DERAIL: ESCAPING TRAJECTORY PRIORS VIA COUNTERFACTUAL ACTION STEERING FOR MID-EXECUTION INSTRUCTION SWITCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Robots operating in the real world must often adapt when user instructions change mid-execution, yet current instruction-conditioned robot policies, including Vision-Language-Action (VLA) models and World-Action Models (WAMs), are largely evaluated under fixed-instruction episodes. We study mid-execution instruction switching and identify trajectory hijacking as a key failure mode, where the policy fails to redirect sufficiently toward the switched target, with its post-switch behavior remaining fully or partially drawn toward trajectories associated with non-target tasks. We propose Counterfactual Action Steering (CAS), a training-free, model-agnostic inference-time method that identifies interfering non-target tasks, queries the same policy with corresponding counterfactual instructions, and uses target–counterfactual action differences to steer execution away from interfering trajectory priors. Experiments on LIBERO-Goal-Switch and CALVIN-Custom-Switch across five VLA and WAM policies, together with real-robot evaluation, show that CAS consistently improves switching success; for example, on CALVIN-Custom-Switch in the $[15, 21]$ switching range, CAS improves $\pi_{0.5}$ from 5.8% to 54.6%. Successful CAS-guided rollouts further provide higher-quality self-training supervision than vanilla successful rollouts with the same number of trajectories, yielding stronger policies that generalize to unseen later instruction switches. Code will be made publicly available.

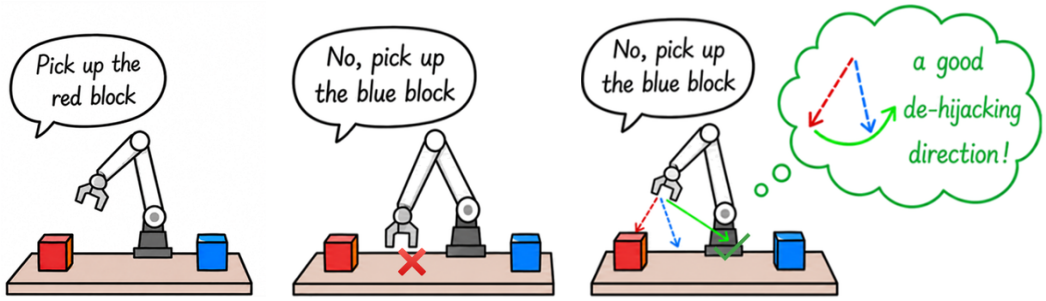
1 INTRODUCTION

Instruction-conditioned robot policies, including Vision-Language-Action (VLA) models (Black et al., 2025b;a; Kim et al., 2025; Li et al., 2024) and World-Action Models (WAMs) (Li et al., 2026; Yuan et al., 2026), have emerged as a promising approach to general-purpose robot manipulation, where a single policy can perform diverse tasks conditioned on natural-language instructions. Despite this progress, existing benchmark evaluations typically provide a fixed task-level instruction at the beginning of each episode and assess whether the corresponding task is completed. This protocol implicitly assumes that the language command remains unchanged throughout execution (Liu et al., 2023; Li et al., 2025b; Zhang et al., 2025b; Chen et al., 2026a).

In real-world manipulation, however, instructions are not always fixed: users may update the task goal after execution has already begun (Chen et al., 2026b; Li et al., 2025a). For example, a user may first ask a robot to grasp an apple, then change the command to grasp an orange after the robot has already moved toward the apple. This demands robot policies capable of redirecting their behavior from an intermediate state in response to a new instruction. Consistent with recent studies (Chen et al., 2026b; Li et al., 2025a), we observe a striking performance degradation across modern VLA and WAM policies under mid-execution instruction switching, even when both the source and target tasks have been individually well-mastered.

Motivation. Fig. 1 (a,b) illustrates a key failure mode in mid-execution instruction switching—*trajectory hijacking*. The robot is initially instructed to pick up the red block. After its end-effector has moved toward the red block, the instruction is changed to pick up the blue block. However, the policy often fails to redirect its motion sufficiently toward the blue block. Instead, its post-switch behavior remains fully or partially drawn toward the red-block trajectory, as if its motion were

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Before switch (b) Without CAS: Hijacked (c) With CAS: Steered Away from Hijacking

Figure 1: **Illustration of trajectory hijacking and the intuition behind CAS.** The robot is first instructed to pick up the red block and starts moving toward it. The instruction is then changed to pick up the blue block. Without CAS, the policy may fail to redirect sufficiently toward the blue block; instead, its post-switch behavior can remain drawn toward the red-block trajectory, a failure mode we refer to as trajectory hijacking. In (c), the **dashed red arrow** denotes the action predicted under the red-block instruction, and the **dashed blue arrow** denotes the action predicted under the blue-block instruction. CAS contrasts these two actions, and their difference provides a corrective direction, producing the **green corrected action** that steers the robot away from the interfering trajectory prior and toward the switched target.

hijacked by this trajectory. We analyze this failure mode in Sec. 2.2, where evidence suggests that visual cues in the current state can induce trajectory priors that contribute to trajectory hijacking.

To address instruction switching failures, SwitchVLA (Li et al., 2025a) introduces a policy module specifically trained to handle execution-aware task transitions. However, its applicability is constrained to a specific model architecture and requires additional training. ReSteer (Chen et al., 2026b) relies on a relatively elaborate multi-stage pipeline for steering-data generation and iterative policy refinement, resulting in considerable engineering and implementation complexity. These limitations motivate us to seek a lightweight and broadly applicable strategy for instruction switching.

Fig. 1 (c) illustrates the design intuition behind our approach. In this representative example, after the instruction switches from the red block to the blue block, the target-conditioned action may still be insufficient to fully redirect the robot toward the blue block. Nevertheless, compared with the counterfactual action predicted under the red-block instruction, it is still biased more toward the desired blue-block behavior. Motivated by this, we ask: *Can action differences between target and hijacking instructions provide a corrective signal for steering execution away from interfering non-target trajectories during mid-execution instruction switching?*

Our solution. In this paper, we propose *Counterfactual Action Steering* (CAS), a training-free, model-agnostic inference-time method for mitigating trajectory hijacking. CAS first estimates which non-target tasks may exert strong trajectory priors at the current robot state by comparing the current end-effector position with task demonstration trajectories. It then queries the policy with the target instruction and with counterfactual instructions corresponding to the identified hijacking tasks. The difference between the target-conditioned action and each counterfactual action provides a corrective direction for steering execution away from the corresponding non-target trajectory prior.

We evaluate CAS across five VLA and WAM policies on our two instruction-switching benchmarks, *LIBERO-Goal-Switch* and *CALVIN-Custom-Switch*, as well as in real-robot experiments. CAS significantly improves mid-execution instruction switching success rates and outperforms PCD (Wu et al., 2026); for example, on CALVIN-Custom-Switch in the $[15, 21]$ switching range, CAS improves π_0 from 6.7% to 45.4% and $\pi_{0.5}$ from 5.8% to 54.6%. Beyond inference-time steering, CAS also supports policy self-improvement: successful CAS-guided rollouts exhibit more effective de-hijacking behavior, providing higher-quality self-training supervision than vanilla successful rollouts. As a result, self-training on CAS-guided rollouts improves the original policy more effectively than vanilla self-training with the same number of trajectories, and this advantage persists when evaluated on unseen, later instruction switches. To summarize, our contributions are as follows:

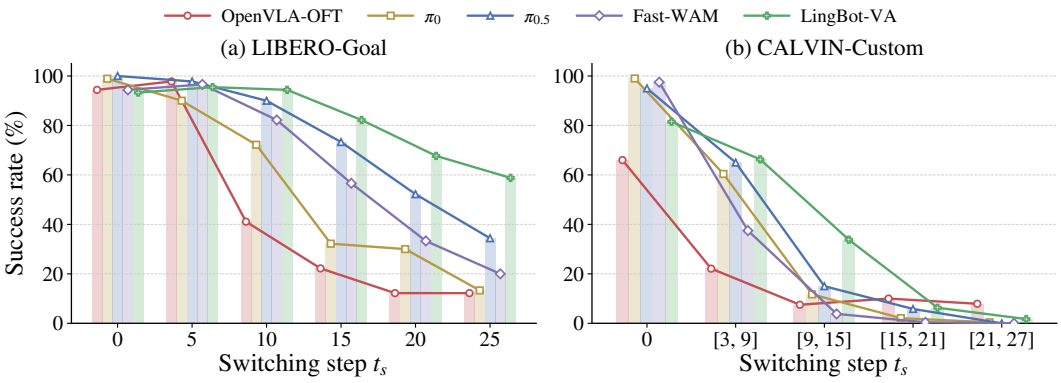


Figure 2: **Success rates of OpenVLA-OFT, π_0, $\pi_{0.5}$, Fast-WAM and LingBot-VA at different switching steps.** Performance generally degrades with later switches.

