,t,n}^{(i)}$ for every timestep t , transformer layer ℓ , and denoising step n . The resulting activation tensor is $\mathcal{R}^{(i)} \in \mathbb{R}^{T_i \times L \times N \times T_a \times d}$. For example, a 30-step rollout with $\pi_{0.5}$ produces a tensor of shape (30, 18, 10, 10, 1024), corresponding to 30 environment timesteps, 18 transformer layers, 10 denoising steps, 10 action tokens, and a hidden dimension of 1024.

We collect up to 10 successful and 10 failed rollouts per task to reduce underdetermination when fitting the linear classifier while keeping data-collection cost manageable. For tasks with very high or very low success rates, fewer examples of the minority outcome may be collected to avoid an excessive number of rollouts.

Contrastive Difference as Steering Vector. Inspired by representation-steering methods for LLMs [6], [3], we consider two approaches for constructing steering vectors. The first uses the mean contrastive difference between residual representations from successful and failed rollouts. For each layer ℓ , denoising step n , and action-token position j , we compute

$$\mathbf{v}_{\ell,n,j} = \frac{1}{|\mathcal{D}_s|} \sum_{(i,t) \in \mathcal{D}_s} \mathbf{r}_{\ell,t,n,j}^{(i)} - \frac{1}{|\mathcal{D}_f|} \sum_{(i,t) \in \mathcal{D}_f} \mathbf{r}_{\ell,t,n,j}^{(i)}, \quad (3)$$

where $\mathcal{D}_s$ and $\mathcal{D}_f$ denote the sets of timestep samples from successful and failed rollouts, respectively. Each steering vector $\mathbf{v}_{\ell,n,j} \in \mathbb{R}^d$ therefore corresponds to a specific transformer layer, denoising step, and action-token position.

Linear Classifier Weight as Steering Vector. The second approach uses the weight of a linear classifier as the steering direction, following analogous approaches in LLMs [3]. For each task, transformer layer, and action-token position, we train a linear classifier to distinguish success-like from failure-like residual streams. The residual stream’s label from each timestep is binary: all timesteps of successful rollouts are 0, and a failed rollout’s timesteps become 1 from the first step where its end-effector leaves the space swept by the successful trajectories.

The failure classifier is defined as

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b), \quad (4)$$

where $y = 1$ denotes failure, $\mathbf{w} \in \mathbb{R}^d$ is the classifier weight, and $b \in \mathbb{R}$ is the bias. We use $\mathbf{w}$ as the corresponding steering direction. Thus, each layer–token pair has its own classifier and steering vector. Although this requires training multiple classifiers per task, both the classifiers and datasets are small; in our experiments, all classifiers can be trained in under three minutes using multiple CPUs.

As noted in [8], linear probes trained on residual streams from the action expert achieve high area under the ROC curve (AUC) for predicting task success. Consistent with this finding, our classifiers also achieve good AUC when distinguishing representations from successful and failed rollouts. The average validation AUC is 0.88 for all trained classifiers.

C. Intervention

One limitation of using the mean contrastive difference alone is that it does not indicate which action-token position or transformer layer should be steered. One could perform a sweep over intervention sites and intervention strength, as in [6], to identify the most effective place for intervention. Such sweeps are relatively inexpensive for LLMs, where performance can often be evaluated with a single question–answer interaction. In robotics, however, each candidate typically requires multi-step policy inferences, making an exhaustive search over intervention sites prohibitively expensive. Instead, we use the linear classifier’s AUC on held-out validation rollouts to select the steering location. From the collected successful and failed rollouts, we reserve a subset from each outcome as validation data and use the remaining rollouts for classifier training. For each layer ℓ , we select the action-token position with the highest failure-detection AUC to steer. This choice is further explained in Section IV-B.

Once the steering location is determined, the selected residual stream at an action token is modified as

$$\mathbf{r}_{\text{steered}} = \mathbf{r} + s\alpha \frac{\|\mathbf{r}\|}{\|\mathbf{v}\|} \mathbf{v}, \quad s \in \{-1, +1\}, \quad (5)$$

where $\alpha \geq 0$ controls the steering strength and s determines the steering direction. Since the residual-stream magnitude varies across layers, we scale the intervention by $\frac{\|\mathbf{r}\|}{\|\mathbf{v}\|}$ so that the steering strength is proportional to the magnitude of the residual stream at each layer.