- We study and analyze mid-execution instruction switching across multiple VLA and WAM policies, identifying trajectory hijacking as a key failure mode driven by visual trajectory priors.
- We propose CAS, a training-free, model-agnostic inference-time method for mitigating trajectory hijacking, and demonstrate its effectiveness in simulation and real-robot experiments.
- We show that CAS-guided rollouts contain more effective de-hijacking behaviors and provide higher-quality self-training supervision than vanilla successful rollouts. With matched numbers of self-training trajectories, CAS-guided self-training yields stronger policies that generalize to unseen, later instruction switches.

2 Problem Formulation and Motivation

We first formulate the mid-execution instruction switching problem and then present key observations that motivate our method. Further discussion of related work is provided in Appendix A.

2.1 Mid-Execution Instruction Switching

We study mid-execution instruction switching for instruction-conditioned robot policies, including both VLA and WAM models. Starting from environment step $t = 0$, the robot acts under a *source instruction* I_{src}. At step t_s, referred to as the *switching step*, the active instruction is changed to a *target instruction* I_{tgt}. From step t_s onward, the policy is expected to redirect its behavior from the current state to complete the target task specified by I_{tgt}. The success of an instruction-switching episode is determined by completion of the target task.

2.2 Key Observations

Observation 1: Mid-execution instruction switching degrades robot policy performance. Fig. 2 evaluates OpenVLA-OFT, π_0, $\pi_{0.5}$, Fast-WAM, and LingBot-VA at different switching steps t_s on LIBERO-Goal and CALVIN-Custom, where $t_s = 0$ denotes directly executing the target task without an instruction switch. At the switch, we discard any unexecuted source-conditioned actions in the current action chunk and clear cached temporal context to eliminate explicit carryover from the source phase. Performance generally declines with increasing switching step t_s, with particularly severe degradation at larger t_s on CALVIN-Custom, although robustness varies across policies. These results show that current robot policies struggle under mid-execution instruction switching. Further experimental details are provided in Appendix H.1.

Observation 2: Robot policies are strongly influenced by visual priors. One possible explanation for this degradation is that policy behavior may be insufficiently sensitive to language instructions. This motivates us to examine how modalities affect policy behavior. To this end, we conduct inference-time modality interventions on LIBERO. We separately perturb vision, language,

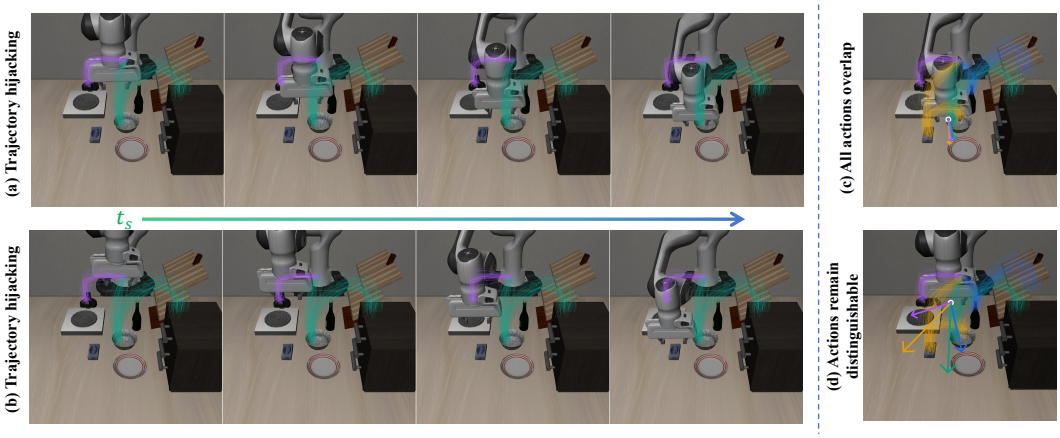


Figure 3: **Visualization of trajectory hijacking and instruction-conditioned action contrast.** Colors denote four tasks: *put the wine bottle on top of the cabinet*, *put the bowl on top of the cabinet*, *put the cream cheese in the bowl*, and *turn on the stove*. Colored curves show demonstration trajectories, and same-colored arrows show the corresponding 3D end-effector translational actions projected onto the 2D camera plane. Panels (a,b) show two forms of trajectory hijacking in bowl-to-stove switching rollouts, with the first frame at t_s and subsequent frames after the switch. In (a), the policy completes the non-target bowl task; in (b), it partially drifts toward the bowl trajectory. Panels (c,d) show instruction-conditioned actions predicted from the same observation under the four task instructions: they nearly overlap in (c) but remain clearly distinguishable in (d).

and, when available, proprioception: we replace visual inputs with a constant mean image, task instructions with "remain still", and proprioceptive inputs with the initial robot state. Across policies, disrupting vision causes the most severe performance collapse, while removing task-specific information from the instruction often preserves non-trivial success. Moreover, under language intervention, policies frequently continue to exhibit recognizable task-like manipulation behaviors, as if still executing a plausible task. In contrast, under visual intervention, such task-likeness is substantially degraded, manifesting as erratic motions, repetitive motion along a nearly fixed direction, or movements with noticeably reduced magnitudes.

We further finetune the policy on the instruction "remain still". When the instruction is switched to "remain still" mid-execution, the policy often still fails to stop. For example, when the visual observation shows the gripper already close to an object, the policy tends to continue the grasping behavior cued by the observed gripper–object configuration, moving further toward the object instead of remaining stationary. These results suggest that robot policies are strongly influenced by visual priors. Appendix D.1 provides additional experimental details, results, and visualizations.

Observation 3: Non-target trajectory priors hijack post-switch behavior. A key failure mode in mid-execution instruction switching is insufficient redirection toward the target task. Consider the switch shown in Fig. 3 (a,b), from grasping the bowl to turning on the stove. By the switching step, the gripper has already moved close to the bowl under the source instruction. After the instruction changes, the target-conditioned action may fail to redirect the robot sufficiently toward the stove, and the subsequent behavior remains fully or partially drawn toward the bowl-grasping trajectory, as if hijacked by this trajectory. More generally, the interfering trajectory need not correspond to the source task: another non-target trajectory can similarly bias the post-switch behavior away from the target. We refer to this failure mode as *trajectory hijacking*, encompassing both complete execution of a non-target task and partial drift toward a non-target trajectory. We observe non-target task completion across multiple policies and switching ranges, as detailed in Appendix D.2.

Observation 2 suggests that misleading visual priors play an important role in trajectory hijacking. For example, in standard demonstrations, states in which the gripper is already close to an object are typically followed by actions that continue toward grasping that object, whereas counterfactual examples are scarce, training the policy to follow visually suggested behavior. We refer to this learned tendency to follow the task behavior suggested by the current state as a *trajectory prior*.

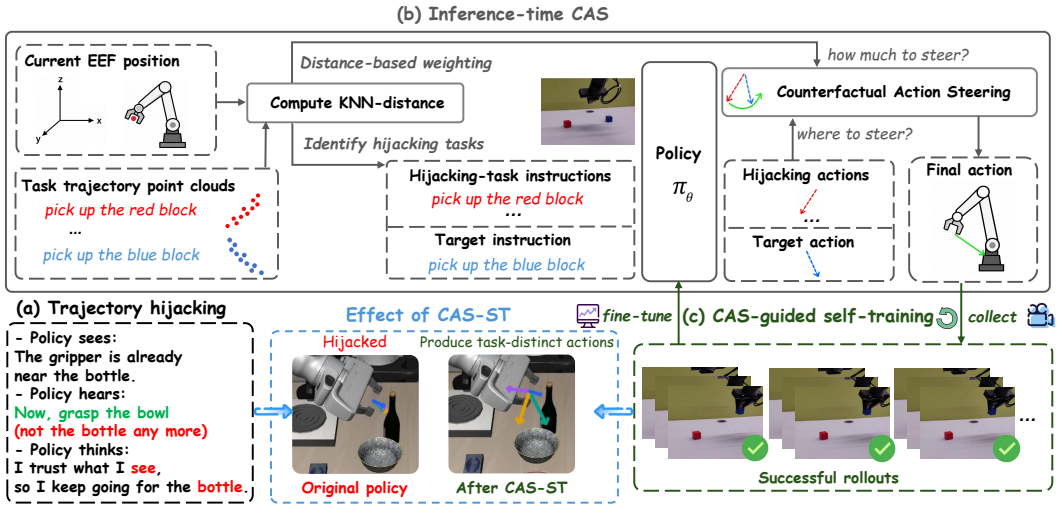


Figure 4: **Overview of CAS.** (a) Under mid-execution instruction switching, strong visual priors can cause the policy to remain drawn toward a non-target trajectory, resulting in trajectory hijacking. (b) Inference-time CAS identifies interfering non-target tasks, uses target–counterfactual action differences to determine where to steer, and trajectory proximity to determine how much to steer. In the red-to-blue block example, the hijacking task happens to coincide with the source task, but this need not hold in general. For clarity, the arrows show a single action from the predicted action chunk. (c) Successful CAS-guided rollouts can be used for self-training (CAS-ST), and this process can be applied iteratively. The blue-boxed comparison illustrates the effect of CAS-ST: given the same observation, the policy produces more task-distinct actions after self-training.

To further examine how trajectory priors affect action prediction, we compare actions predicted from the same observation under different task instructions. In the extreme case shown in Fig. 3 (c), a strong trajectory prior causes actions conditioned on different instructions to nearly overlap. However, in many states, the relative differences among actions conditioned on different task instructions remain informative: different instructions still induce distinguishable action directions, as shown in Fig. 3 (d). This suggests that instruction-conditioned action differences can provide a useful signal for correcting trajectory-induced bias.

3 Method

3.1 Overview

Figure 4 provides an overview of our approach. CAS is a training-free, model-agnostic inference-time method for mitigating trajectory hijacking during mid-execution instruction switching. Beyond inference, successful CAS-guided rollouts can be used for self-training, providing de-hijacking supervision and enabling iterative improvement of the underlying policy.

3.2 Inference-Time Counterfactual Action Steering

At a post-switch step t, let V_t be the visual observation and $p_t \in \mathbb{R}^3$ the end-effector position, and assume the scene contains n candidate tasks. The policy predicts the target-conditioned action chunk $A_{\text{tgt}} = \pi_\theta(V_t, I_{\text{tgt}}) = [a^1_{\text{tgt}}, \ldots, a^H_{\text{tgt}}]$, where H is the prediction horizon and a^k_{tgt} denotes the k-th action in the chunk. CAS first identifies non-target tasks whose trajectory priors may interfere with the current state, and then determines where and how strongly to steer away from them.

For each task $i \in \{1, \ldots, n\}$, let D_i denote its task-specific trajectory point cloud, formed by end-effector positions from training demonstrations. In standard demonstrations, states in which the end effector is already near an object are typically followed by actions that continue interacting with that object, while counterfactual continuations are scarce. Repeated exposure to such gripper–object configurations therefore induces a stronger tendency to continue the corresponding task behavior.

We use local trajectory proximity as a lightweight proxy for the strength of this visual trajectory prior, and compute the average K-nearest-neighbor distance

$$d_i(p_t) = \frac{1}{K} \sum_{p \in \mathrm{KNN}(p_t, D_i)} \|p_t - p\|_2. \tag{1}$$

A smaller $d_i(p_t)$ means that p_t is close to many demonstrated states of task i, indicating stronger local support for continuing that task. CAS retains non-target tasks with $d_i(p_t) < \delta$ as hijacking tasks, where δ is a hyperparameter threshold, denoted by $\mathcal{H}_t = \{h_1, \ldots, h_m\}$.

For each $h_j \in \mathcal{H}_t$, CAS queries the same policy under the same observation with the corresponding hijacking instruction I_{h_j}, obtaining the counterfactual action chunk $A_{h_j} = \pi_\theta(V_t, I_{h_j}) = [a^1_{h_j}, \ldots, a^H_{h_j}]$. At horizon index k, the contrast $a^k_{\mathrm{tgt}} - a^k_{h_j}$ determines *where to steer*: it provides a direction away from the trajectory prior of h_j relative to the target behavior. The trajectory distance determines *how much to steer*: CAS weights this direction by $\lambda / d_{h_j}(p_t)$, giving stronger corrections to hijacking tasks with stronger trajectory priors. The corrected action at horizon index k is

$$a^k_{\mathrm{final}} = a^k_{\mathrm{tgt}} + \lambda \sum_{h_j \in \mathcal{H}_t} \frac{1}{d_{h_j}(p_t)} \big(a^k_{\mathrm{tgt}} - a^k_{h_j}\big), \qquad k = 1, \ldots, H, \tag{2}$$

where $\lambda > 0$ controls the overall steering strength. CAS applies this correction to all action dimensions except the gripper, whose command is taken from the target-conditioned action. The corrected action chunk steers execution away from interfering trajectory priors and toward the switched target. Additional implementation details and qualitative examples are provided in Appendix B.

As a lightweight variant, *Source-only CAS* uses only the source instruction as the counterfactual and applies a fixed steering weight, requiring neither trajectory data nor additional candidate-task instructions while retaining the core counterfactual steering mechanism of CAS. Although less effective than full CAS, it still noticeably improves switching success, as shown in Sec. 4.4.

3.3 CAS-Guided Self-Training

Beyond inference-time steering, CAS can also improve the underlying policy through self-training. We collect successful CAS-guided rollouts and finetune the original policy on their post-switch segments under the updated instruction. These trajectories provide explicit de-hijacking supervision, helping the policy learn to escape interfering trajectory priors and generalize recovery to later switching states unseen during self-training. Additional details are provided in Appendix H.3.

4 Experiments

4.1 Benchmark Construction

We construct two benchmarks for evaluating mid-execution instruction switching: LIBERO-Goal-Switch, built on LIBERO-Goal (Liu et al., 2023), and CALVIN-Custom-Switch, built on the CALVIN simulator (Mees et al., 2022). For *LIBERO-Goal-Switch*, we evaluate switching at $t_s \in \{10, 15, 20, 25\}$, covering progressively later stages of source-task execution. For *CALVIN-Custom-Switch*, episodes are grouped into three switching ranges, $t_s \in [9, 15]$, $[15, 21]$, and $[21, 27]$, corresponding to progressively later progress along the source-task trajectory. Further benchmark construction and evaluation details are provided in Appendix C.

4.2 CAS Mitigates Trajectory Hijacking at Inference Time

Simulation evaluation. We evaluate CAS on *LIBERO-Goal-Switch* and *CALVIN-Custom-Switch* using OpenVLA-OFT, π_0, $\pi_{0.5}$, Fast-WAM, and LingBot-VA. To the best of our knowledge, no public implementations or compatible checkpoints of existing instruction-switching methods were available; faithfully reproducing them would require substantial method-specific implementation and retraining, making a faithful comparison impractical. As a training-free baseline, we compare against Policy Contrastive Decoding (PCD) (Wu et al., 2026), which mitigates spurious correlations inherited from pre-training trajectories through inference-time contrastive decoding. We omit PCD

Table 1: **Simulation results for inference-time instruction switching.** We report target-task success rates (%) on LIBERO-Goal-Switch and CALVIN-Custom-Switch.

Policy	Method	LIBERO-Goal-Switch				CALVIN-Custom-Switch		
		$t_s = 10$	$t_s = 15$	$t_s = 20$	$t_s = 25$	$[9, 15]$	$[15, 21]$	$[21, 27]$
OFT	Base	41.1	22.2	12.2	12.2	11.3	5.4	0.0
	+ CAS	**65.6**	**31.1**	**21.1**	**15.6**	**25.4**	**13.8**	**4.6**
π_0	Base	72.2	32.2	30.0	13.3	22.1	6.7	0.0
	+ PCD	62.2	35.6	28.9	17.8	22.5	5.4	0.0
	+ CAS	**77.8**	**51.1**	**37.8**	**23.3**	**73.8**	**45.4**	**20.4**
$\pi_{0.5}$	Base	90.0	73.3	52.2	34.4	23.3	5.8	0.8
	+ PCD	95.6	74.4	51.1	32.2	26.3	5.4	0.0
	+ CAS	**96.7**	**85.6**	**62.2**	**44.4**	**77.1**	**54.6**	**14.2**
Fast-WAM	Base	82.2	56.7	33.3	20.0	16.7	0.4	0.0
	+ CAS	**93.3**	**67.8**	**44.4**	**34.4**	**49.6**	**19.6**	**0.4**
LingBot-VA	Base	94.4	82.2	67.8	58.9	25.0	9.6	0.0
	+ CAS	**96.7**	**90.0**	**75.6**	**65.6**	**40.0**	**25.0**	**17.1**