The sign of the intervention is selected empirically rather than assuming that the steering vector is to pull residual streams from failure-relevant space to success-relevant space. The linear representation hypothesis [16] suggests that high-level concepts can be represented as directions in a model’s representation space. From this view, we interpret $\mathbf{v}$ as a task-relevant latent direction rather than a direction pointing from failure relevant latent space to success relevant spaces. We hypothesize that successful behavior may require an appropriate level of the latent factor encoded by $\mathbf{v}$. For a task in which this factor is under-activated, steering along $+\mathbf{v}$ may improve performance, whereas a task in which it is over-activated may benefit from $-\mathbf{v}$. We therefore first evaluate one steering sign and test the opposite sign only when the sign shows a monotonic performance decrease as shown in Fig. 3, avoiding twice the rollout cost for every task. Importantly, a linearly discriminative direction need not be a causal intervention direction [16]. Therefore, particularly for the classifier-derived vectors, the direction may encode information that is predictive but not causally useful to change robot behaviors.

IV. RESULTS AND ANALYSIS

We structure our experiments around the following questions:

- **Q1:** Does steering the internal representations improve task success rates in both simulation and on the real-robot experiment?
- **Q2:** Does the proposed approach outperform alternative inference time intervention methods?
- **Q3:** Can the proposed framework be extended from a VLA model to a world action model?

Success rate (SR) is used as the primary evaluation metric. We use LIBERO-PRO [1] as our primary simulation benchmark, as it evaluates generalization to unseen task variations introduced through changes in object layout and appearance, background scene, and task instructions. For steering experiments, we use the $\pi_{0.5}$ VLA model fine-tuned on the original LIBERO dataset [33], which does not contain demonstrations corresponding to the perturbations introduced in LIBERO-PRO.

A. Effectiveness of Internal Representation Steering

Q1: Does steering the internal representations improve task success rates in both simulation and the real world?

We first evaluate the proposed method using the $\pi_{0.5}$ policy on LIBERO-PRO. We select 15 LIBERO-PRO tasks for which the unsteered policy achieves a non-saturated success rate between approximately 10% and 85%. This ensures that both successful and failed rollouts are available for identifying outcome-relevant internal representations. Table I reports the success rate of each task with and without steering. The *Suite* column denotes the original LIBERO task suite together with

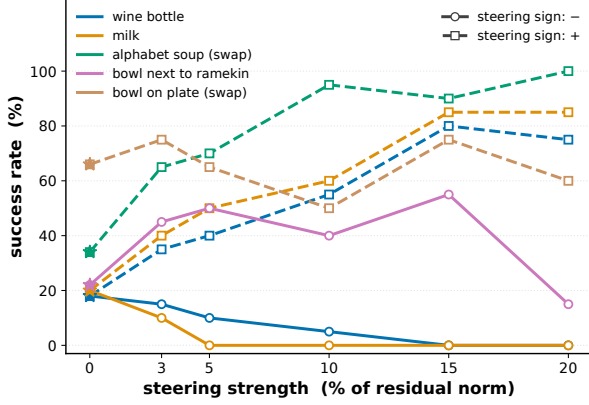


Fig. 3: Steering strength vs. SR from a few tasks to show the typical patterns: monotonic increase/decrease; increase at moderate strengths before degrading at larger strengths.

the LIBERO-PRO perturbation type, where *env* changes the environment or background scene and *swap* changes object positions to create an unseen layout. Although LIBERO-PRO also includes perturbations such as changes to object appearance and task instructions, we exclude those settings because the $\pi_{0.5}$ policy either fails almost completely or already achieves nearly 100% success, leaving little room to evaluate steering effectiveness.

For $\pi_{0.5}$, we apply steering only at the final denoising step and select the action-token position with the highest validation AUC within each transformer layer, as detailed in Section IV-B. We sweep each task on $\alpha = [0.03, 0.05, 0.10, 0.15, 0.20]$ for 20 rollouts on one random seed to identify the best steering strength and then use the best steering strength for evaluation for 150 rollouts across another 3 random seeds. In Fig. 3, we show how the SR changes along with different steering strength from a few representative tasks.