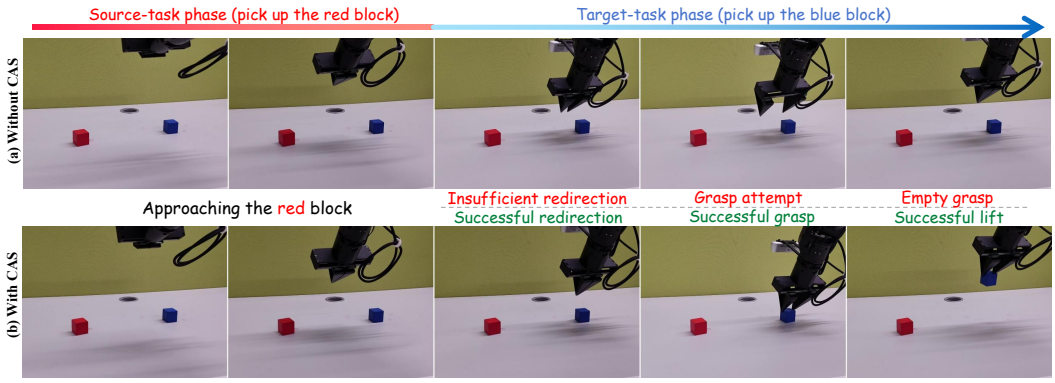


Figure 5: **Real-world comparison of mid-execution instruction switching.** The instruction switches from *picking up the red block* to *picking up the blue block*. Without CAS, insufficient redirection leads to an empty grasp; with CAS, the robot successfully picks up the blue block.

for OpenVLA-OFT because PCD is designed for autoregressive and diffusion-based policies and is not directly applicable to the mainstream OpenVLA-OFT implementation. As shown in Table 1, CAS consistently improves over the corresponding base policies across both benchmarks, all five evaluated policies, and all switching settings, while also outperforming PCD in every setting where PCD is applicable. The gains are particularly pronounced on *CALVIN-Custom-Switch*. For example, in the $[9, 15]$ switching range, CAS improves the success rate of π_0 from 22.1% to 73.8% and that of $\pi_{0.5}$ from 23.3% to 77.1%, corresponding to gains of 51.7 and 53.8 percentage points, respectively. We further analyze the inference-time efficiency of CAS by comparing the base policy, PCD, and CAS using π_0 on *CALVIN-Custom-Switch* in the $[15, 21]$ switching range, with all methods evaluated under the same hardware and evaluation setup. Despite using a serial implementation of counterfactual policy queries without batching, CAS incurs only moderate inference overhead. Its average inference latency per replan is $1.54\times$ that of the base policy, whereas PCD requires $7.71\times$ the base latency. Interestingly, CAS also achieves a lower average episode time than the base policy (12.13 s vs. 14.14 s), as its substantially higher switching success allows more episodes to complete the target task before reaching the maximum episode horizon. Details are provided in Appendix E.1.

We further evaluate CAS under diverse settings. CAS consistently improves over base policies across varying trajectory concentrations and fully randomized object placements (Appendix E.2). It also remains effective under increasing training–evaluation appearance gaps (Appendix E.3) and consistently outperforms the base policy across inference random seeds (Appendix E.4).

Table 2: **Evaluation of CAS-guided self-training.** We compare CAS-ST with Vanilla-ST using equal numbers of successful trajectories at the data-collection setting and an unseen later switch.

	Data-collection settings				Generalization to Unseen Later Switches			
	LIBERO-Goal-Switch		CALVIN-Custom-Switch		LIBERO-Goal-Switch		CALVIN-Custom-Switch	
Method	$\pi_{0.5}$	Fast-WAM	$\pi_{0.5}$	Fast-WAM	$\pi_{0.5}$	Fast-WAM	$\pi_{0.5}$	Fast-WAM
Base	34.4	20.0	23.3	16.7	27.8	16.7	5.8	0.4
Vanilla-ST	48.9	24.4	32.5	21.3	32.2	21.1	8.3	7.9
CAS-ST	**52.2**	**27.8**	**57.1**	**50.0**	**40.0**	**27.8**	**23.8**	**15.4**

Real-world evaluation. We evaluate CAS on an AIRBOT Play robot arm. The robot first executes an instruction to pick up one block (red or blue), and the instruction is then switched to the other at step t_s. We test both switching directions at $t_s = 20$ and $t_s = 30$, with 20 trials per setting, using matched object placements for the base and CAS trials. As shown in Table 3, CAS improves π_0 in all settings. The real-world policy predicts delta joint actions rather than the delta end-effector actions used in simulation, suggesting that CAS is not tied to a specific action representation. Fig. 5 shows a representative switch from picking up the red block to picking up the blue block. Without CAS, the policy remains biased toward the red-block trajectory prior and fails to redirect sufficiently; with CAS, it escapes the interfering prior and successfully picks up the blue block.

4.3 Beyond Inference: CAS Produces Higher-Quality Rollouts for Self-Training

We finetune the original policy on successful CAS-guided rollouts, denoted as CAS-ST, and compare it with the base policy and vanilla self-training (Vanilla-ST), which finetunes the same base policy on the same number of successful rollouts collected without CAS. All finetuned policies are evaluated without CAS at inference time. We evaluate direct self-training gains at the data-collection setting and generalization at unseen later switches, where greater progress along the source-task trajectory makes redirection more challenging. For LIBERO-Goal-Switch, self-training trajectories are collected at $t_s = 25$; we evaluate at $t_s = 25$ and further test generalization at the unseen later switch $t_s = 30$. For CALVIN-Custom-Switch, trajectories are collected in the $t_s \in [9, 15]$ switching range and evaluated in both the same range and the unseen later range $t_s \in [15, 21]$.

As shown in Table 2, Vanilla-ST improves over the base policy, while CAS-ST consistently yields further gains across both benchmarks and both evaluated policies, both at the data-collection settings and at unseen later switches. These results show that successful CAS-guided rollouts provide higher-quality supervision for self-training than vanilla successful rollouts with the same number of trajectories. Qualitative comparisons of paired successful Base and CAS rollouts in Appendix F.1 further show that CAS can produce more effective de-hijacking behavior even when both methods ultimately complete the target task, highlighting differences in trajectory quality that are not captured by binary success alone. The consistent gains of CAS-ST at unseen later switches further indicate that the recovery behaviors learned from CAS-guided trajectories extrapolate beyond the switching regimes used for self-training.

Motivated by this extrapolation, we further investigate whether CAS-guided self-training can be applied iteratively to progressively improve recovery at later switches. Starting from the one-round $\pi_{0.5}$ CAS-ST checkpoint on LIBERO-Goal-Switch, whose self-training trajectories were collected at $t_s = 25$, we collect additional successful CAS-guided trajectories at $t_s = 40$ and perform a second round of self-training. As shown in Table 4, the one-round checkpoint already outperforms the base policy at $t_s = 40, 50, 60$, well beyond its data-collection setting. The second round further improves performance at all three switching steps. Notably, although the second-round data are collected only at $t_s = 40$, the gains persist at the unseen later switches $t_s = 50$ and $t_s = 60$. These results show that CAS-guided self-training can be iterated to progressively improve recovery under increasingly challenging switching conditions. To further analyze the effect of self-training on instruction-conditioned action predictions, we compare instruction-conditioned actions before and after two rounds of CAS-ST, as in Fig. 3 (d). After self-training, the policy produces more distin-

guishable actions that are better aligned with the corresponding task directions, even in challenging states where the base policy predicts nearly overlapping actions; see Appendix F.2.

We provide additional analyses of CAS-guided self-training in the appendix. CAS-ST is substantially more finetuning-efficient than Vanilla-ST (Appendix F.3). Pair-level analysis identifies cases where CAS-ST outperforms Vanilla-ST despite fewer retained self-training trajectories (Appendix F.4). CAS-ST's advantage also persists at a later unseen switch (Appendix F.5) and under fully randomized trajectory distributions (Appendix F.6).

Table 3: Real-world switching.

Policy	t_s	Blue→Red	Red→Blue
Base	20	7/20	7/20
+ CAS	20	**13/20**	**15/20**
Base	30	2/20	6/20
+ CAS	30	**10/20**	**14/20**

Table 4: Iterative CAS-guided self-training on LIBERO-Goal-Switch with $\pi_{0.5}$.

Method	$t_s = 40$	$t_s = 50$	$t_s = 60$
Base	28.9	24.4	17.8
CAS-ST (1 round)	36.7	32.2	28.9
CAS-ST (2 rounds)	**46.7**	**42.2**	**37.8**

Table 5: **CAS variants and design ablation.** We compare full CAS, its lightweight Source-only CAS variant, and All non-target Fixed. Source-only CAS requires no trajectory data and achieves 35.4% Switch SR. Full CAS achieves the highest success rate of 45.4% with state-dependent task selection and adaptive steering across relevant hijacking tasks.

Method	Counterfactual task set	Required task trajectory data	Steering weight	Switch SR ↑
Base	–	None	–	6.7
Source-only CAS	Source task	None	$w = 0.6$	35.4
All non-target Fixed	All non-target tasks	None	$w = 0.4$	23.3
CAS (Full)	KNN-selected non-target tasks	All candidate tasks	Adaptive	**45.4**

4.4 Ablation Study

We conduct all ablation studies using π_0 on CALVIN-Custom-Switch in the switching range $t_s \in [15, 21]$, covering both design and hyperparameter ablations.

Design ablations. Table 5 compares full CAS with its lightweight Source-only CAS variant and All non-target Fixed, which uniformly steers against all non-target tasks. For a fair comparison, we sweep the fixed steering weight over $w \in \{0.1, 0.2, 0.4, 0.6, 0.8\}$ for both fixed-weight variants and report their best results. The complete sweep is provided in Appendix E.5.1. Both variants outperform the base policy under every tested steering weight. Source-only CAS improves Switch SR from 6.7% to 35.4%, demonstrating that the core counterfactual steering mechanism remains effective even without trajectory-based task selection and weighting. All non-target Fixed can over-steer against weakly interfering trajectories, yielding 23.3% Switch SR. Full CAS adaptively weights all identified hijacking tasks and performs best at 45.4%, exceeding Source-only CAS by 10.0 percentage points and highlighting the importance of adaptive steering across relevant hijacking tasks.

Hyperparameter ablations. We vary K, δ, and λ and find that CAS consistently outperforms the base policy across all tested settings. Detailed results are provided in Appendix E.5.2.

5 Conclusion

We studied mid-execution instruction switching across VLA and WAM policies and identified trajectory hijacking as a key failure mode. We proposed CAS, a training-free, model-agnostic inference-time method that steers the policy away from interfering trajectory priors using target–counterfactual action differences. Across two switching benchmarks and real-robot evaluation, CAS consistently improves switching success. CAS-guided rollouts also provide better self-training supervision, yielding stronger policies that generalize to unseen later switches.

AI USE STATEMENT

In this work, we did not use generative AI tools for any tasks with required disclosure. Additionally, we used generative AI tools to edit the research paper to improve readability, including language polishing, grammar correction, and refinement of wording. We have reviewed all AI-assisted edits to ensure that they did not alter the scientific meaning, claims, results, or conclusions of the work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to facilitate reproduction of our results. The formulation and implementation details of CAS are described in Sec. 3 and Appendix B. The construction of LIBERO-Goal-Switch and CALVIN-Custom-Switch, together with their evaluation protocols, is documented in Appendix C and Appendix H.1. Additional experimental details include real-robot settings (Appendix H.2), robustness across random seeds and CAS hyperparameter analyses (Appendices E.4 and E.5), self-training data collection, subsampling, finetuning, and evaluation configurations (Appendix H.3), and implementation details of the PCD baseline (Appendix H.4). Code and the CAS-guided self-training checkpoints underlying our main results will be made publicly available.

REFERENCES

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 27–30 Sep 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025b. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=itonej9GIV.

Zhaorun Chen, Zhuokai Zhao, Hongyin Luo, Huaxiu Yao, Bo Li, and Jiawei Zhou. HALC: Object hallucination reduction via adaptive focal-contrast decoding. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 7824–7846. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/chen24bi.html.

Zhenyang Chen, Alan Tian, Liquan Wang, Benjamin Joffe, Yingyan Celine Lin, Yuxiao Chen, Siddharth Karamcheti, and Danfei Xu. ReSteer: Quantifying and Refining the Steerability of Multitask Robot Policies. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026b. doi: 10.15607/RSS.2026.XXII.143. URL https://www.roboticsproceedings.org/rss22/p143.html.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R Glass, and Pengcheng He. DoLa: Decoding by contrasting layers improves factuality in large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54158–54183, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edc36117f795ca52a0cbf6a7b3882859-Paper-Conference.pdf.

Alessandro Favero, Luca Zancato, Matthew Trager, Siddharth Choudhary, Pramuditha Perera, Alessandro Achille, Ashwin Swaminathan, and Stefano Soatto. Multi-modal hallucination control by visual information grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14303–14312, June 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Favero_Multi-Modal_Hallucination_Control_by_Visual_Information_Grounding_CVPR_2024_paper.html.

Qidong Huang, Xiaoyi Dong, Pan Zhang, Bin Wang, Conghui He, Jiaqi Wang, Dahua Lin, Weiming Zhang, and Nenghai Yu. OPERA: alleviating hallucination in multi-modal large language models via over-trust penalty and retrospection-allocation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pp. 13418–13427.

IEEE, 2024. doi: 10.1109/CVPR52733.2024.01274. URL https://doi.org/10.1109/CVPR52733.2024.01274.

Junho Kim, Hyun Jun Kim, Yeon Ju Kim, and Yong Man Ro. CODE: contrasting self-generated description to combat hallucination in large multi-modal models. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, volume 37, pp. 133571–133599. Curran Associates, Inc., 2024. doi: 10.52202/079017-4246. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/f1592b0d4ab737e18bb1899484d28d96-Abstract-Conference.html.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.017. URL https://www.roboticsproceedings.org/rss21/p017.html.

Yi-Lun Lee, Yi-Hsuan Tsai, and Wei-Chen Chiu. Delve into visual contrastive decoding for hallucination mitigation of large vision-language models, 2024. URL https://arxiv.org/abs/2412.06775.

Sicong Leng, Hang Zhang, Guanzheng Chen, Xin Li, Shijian Lu, Chunyan Miao, and Lidong Bing. Mitigating object hallucinations in large vision-language models through visual contrastive decoding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pp. 13872–13882. IEEE, 2024. doi: 10.1109/CVPR52733.2024.01316. URL https://doi.org/10.1109/CVPR52733.2024.01316.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Luyao Zhang, Mingrui Yu, Zelin Gao, Nan Xue, Boyu Zhou, Xing Zhu, Mingyu Ding, Yujun Shen, and Yinghao Xu. Causal World Modeling for Robot Control. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026. doi: 10.15607/RSS.2026.XXII.016. URL https://www.roboticsproceedings.org/rss22/p016.html.

Meng Li, Zhen Zhao, Zhengping Che, Fei Liao, Kun Wu, Zhiyuan Xu, Pei Ren, Zhao Jin, Ning Liu, and Jian Tang. SwitchVLA: Execution-aware task switching for vision-language-action models, 2025a. URL https://arxiv.org/abs/2506.03574.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. CogACT: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation, 2024. URL https://arxiv.org/abs/2411.19650.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori B. Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12286–12312, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.687. URL https://aclanthology.org/2023.acl-long.687/.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 06–09 Nov 2025b. URL https://proceedings.mlr.press/v270/li25c.html.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 6691–6706. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.ACL-LONG.522. URL https://doi.org/10.18653/v1/2021.acl-long.522.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8c3c666820ea055a77726d66fc7d447f-Paper-Datasets_and_Benchmarks.pdf.

Avshalom Manevich and Reut Tsarfaty. Mitigating hallucinations in large vision-language models (LVLMs) via language-contrastive decoding (LCD). In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6008–6022, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.359. URL https://aclanthology.org/2024.findings-acl.359/.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. doi: 10.1109/LRA.2022.3180108. URL https://doi.org/10.1109/LRA.2022.3180108.

Sean O'Brien and Mike Lewis. Contrastive decoding improves reasoning in large language models. *arXiv preprint arXiv:2309.09117*, 2023. URL https://arxiv.org/abs/2309.09117.

Guillaume Sanchez, Alexander Spangher, Honglu Fan, Elad Levi, and Stella Biderman. Stay on topic with classifier-free guidance. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43197–43234. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/sanchez24a.html.