Overall, steering improves the average success rate from 44.4% to 66.2% on LIBERO-PRO tasks, corresponding to a gain of 21.8 percentage points. Among the 15 evaluated tasks, 12 tasks (80%) improve after steering, and all 12 satisfy the criterion that the steered mean SR exceeds the unsteered mean SR plus one standard deviation. The remaining three tasks do not show a clear improvement in mean SR: L5 and L12 decrease slightly, while L8 remains unchanged. However, the decreases for L5 and L12 remain within the standard-deviation range of the unsteered performance, suggesting that steering does not cause a substantial degradation on these tasks.

We additionally evaluate RoboIRS on two real-robot manipulation tasks, as illustrated in Fig. 1. In both tasks, RoboIRS improves the SR of a $\pi_{0.5}$ policy fine-tuned on demonstrations collected under clean-background conditions and evaluated under visually shifted settings.

B. Comparison with Other Baselines and Ablation Studies

Q2: Does the proposed approach outperform alternative inference-time intervention methods?

We compare the proposed approach against other inference-time intervention baselines: contrastive activation addition

TABLE I: Success rates (%) of unsteered and steered $\pi_{0.5}$. Bold indicates improvements larger than one unsteered standard deviation for LIBERO-PRO. The Sign column denotes the steering direction s . Parentheses in the real-robot rows denote successes/total trials.

ID	Suite	Task	Unsteered	Steered	Sign
L1	goal_env	Open top drawer; put bowl inside	12.0 $\pm$ 7.2	34.0 $\pm$ 2.0	-1
L2	goal_env	Put bowl on plate	32.7 $\pm$ 1.2	96.7 $\pm$ 3.1	-1
L3	goal_env	Put bowl on cabinet	72.0 $\pm$ 7.2	89.3 $\pm$ 2.3	-1
L4	goal_env	Put wine bottle on cabinet	17.3 $\pm$ 3.1	68.0 $\pm$ 2.0	+1
L5	goal_swap	Put bowl on plate	70.0 $\pm$ 4.0	66.7 $\pm$ 9.9	+1
L6	object_env	Alphabet soup to basket	59.3 $\pm$ 1.2	71.3 $\pm$ 1.2	-1
L7	object_env	BBQ sauce to basket	80.7 $\pm$ 1.2	95.3 $\pm$ 2.3	-1
L8	object_env	Ketchup to basket	84.7 $\pm$ 5.0	84.7 $\pm$ 2.3	-1
L9	object_env	Milk to basket	19.3 $\pm$ 1.2	72.7 $\pm$ 4.6	+1
L10	obj_swap	Alphabet soup to basket	39.3 $\pm$ 4.6	90.0 $\pm$ 2.0	+1
L11	obj_swap	Butter to basket	32.0 $\pm$ 5.3	43.3 $\pm$ 2.3	+1
L12	spatial_env	Bowl between plate/ramekin to plate	40.7 $\pm$ 4.2	36.7 $\pm$ 4.2	-1
L13	spatial_env	Bowl in top drawer to plate	25.3 $\pm$ 6.1	34.0 $\pm$ 3.5	+1
L14	spatial_env	Bowl next to plate to plate	58.7 $\pm$ 4.2	76.7 $\pm$ 3.1	-1
L15	spatial_env	Bowl next to ramekin to plate	22.0 $\pm$ 4.0	33.3 $\pm$ 5.8	-1
LIBERO-PRO avg.			44.4 $\pm$ 4.0	66.2 $\pm$ 3.4	--
H1	env	Disassemble RAM from motherboard	24.0 (12/50)	44.0 (22/50)	+1
H2	env	Pull out USB connector	12.0 (6/50)	32.0 (16/50)	+1
Real-robot avg.			18.0	38.0	--

TABLE II: Ablation study over steering denoise steps, vectors, and locations, ordered by average SR.

Steering Strategy	Avg. SR (%)
One Random Direction	44.0
Unsteered	44.4
Worst Token per Layer	52.7
Highest-AUC Layer	54.3
Random Token per Layer CAA	56.5
Random Token per Layer	57.3
All Denoise Steps	59.3
Highest-AUC Token per Layer	66.2

(CAA) [6], sample and re-rank (Re-rank), and gradient guidance on the flow velocity (Grad.).

This CAA baseline constructs the steering vector using the difference between the residual streams of a selected token from successful and failed rollouts. For a controlled comparison, the CAA baseline uses the same successful and failed rollout data and intervenes at the same model positions. Steering strengths are selected using the same validation protocol.