Weijia Shi, Xiaochuang Han, Mike Lewis, Yulia Tsvetkov, Luke Zettlemoyer, and Wen-tau Yih. Trusting your evidence: Hallucinate less with context-aware decoding. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 783–791, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-short.69. URL https://aclanthology.org/2024.naacl-short.69/.

Yixuan Su and Nigel Collier. Contrastive search is what you need for neural text generation. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=GbkWw3jwL9.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. A contrastive framework for neural text generation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 21548–21561. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/871cae8f599cb8bbfcb0f58fe1af95ad-Paper-Conference.pdf.

Xintong Wang, Jingheng Pan, Liang Ding, and Chris Biemann. Mitigating hallucinations in large vision-language models with instruction contrastive decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*,

pp. 15840–15853, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.937. URL https://aclanthology.org/2024.findings-acl.937/.

Shihan Wu, Xu Luo, Ji Zhang, Junlin Xie, Jingkuan Song, Heng Tao Shen, and Lianli Gao. Policy contrastive decoding for robotic foundation models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 112486–112506, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/b6d67c380f8bde2adc4247d0036c0c73-Paper-Conference.pdf.

Senqiao Yang, Chengyao Wang, Yuxin Chen, Zixuan Wang, Longxiang Tang, Haokun Gui, Jinhui Ye, Changsheng Lu, Xiaoyang Wu, Mingkang Zhu, et al. Beyond data scaling: Representation-centric continued pre-training for vision-language-action models. *arXiv preprint arXiv:2608.27550*, 2026a. URL https://arxiv.org/abs/2608.27550.

Senqiao Yang, Kaichen Zhang, Zhaoyang Jia, Jinghao Guo, Yifei Shen, Xinjie Zhang, Xiaoyi Zhang, Haoqing Wang, Xiao Li, Peng Zhang, et al. Mage-VL: An efficient codec-native streaming multimodal foundation model. *arXiv preprint arXiv:2607.24904*, 2026b. URL https://arxiv.org/abs/2607.24904.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Tong Xu, Hao Wang, Dianbo Sui, Yunhang Shen, Ke Li, Xing Sun, and Enhong Chen. Woodpecker: Hallucination correction for multimodal large language models. *Science China Information Sciences*, 67(12):220105, 2024. doi: 10.1007/s11432-024-4251-x. URL https://doi.org/10.1007/s11432-024-4251-x.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-WAM: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026. URL https://arxiv.org/abs/2603.16666.

Ce Zhang, Zifu Wan, Zhehan Kan, Martin Q. Ma, Simon Stepputtis, Deva Ramanan, Russ Salakhutdinov, Louis-Philippe Morency, Katia Sycara, and Yaqi Xie. Self-correcting decoding with generative feedback for mitigating hallucinations in large vision-language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, pp. 5494–5524, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/109cf25cbc36037deecdbeabfa199956-Abstract-Conference.html.

Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. VLABench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11142–11152, 2025b. doi: 10.1109/ICCV51701.2025.01037. URL https://openaccess.thecvf.com/content/ICCV2025/html/Zhang_VLABench_A_Large-Scale_Benchmark_for_Language-Conditioned_Robotics_Manipulation_with_Long-Horizon_ICCV_2025_paper.html.

Appendix

CONTENTS

A Related Work **17**

B CAS Implementation Details and Illustrative Examples **19**

B.1 CAS Implementation Details 19

B.2 Illustrative Examples 20

C Benchmark Construction Details **21**

C.1 Complementary Online and Offline Protocols 21

C.2 CALVIN-Custom-Switch Details 22

C.3 CALVIN-Custom-Switch-Random Details 23

C.4 LIBERO-Goal-Switch Details 23

D Analyses of Mid-Execution Instruction Switching **24**

D.1 Analysis of Visual Priors 24

D.2 Non-Target Task Completion in Failed Instruction Switching 33

E Additional Experiments and Analyses of Inference-Time CAS **34**

E.1 Inference-Time Efficiency Analysis 34

E.2 Generality across Diverse Trajectory Regimes 35

E.3 Robustness to Scene Appearance Shifts 37

E.4 Robustness Across Evaluation Random Seeds 38

E.5 Additional Ablation Analysis 38

F Additional Experiments and Analyses of CAS-Guided Self-Training **40**

F.1 Qualitative Analysis of Higher-Quality CAS-guided Rollouts 40

F.2 Instruction-Conditioned Action Visualization 42

F.3 Self-Training Performance under Different Training Budgets 45

F.4 Pair-Level Analysis of CAS-Guided Self-Training 45

F.5 Further Generalization to Later Switches 46

F.6 Self-Training on CALVIN-Custom-Switch-Random 46

G Beyond Instruction Switching: Trajectory-Biased Initial States **47**

G.1 Construction of Trajectory-Biased Initial States 47

G.2 Inference-Time Evaluation 48

G.3 CAS-Guided Self-Training 49

H Experimental Details **50**

H.1 Mid-Execution Instruction Switching Evaluation Details 50
H.2 Real-World Experiment Details 51
H.3 Self-Training Details 52
H.4 PCD Implementation Details 54

A Related Work

Generalist instruction-conditioned robot policies. Instruction-conditioned robot policies have emerged as a promising paradigm for general-purpose robot manipulation, enabling a single policy to perform diverse tasks from visual observations and natural-language instructions. A prominent line of work develops Vision-Language-Action (VLA) policies that directly connect multimodal perception and language understanding to action generation. π_0 formulates robot control as a vision-language-action flow model (Black et al., 2025b), while $\pi_{0.5}$ extends this framework toward open-world generalization through co-training on heterogeneous robot data, high-level semantic prediction, web data, and hybrid multimodal examples (Black et al., 2025a). OpenVLA-OFT focuses on efficient adaptation of pretrained VLA models to downstream manipulation tasks through improved action decoding, action representations, and finetuning objectives (Kim et al., 2025). VLAct further explores representation-centric continued pre-training of VLM backbones for improved transfer across action heads, tasks, and robot embodiments (Yang et al., 2026a), while Mage-VL investigates codec-native visual encoding for efficient multimodal perception (Yang et al., 2026b). More recently, World-Action Models (WAMs) have explored predictive world modeling for robot control. LingBot-VA studies causal world modeling for action generation (Li et al., 2026), while Fast-WAM investigates whether explicit future imagination is necessary at test time and develops a more efficient WAM formulation for action prediction (Yuan et al., 2026). Together, these VLA and WAM approaches represent a diverse family of generalist instruction-conditioned robot policies.

Alongside model development, benchmarks and evaluation frameworks such as LIBERO, simulation-based evaluation of real-world robot manipulation policies, VLABench, and RoboTwin 2.0 provide broad testbeds for language-conditioned manipulation, knowledge transfer, long-horizon reasoning, and robust bimanual control (Liu et al., 2023; Li et al., 2025b; Zhang et al., 2025b; Chen et al., 2026a). However, standard evaluations typically assume that the task instruction remains fixed throughout an episode. They therefore measure whether a policy can execute a specified task from an initial state, but do not directly test whether it can redirect its behavior from an intermediate state after the instruction changes. We study this setting under *mid-execution instruction switching* and show that strong policies across both VLA and WAM families can suffer substantial performance degradation, with trajectory hijacking emerging as a key failure mode. These observations motivate approaches that directly address switching failures after execution has already begun.

Instruction switching in instruction-conditioned robot policies. A small but growing body of work has begun to study whether robot policies can follow newly issued instructions from intermediate execution states. Existing approaches have primarily focused on VLA policies, while mid-execution instruction switching in WAMs remains comparatively underexplored.

SwitchVLA (Li et al., 2025a) formulates task switching as an execution-aware behavior modulation problem. Its policy conditions on the previous instruction, current instruction, and propagated contact state, and jointly predicts the contact state, a discrete behavior mode, and a low-level action chunk. It defines three behavior modes: *forward*, which continues execution under the current instruction; *rollback*, which reverses previous actions when an instruction mismatch occurs during physical contact; and *advance*, which transitions toward a canonical starting configuration of the new task when no contact is present. These modes are trained with behavior-specific action targets: forward uses future actions from the original demonstration, rollback uses reversed historical actions, and advance uses an interpolated trajectory toward the new task's canonical start pose. Contact states are obtained through contact-aware trajectory annotation, while behavior modes are explicitly supervised. In this way, SwitchVLA learns switching behaviors from existing demonstrations without requiring additional switch-specific trajectories, but relies on a specialized execution-aware architecture, contact-state supervision, and behavior-specific training.