Following the same candidate generation and selection paradigm as prior sample and re-rank methods [26], [28], [27], we implement a re-ranking baseline for our setting. At each inference step, the observation is replicated across a batch of size k , so that a shared forward pass through the vision encoder and multimodal prefix produces k candidate action chunks that differ only in their flow-matching noise. Each candidate is scored by a learned linear classifier estimating its likelihood of leading to successful execution, and the highest-scoring candidate is executed. We evaluate $K \in \{8, 16\}$ and report $K = 16$, which achieves the higher average SR.

Following the same classifier-guided inference paradigm as prior gradient-guidance methods [31], we train a two-logit

classifier for each task to estimate $P(\text{success} \mid s, a)$ using the mean-pooled PaliGemma layer-17 prefix s and flattened action chunk a . At inference, the policy remains frozen and only the flow velocity is modified. At each Euler step, we compute a clean-action estimate $\hat{a} = x_t - t v_t$, then the flow velocity is modified as

$$v_{\text{guided}} = v_t - \alpha P(\text{failure} \mid s, \hat{a}) \frac{\|v_t\|}{\|d\|} d, \quad (6)$$

where $d \propto \nabla_a \log P(\text{success} \mid s, \hat{a})$. The negative sign is because $\pi_{0.5}$ integrates the flow backward in time ($dt < 0$) for the corresponding Euler position update moves in the $\nabla_a \log P(\text{success})$ direction. We sweep $\alpha \in \{0.02, 0.05, 0.1, 0.2\}$ and use the best value of each task for evaluation.

Table III reports the task-level comparison on the simulation benchmark. Again, each baseline is evaluated on each task for a total of 150 rollouts covering 3 random seeds. As shown in Table III, classifier-based steering (RoboIRS) achieves the highest average success rate of 66.2%, improving over the unsteered policy by 21.8 percentage points and outperforming other baselines. This improvement also incurs little additional inference cost in comparison with re-rank which increases the average inference time to 1074.86 ms, approximately 8 times that of other methods. Re-ranking outperforms RoboIRS on 7 of 15 tasks by an average margin of 9.7 percentage points, whereas RoboIRS outperforms re-ranking on the remaining 8 tasks by a substantially larger average margin of 22.7 percentage points. Gradient guidance barely helps in our case because it’s found that the classifier relies primarily on the encoded state, producing weak action-space gradients. For completeness, computing the guidance gradient with an action-only classifier yields an average SR of 43.5%. In contrast, our method remains effective with a simple linear classifier and does not require carefully designed gradient functions.

In addition, we conduct an ablation study to evaluate whether steering the highest-AUC action token in each transformer layer is beneficial for the $\pi_{0.5}$ model. We compare three intervention strategies: steering all action tokens in the single layer with the highest AUC at the final denoising step, steering the highest-AUC token in each layer across all denoising steps, and steering only the highest-AUC token in each layer at the final denoising step. As shown in Table II, steering the highest-AUC token in each layer only at the final denoising step achieves the best overall performance for $\pi_{0.5}$.

We also compare classifier-based steering with CAA applied using randomly selected action tokens. Although CAA can outperform probe-based steering on a few individual tasks, its average SR is lower when the intervention token is not selected using the classifier AUC, indicating that the AUC-based token selection could help. Besides, a random-direction control performs near the unsteered baseline, further supporting the importance of the classifier-derived steering direction.

C. Extension to a World Action Model

Q3: Can the proposed framework be extended to a world action model?

TABLE III: Comparison of task-level success rates (%) and average inference time among baselines and RoboIRS. Highest number is marked bold. The Win Margin rows report the average SR difference over the subset of tasks where either the corresponding baseline or RoboIRS performs better, with the number of winning tasks shown in parentheses.

ID	Unsteered	Grad.	Re-rank (K=16)	CAA	Ours
L1	12.0	12.7	44.7	34.0	34.0
L2	32.7	30.7	74.0	68.7	96.7
L3	72.0	58.7	79.3	75.3	89.3
L4	17.3	19.3	51.3	94.0	68.0
L5	70.0	63.3	70.7	85.3	66.7
L6	59.3	56.7	79.3	66.0	71.3
L7	80.7	83.3	98.7	95.3	95.3
L8	84.7	80.0	92.0	82.0	84.7
L9	19.3	20.7	32.0	16.0	72.7
L10	39.3	43.3	31.3	75.3	90.0
L11	32.0	34.7	21.3	59.3	43.3
L12	40.7	43.3	64.0	48.0	36.7
L13	25.3	29.3	32.0	36.0	34.0
L14	58.7	58.0	68.0	75.3	76.7
L15	22.0	20.7	40.7	28.7	33.3
Avg. SR	44.4	43.6	58.6	62.6	66.2
Win Margin (Baseline)	3.7 (2/15)	6.6 (1/15)	9.7 (7/15)	14.8 (5/15)	–
Win Margin (Ours)	27.8 (12/15)	24.6 (14/15)	22.7 (8/15)	15.9 (8/15)	–
Avg. Inference Time(ms)	134.0	137.9	1074.9	136.3	136.4

TABLE IV: Task-level success rates of the unsteered and steered Cosmos Policy. Improvements larger than one unsteered standard deviation are marked bold.

ID	Suite	Task	Unsteered	Steered
C1	goal_object	Put bowl on top of cabinet	36.7 ± 10.0	92.2 ± 5.1
C2	goal_swap	Open top drawer and put bowl inside	16.7 ± 3.3	14.4 ± 5.1
C3	object_env	Pick up alphabet soup and place in basket	17.8 ± 1.9	30.0 ± 3.3
C4	object_env	Pick up ketchup and place in basket	17.8 ± 1.9	24.4 ± 3.8
C5	spatial_swap	Pick up black bowl next to plate and place on plate	62.2 ± 3.8	97.8 ± 1.9
C6	goal_object	Open top drawer and put bowl inside	12.2 ± 1.9	27.8 ± 6.9
C7	object_env	Pick up BBQ sauce and place in basket	78.9 ± 6.9	84.4 ± 5.1
C8	object_object	Pick up alphabet soup and place in basket	41.1 ± 5.1	72.2 ± 1.9
Macro average			35.4	55.4

To determine whether the method is specific to the internal representations of $\pi_{0.5}$ policy, we additionally apply it to Cosmos Policy [14] which has a different model architecture (DiT) and a different training recipe. Eight tasks with unsaturated SR in LIBERO-PRO is selected for evaluation. We sweep the each task on $\alpha = [0.05, 0.10, 0.15, 0.30]$ for 10 rollouts on one random seed to identify the best steering strength per task. Then evaluate on a total of 90 rollouts over 3 random seeds. As shown in Table IV, steering improves the macro-average success rate of Cosmos Policy from 35.4% to 55.4%. Among the eight evaluated tasks, six achieve meaningfully higher success rates after steering.

However, Cosmos Policy differs substantially from $\pi_{0.5}$ in its action-generation mechanism. It does not use a separate action expert. Instead, it repeatedly tiles an action chunk into a latent frame and generates that frame with the video diffusion transformer. The action is decoded by averaging these repeated representations. In the LIBERO-PRO configuration, the action

frame contains 196 tokens. Steering a single token therefore affects only a small part of the action latent. We instead steer all 196 action-frame tokens in the final transformer block. The corresponding classifier is instead trained on mean-pooled residual streams over the 196 tokens. Because the classifier is trained in normalized residual-stream space, there are two choices of steering vector $\mathbf{w}/\sigma$ and $\mathbf{w} \odot \sigma$. Interpreting $\mathbf{w}$ as the classifier normal gives $\mathbf{w}/\sigma$, which corresponds to the gradient of the classifier score with respect to the raw residual. Alternatively, interpreting $\mathbf{w}$ as a displacement direction in normalized representation space gives $\mathbf{w} \odot \sigma$ after mapping the displacement back to the raw residual coordinates. Empirically, for Cosmos Policy, $\mathbf{w}/\sigma$ performs better on tasks C1–C5, whereas $\mathbf{w} \odot \sigma$ performs better on C6–C8. Overall, these results show that RoboIRS transfers beyond $\pi_{0.5}$ to a policy with a different architecture and training recipe, while requiring architecture-specific choices for classifier and intervention sites.

D. Observations and Analysis

In this subsection, we share a few observations and analysis by answering the following questions.