ReSteer (Chen et al., 2026b) studies a closely related problem under the notion of policy steerability, asking whether a multitask robot policy can respond to newly issued task instructions from intermediate execution states. In addition to an exhaustive rollout-based evaluation protocol, ReSteer introduces conditional mutual information (CMI) between language and action as a proxy for identifying states in which the policy is weakly conditioned on the instruction. Low-CMI states are then prioritized for additional steering supervision. To improve steerability, ReSteer develops a multi-stage data-centric refinement pipeline. It first segments manipulation demonstrations into semantic stages, such as pre-grasp, transport, and place, using kinematic and contact cues. For a selected

low-steerability source state, it searches for a stage-compatible state from a target-task demonstration and synthesizes a cross-task transition between the two, followed by execution under the target instruction. The resulting steering trajectories are added to the training data for policy finetuning. ReSteer further applies self-refining behavioral cloning (SRBC), collecting successful task-switching rollouts from the updated policy and using them for further refinement. Overall, ReSteer improves steerability through a relatively elaborate workflow that combines state-wise steerability estimation, stage-aware trajectory processing, cross-task transition synthesis, policy finetuning, and self-refinement, resulting in considerable methodological and engineering complexity.

These works establish the importance of execution-time instruction switching, but address switching failures primarily through specialized policy design or additional data generation and training. SwitchVLA relies on an execution-aware architecture with explicitly supervised transition modes, while ReSteer constructs steering supervision through a multi-stage refinement pipeline. In contrast, the core CAS method targets instruction-conditioned robot policies broadly, including both VLA and WAM policies, and operates as a training-free, model-agnostic inference-time intervention. Rather than learning or synthesizing a dedicated transition mechanism, CAS queries the same policy under the target instruction and counterfactual hijacking-task instructions, and uses their action differences to steer execution away from interfering trajectory priors. Successful CAS-guided rollouts can further be reused for self-training, providing more effective de-hijacking supervision than vanilla successful rollouts.

Contrastive decoding. Contrastive decoding adjusts model outputs at inference time by contrasting predictions under different conditions to suppress undesirable biases or priors.

In language models, contrastive methods compare predictions from different models, conditioning contexts, or internal representations to improve generation. Contrastive decoding (CD) contrasts a large expert model with a smaller amateur model to suppress degenerate generations (Li et al., 2023), and has also been shown to improve reasoning (O'Brien & Lewis, 2023). Related approaches include DExperts, which combines expert and anti-expert models for decoding-time control (Liu et al., 2021); classifier-free guidance, which contrasts conditional and unconditional predictions (Sanchez et al., 2024); context-aware decoding, which contrasts predictions with and without external context (Shi et al., 2024); and DoLa, which contrasts logits from different transformer layers (Chuang et al., 2024). Related work on contrastive search and training further explores contrastive objectives for mitigating degeneration in neural text generation (Su et al., 2022; Su & Collier, 2023). These works broadly show that undesirable generation patterns can be exposed by an appropriately chosen alternative prediction and suppressed through inference-time contrast.

In vision-language models, similar ideas have been used to mitigate hallucinations through visual or linguistic counterfactuals. Visual Contrastive Decoding (VCD) contrasts predictions from original and distorted visual inputs to reduce reliance on statistical bias and language priors (Leng et al., 2024), while subsequent methods explore alternative visual contrastive signals (Chen et al., 2024; Kim et al., 2024; Lee et al., 2024; Favero et al., 2024). Language-side methods such as LCD also use contrastive signals to reduce hallucination (Manevich & Tsarfaty, 2024). Most related to our use of counterfactual instructions, Instruction Contrastive Decoding (ICD) contrasts predictions under standard and disturbance instructions to suppress hallucination-prone concepts (Wang et al., 2024). Other training-free or decoding-time approaches similarly mitigate LVLM hallucinations through attention reallocation, output validation, or generative feedback (Huang et al., 2024; Yin et al., 2024; Zhang et al., 2025a). Collectively, these methods motivate viewing hallucination as a prior-driven grounding failure that can be exposed through counterfactual predictions.

More recently, Policy Contrastive Decoding (PCD) extends contrastive decoding to generalist robot policies to mitigate spurious correlations learned from robot trajectories (Wu et al., 2026). It contrasts action distributions predicted from the original visual observation and an object-masked counterpart, amplifying predictions associated with task-relevant object features while suppressing spurious visual correlations.

During mid-execution instruction switching, trajectory priors can bias the target-conditioned action, making it insufficient to redirect the policy toward the switched task. Yet even when its absolute direction is inaccurate, the target-conditioned action can remain relatively more aligned with the desired behavior than the action induced by a hijacking-task instruction. This suggests that the target–hijacking action difference can preserve a useful corrective direction despite bias in the target

action itself. Inspired by the core idea of contrastive decoding, CAS amplifies this relative signal to steer instruction-conditioned robot policies away from interfering trajectory priors and toward the switched target.

B CAS Implementation Details and Illustrative Examples

B.1 CAS Implementation Details

We describe several practical details for implementing CAS during inference. The core update follows Eq. (2), while lightweight safeguards such as weight capping or action clipping can be used to improve stability when the counterfactual correction becomes large. We also discuss gripper-dimension exclusion and batched computation of instruction-conditioned action chunks.

Capping per-task steering weights. For each hijacking task $h_j \in \mathcal{H}_t$, the scalar weight associated with its counterfactual correction term in Eq. (2) is

$$r_{h_j} = \frac{\lambda}{d_{h_j}(p_t)}. \tag{3}$$

When the end-effector is very close to the demonstrated trajectory of task h_j, the distance $d_{h_j}(p_t)$ can become small, resulting in an excessively large steering weight. In this case, at horizon index k, the corresponding correction term, $r_{h_j}\left(a^k_{\mathrm{tgt}} - a^k_{h_j}\right)$, may dominate the aggregated correction. To avoid such over-amplification, we cap each per-task steering weight by a maximum value $r_{\max}$:

$$\tilde{r}_{h_j} = \min\left(\frac{\lambda}{d_{h_j}(p_t)},\, r_{\max}\right). \tag{4}$$

This cap is applied independently to each hijacking task before the correction terms are aggregated. When capping is used, $\tilde{r}_{h_j}$ replaces $\lambda/d_{h_j}(p_t)$ in Eq. (2).

Bounding the corrected action. After aggregating the counterfactual correction terms, a corrected action a^k_{final} may occasionally have a larger magnitude than desired, especially when multiple hijacking tasks are selected or when their KNN distances are small. In such cases, bounding the corrected action can improve execution stability. Bounding is applied only to the non-gripper dimensions. A simple choice is per-dimension clipping, which clamps each steered dimension to the valid action range of the policy, such as the normalized range $[-1, 1]$. Another possible choice is norm rescaling, which scales the non-gripper components of each corrected action together so that their magnitude remains within a fixed factor of the corresponding target-conditioned action a^k_{tgt} restricted to the same dimensions, while preserving the direction of the steered non-gripper action vector. The per-task cap $r_{\max}$ limits the contribution of each individual hijacking task before summation, whereas this final bounding step constrains the overall magnitude of a^k_{final} after all correction terms have been aggregated.

Excluding the gripper dimension from steering. CAS steering is applied to all action dimensions except the gripper. In simulation, the steered dimensions correspond to delta end-effector translation and rotation. On the real robot, they correspond to delta joint commands. For each horizon index k, the gripper open/close command is taken directly from the corresponding target-conditioned action a^k_{tgt} and is not modified by CAS.

Computing multiple instruction-conditioned action chunks. CAS requires action-chunk predictions under multiple instructions at the same observation, including the target instruction and the counterfactual instructions associated with the selected hijacking tasks. A simple implementation is to compute these chunks through multiple policy forward passes: one forward pass for I_{tgt} and one additional forward pass for each $h_j \in \mathcal{H}_t$ using instruction I_{h_j}. Each policy query returns an action chunk, and CAS applies Eq. (2) to corresponding horizon indices across the target and counterfactual chunks. This implementation requires only minimal changes to an existing robot-policy inference pipeline, since the policy can be queried repeatedly with the same visual observation and different language instructions.