How do residual-stream representations change before and after steering? Steering tends to make the residual representations more locally concentrated for most tasks, although this effect is not consistent across all tasks. We visualize the residual streams from unsteered and steered rollouts using t-SNE [34]. As shown in Fig. 4(a), for the strongly improved L10 task, the steered residuals form tighter local clusters while remaining globally distributed over a similar region as the unsteered residuals. In Fig. 4(b), however, some steered residuals from failed L8 rollouts appear in regions occupied by successful rollouts but not by unsteered failed rollouts, even though steering does not improve the task-level SR. This might suggest that a direction predictive of success and failure may recover some success-like residual representations without necessarily providing a causally effective steering direction for improving task SR.

What types of failure modes are corrected by steering?

We observe two broad categories of failure modes, consistent with the failure taxonomy discussed in Sentinel [35]. The first is dominated by decision-making errors, including L3, L6, L9, L10, L11, L14, and L15, where the policy often makes an incorrect high-level choice despite relatively accurate low-level execution. For example, in L10, the unsteered policy frequently selects butter instead of alphabet soup after the object-position swap, whereas the steered policy usually selects the correct object. Across these tasks, RoboIRS improves the average SR by 24.9%.

The second category is dominated by action-precision errors, including L1, L2, L4, L5, L7, L8, L12, and L13. Typical failures include inaccurate grasps that cause objects to slip, repeated unsuccessful execution leading to timeout, or the gripper becoming stuck on the drawer handle during opening. For these tasks, the average SR improves by 19.1%.

For the three tasks where steering does not improve performance, the corresponding classifiers also have lower validation

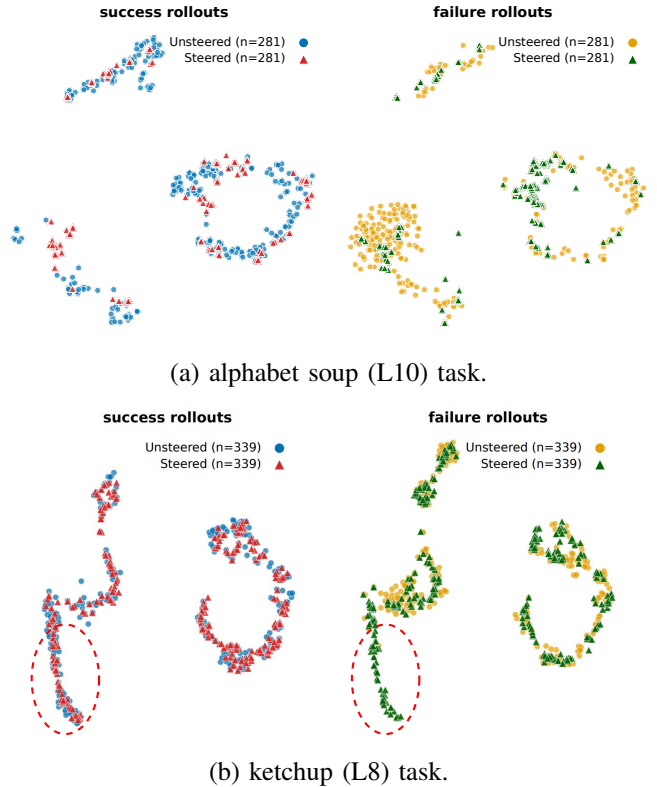


Fig. 4: t-SNE visualization of the last-layer’s residuals. Each panel compares residual representations from unsteered and steered policies separately for successful and failed rollouts.

AUCs: 0.701 for L5, 0.827 for L8, and 0.820 for L12, compared with an average AUC of 0.90 across the selected steering locations for all tasks.

Can a steering direction transfer across perturbations of the same task? Not necessarily, transfer appears to depend on whether the perturbations induce similar failure reasons. For “Put the bowl on the plate,” transferring the L2 steering vector to L5 improves SR from 62% to 86% on the same evaluation seed. Both tasks commonly fail by grasping the bowl inaccurately. In contrast, transferring the L10 vector to L6 for “Pick up the alphabet soup and place it in the basket” reduces SR to 46%. In L10, the robot mainly confuses alphabet soup with butter, whereas it confuses it with salad dressing in L6, suggesting that the learned directions may encode different failure-specific concepts.

V. CONCLUSION AND LIMITATIONS

Conclusion. We propose RoboIRS, an inference-time representation-steering method that uses successful and failed rollouts to identify and manipulate outcome-relevant internal representations without updating policy parameters. RoboIRS improves $\pi_{0.5}$ performance on LIBERO-PRO from 44.4% to 66.2%, also improves real-robot manipulation, and transfers to Cosmos Policy, where success rate increases from 35.4% to 55.4%. These results show that internal representations of frozen robot policies can be directly steered to improve task performance at inference time.

Limitations. The proposed method may be less effective on tasks where successful and failed rollouts do not exhibit clear differences in their actions, making it difficult to label and train a classifier. Our experiments are also limited to a small number of VLA and WAM policy architectures, as the primary goal of this work is to investigate the feasibility of transferring representation-steering techniques from LLMs to robotic policies. Different model architectures may encode task-relevant information differently and may therefore require different intervention locations, steering vectors, or steering strategies. Finally, like related representation-steering methods for LLMs, RoboIRS requires empirical selection of hyperparameters such as steering strength, introducing additional cost before deployment.

ACKNOWLEDGMENTS

As listed in submission requirements, AI tools like codex and claude are used to generate code for experiments and implementation which are reviewed by authors.

REFERENCES

- [1] X. Zhou, Y. Xu, G. Tie, Y. Chen, G. Zhang, D. Chu, P. Zhou, and L. Sun, “Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization,” *arXiv preprint arXiv:2510.03827*, 2025.
- [2] S. Fei, S. Wang, J. Shi, Z. Dai, J. Cai, P. Qian, L. Ji, X. He, S. Zhang, Z. Fei *et al.*, “Libero-plus: A progressive robustness benchmark for visual-language-action models,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026, pp. 38 574–38 583.
- [3] K. Li, O. Patel, F. Viégas, H. Pfister, and M. Wattenberg, “Inference-time intervention: Eliciting truthful answers from a language model,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 41 451–41 530, 2023.
- [4] H. Zhang, Z. Zhang, M. Wang, Z. Su, Y. Wang, Q. Wang, S. Yuan, E. Nie, X. Duan, Q. Xue *et al.*, “Locate, steer, and improve: A practical survey of actionable mechanistic interpretability in large language models,” *Computer Science Review*, vol. 62, p. 101011, 2026.
- [5] A. M. Turner, L. Thiergart, G. Leech, D. Udell, J. J. Vazquez, U. Mini, and M. MacDiarmid, “Steering language models with activation engineering,” *arXiv preprint arXiv:2308.10248*, 2023.
- [6] N. Rimskey, N. Gabrieli, J. Schulz, M. Tong, E. Hubinger, and A. Turner, “Steering llama 2 via contrastive activation addition,” in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 15 504–15 522.
- [7] A. Zou, L. Phan, S. Chen, J. Campbell, P. Guo, R. Ren, A. Pan, X. Yin, M. Mazeika, A.-K. Dombrowski *et al.*, “Representation engineering: A top-down approach to ai transparency,” *arXiv preprint arXiv:2310.01405*, 2023.
- [8] B. Grant, X. Zhao, and P. Wang, “Not all features are created equal: A mechanistic study of vision-language-action models,” *arXiv preprint arXiv:2603.19233*, 2026.
- [9] A. Swann, L. McGranahan, H. Buurmeijer, M. Kennedy III, and M. Schwager, “Sparse autoencoders reveal interpretable and steerable features in vla models,” *arXiv preprint arXiv:2603.19183*, 2026.
- [10] B. Häon, K. Stocking, I. Chuang, and C. Tomlin, “Mechanistic interpretability for steering vision-language-action models,” *arXiv preprint arXiv:2509.00328*, 2025.
- [11] H. Buurmeijer, C. A. Alonso, A. Swann, and M. Pavone, “Observing and controlling features in vision-language-action models,” *arXiv preprint arXiv:2603.05487*, 2026.
- [12] J. Hong, J. Skifstad, Q. Dai, A. Chan, and G. Chou, “Steering robustness into world action models via mechanistic interpretability and optimal control,” *arXiv preprint arXiv:2607.14943*, 2026.
- [13] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai *et al.*, “ $\pi_{0.5}$: a vision-language-action model with open-world generalization,” *arXiv preprint arXiv:2504.16054*, 2025.
- [14] M. J. Kim, Y. Gao, T.-Y. Lin, Y.-C. Lin, Y. Ge, G. Lam, P. Liang, S. Song, M.-Y. Liu, C. Finn *et al.*, “Cosmos policy: Fine-tuning video models for visuomotor control and planning,” *arXiv preprint arXiv:2601.16163*, 2026.
- [15] A. Geiger, D. Ibeling, A. Zur, M. Chaudhary, S. Chauhan, J. Huang, A. Arora, Z. Wu, N. Goodman, C. Potts *et al.*, “Causal abstraction: A theoretical foundation for mechanistic interpretability,” *Journal of Machine Learning Research*, vol. 26, no. 83, pp. 1–64, 2025.
- [16] K. Park, Y. J. Choe, and V. Veitch, “The linear representation hypothesis and the geometry of large language models,” in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML’24. JMLR.org, 2024.
- [17] S. Marks and M. Tegmark, “The geometry of truth: Emergent linear structure in large language model representations of true/false datasets,” *arXiv preprint arXiv:2310.06824*, 2023.
- [18] C. Tigges, O. J. Hollinsworth, A. Geiger, and N. Nanda, “Linear representations of sentiment in large language models,” *arXiv preprint arXiv:2310.15154*, 2023.
- [19] R. Hendel, M. Geva, and A. Globerson, “In-context learning creates task vectors,” *arXiv preprint arXiv:2310.15916*, 2023.
- [20] E. Todd, M. L. Li, A. S. Sharma, A. Mueller, B. C. Wallace, and D. Bau, “Function vectors in large language models,” *arXiv preprint arXiv:2310.15213*, 2023.
- [21] S. Zhang, T. Yu, and Y. Feng, “Truthx: Alleviating hallucinations by editing large language models in truthful space,” in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 8908–8949.
- [22] T. Wang, X. Jiao, Y. Zhu, Z. Chen, Y. He, X. Chu, J. Gao, Y. Wang, and L. Ma, “Adaptive activation steering: A tuning-free llm truthfulness improvement method for diverse hallucinations categories,” in *Proceedings of the ACM on Web Conference 2025*, 2025, pp. 2562–2578.
- [23] A. Arditi, O. Obeso, A. Syed, D. Paleka, N. Panickssery, W. Gurnee, and N. Nanda, “Refusal in language models is mediated by a single direction,” *arXiv preprint arXiv:2406.11717*, 2024.
- [24] M. A. Khan, N. Boskov, F. M. Anwar, and M. A. Khan, “Controlling vision–language–action policies through sparse latent directions,” in *Mechanistic Interpretability Workshop at NeurIPS 2025*, 2025. [Online]. Available: <https://openreview.net/forum?id=wtf3ww1EOL>
- [25] C. Mitra, Y. Luo, R. Saravanan, D. Niu, A. Pai, J. Thomason, T. Darrell, A. Anwar, D. Ramanan, and R. Herzig, “Mechanistic finetuning of vision-language-action models via few-shot demonstrations,” *arXiv preprint arXiv:2511.22697*, 2025.
- [26] M. Nakamoto, O. Mees, A. Kumar, and S. Levine, “Steering your generalists: Improving robotic foundation models via value guidance,” in *Conference on Robot Learning*, 2025.
- [27] J. Kwok *et al.*, “Robomonkey: Scaling test-time sampling and verification for vision-language-action models,” in *Conference on Robot Learning*, 2025.
- [28] S. Jang, D. Kim, C. Kim, Y. Kim, and J. Shin, “Verifier-free test-time sampling for vision-language-action models,” in *ICLR*, 2026.
- [29] M. Dai *et al.*, “Rover: Robot reward model as test-time verifier for vision-language-action model,” *arXiv preprint arXiv:2510.10975*, 2025.
- [30] G. Zhang, H. Tang, and Y. Yan, “Versatile navigation under partial observability via value-guided diffusion policy,” in *CVPR*, 2024.
- [31] L. Yang, Z. Jiang, C. Sheng, and Z. Tang, “Guided action flow: Q-guided inference for flow-matching vision-language-action policies,” *arXiv preprint arXiv:2607.02092*, 2026.
- [32] Z. Wang, D. K. Jha, A. H. Qureshi, and D. Romeres, “Ppguide: Steering diffusion policies with performance predictive guidance,” *arXiv preprint arXiv:2603.10980*, 2026.
- [33] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone, “Libero: Benchmarking knowledge transfer for lifelong robot learning,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 44 776–44 791, 2023.
- [34] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [35] C. Agia, R. Sinha, J. Yang, Z. Cao, R. Antonova, M. Pavone, and J. Bohg, “Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress,” in *Proceedings of The 8th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 270. PMLR, 2025, pp. 689–723.