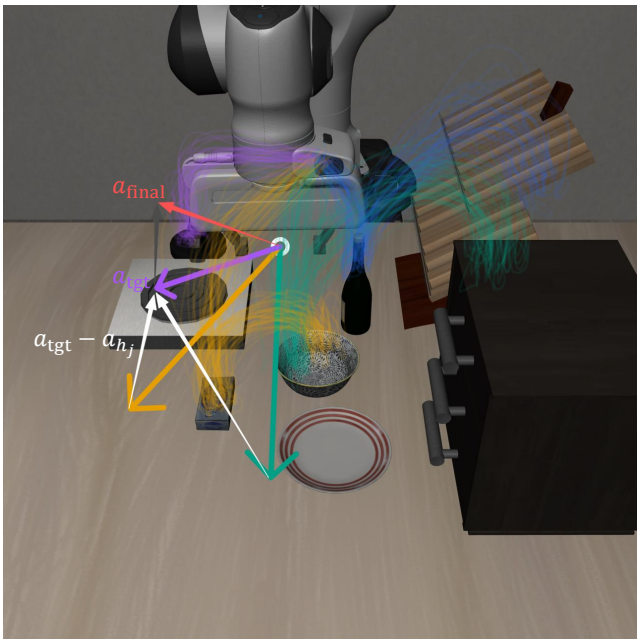


Figure 6: **Action-level illustration of Counterfactual Action Steering.** The robot switches from the source task *put the bowl on top of the cabinet* to the target task *turn on the stove*. Since the current end-effector position p_t is close to the demonstration trajectory distributions of the bowl-related tasks, CAS selects *put the bowl on top of the cabinet* (green) and *put the cream cheese in the bowl* (yellow) as hijacking tasks based on their small KNN distances, while excluding *put the wine bottle on top of the cabinet* (blue) due to its larger KNN distance. CAS contrasts the target-conditioned action a_{tgt} with the selected counterfactual actions a_{h_j} and aggregates the weighted directions $a_{\text{tgt}} - a_{h_j}$ to produce a_{final} (red), which steers the robot away from interfering trajectory priors and toward the stove knob. For clarity, the arrows visualize the translational components of a single action from each predicted action chunk, taken at the same horizon index.

When inference efficiency is important, these instruction-conditioned action chunks can instead be computed in a single batched forward pass. Specifically, the current visual observation V_t is repeated across the batch dimension, while the language inputs are set to $\{I_{\text{tgt}}, I_{h_1}, \ldots, I_{h_m}\}$. The policy then returns the target-conditioned action chunk and all counterfactual action chunks together. In this batched implementation, CAS does not introduce additional sequential policy-forward time for obtaining multiple action chunks in principle.

B.2 Illustrative Examples

Fig. 6 illustrates how CAS corrects the policy action in a concrete mid-execution instruction-switching episode. We illustrate the correction at a single horizon index shared across the predicted action chunks and omit the superscript k for clarity. The robot is first commanded to *put the bowl on top of the cabinet*. After the robot has moved for several steps under this source instruction, the user switches the target task to *turn on the stove*. At the switching state, the current end-effector position p_t is already close to the bowl and to the trajectory distribution of the source task. As a result, directly querying the policy with the switched target instruction produces the target-conditioned action a_{tgt}, shown by the purple arrow, but this action can still be biased toward the bowl-related trajectory prior.

CAS uses the current end-effector position to identify which non-target tasks are likely to interfere with the switched target behavior. In this example, two task trajectories are selected as hijacking sources: *put the bowl on top of the cabinet*, shown in green, and *put the cream cheese in the bowl*, shown in yellow. For both tasks, the corresponding trajectory point clouds contain demonstration points close to the current end-effector position p_t, resulting in small KNN distances. CAS therefore treats them as hijacking tasks and queries the policy with their corresponding counterfactual

instructions. In contrast, the trajectory of *put the wine bottle on top of the cabinet*, shown in blue, is farther from the current end-effector position. Since its KNN distance is larger than the threshold, CAS does not include it in the hijacking set.

For each selected hijacking task $h_j \in \mathcal{H}_t$, CAS compares the target-conditioned action with the action predicted under the corresponding counterfactual instruction. Let a_{h_j} denote this counterfactual action at the same horizon index. The contrast $a_{\mathrm{tgt}} - a_{h_j}$ indicates a direction that moves the policy away from that non-target trajectory prior. In Fig. 6, these contrastive directions are shown by the white arrows. The correction is weighted by the inverse KNN distance, so the smaller the KNN distance to the demonstration trajectory of a hijacking task, the more strongly it contributes to the final correction. Following the core update in Eq. (2), we obtain

$$a_{\mathrm{final}} = a_{\mathrm{tgt}} + \lambda \sum_{h_j \in \mathcal{H}_t} \frac{1}{d_{h_j}(p_t)} \big(a_{\mathrm{tgt}} - a_{h_j}\big), \tag{5}$$

where $\mathcal{H}_t$ contains the selected hijacking tasks, $d_{h_j}(p_t)$ is the KNN distance from p_t to the trajectory point cloud D_{h_j} of task h_j, and $\lambda > 0$ controls the overall steering strength.

The resulting action a_{final} is shown by the red arrow. Compared with the original target-conditioned action a_{tgt}, the corrected action is steered away from the bowl-related and cream-cheese-related trajectory priors and points more directly toward the stove knob. In this way, CAS helps the policy escape the interfering trajectory prior induced by the source-side behavior and redirects execution toward the switched target task, *turn on the stove*.

Let us now return to Figure 1 (c) and explain the example more concretely in terms of CAS. After the instruction switches to the target task, i.e., picking up the blue block, CAS first considers which non-target task may interfere with the current state. At this moment, the end-effector has already moved along the previous red-block trajectory, so the local trajectory prior associated with the red-block task is strong. The red-block task is therefore identified as a hijacking task. CAS then queries the policy twice at the same observation: once with the target instruction "pick up the blue block" and once with the counterfactual hijacking instruction "pick up the red block." The blue-conditioned action gives the policy's current attempt to follow the switched target, while the red-conditioned action reveals the direction induced by the interfering trajectory prior. CAS contrasts these two actions and adds the resulting correction to the target-conditioned action. In this example, the correction suppresses the red-block bias and shifts the final action toward the blue block, producing the green corrected action shown in the figure.

C Benchmark Construction Details

C.1 Complementary Online and Offline Protocols

We construct two primary simulation benchmarks, *LIBERO-Goal-Switch* and *CALVIN-Custom-Switch*, using complementary protocols for evaluating mid-execution instruction switching in VLA and WAM policies. We additionally introduce *CALVIN-Custom-Switch-Random* as a supplementary variant of *CALVIN-Custom-Switch*, in which object placements are broadly randomized to evaluate mid-execution instruction switching without fixed nominal object locations.

LIBERO-Goal-Switch adopts an *online* switching protocol. The evaluated policy first executes a source instruction and, at a designated switching step t_s, receives a new target instruction within the same rollout. The switching state is therefore produced naturally by the policy's own preceding execution, closely reflecting realistic instruction-switching scenarios in which a user changes the instruction while the robot is already acting. A minor consequence of this protocol is that different policies may make slightly different spatial progress along the source-task trajectory by the same switching step t_s, due to small differences in execution speed. This can introduce modest variations in switching difficulty across models. Nevertheless, comparisons between methods built on the same base policy remain well controlled, since they use the same pre-switch policy and any intervention is activated only after the instruction changes.

CALVIN-Custom-Switch, in contrast, adopts a controlled *offline* switching protocol. Each episode is initialized from a predefined intermediate state representing partial progress toward a source task, and the policy is then evaluated on whether it can complete the updated target instruction from that

state. All evaluated policies and methods receive exactly the same robot state, object configuration, and target instruction. This state-matched design removes differences caused by pre-switch execution speed and avoids small variations introduced by policy stochasticity before the switch, making comparisons across all evaluated models and methods strictly fair by construction.

The two primary benchmarks thus provide complementary views of instruction switching. *LIBERO-Goal-Switch* emphasizes realistic online adaptation from states naturally reached by the evaluated policy, whereas *CALVIN-Custom-Switch* isolates post-switch adaptation under identical physical conditions. *CALVIN-Custom-Switch-Random* retains the controlled offline protocol of *CALVIN-Custom-Switch* while removing its fixed nominal object locations, providing a complementary evaluation under broadly randomized object placements. In all benchmarks, success is determined by completion of the target task after the instruction change.

C.2 CALVIN-CUSTOM-SWITCH DETAILS

Scene and tasks. CALVIN-Custom-Switch is built on top of the CALVIN simulator. We design a tabletop scene containing three colored blocks—red, pink, and blue—and a press button. The benchmark contains four atomic tasks with fixed natural-language instructions: *"lift the red block"*, *"lift the pink block"*, *"lift the blue block"*, and *"press the button"*.

Demonstration generation. Since the evaluated pretrained policies do not directly solve these custom tasks, we construct a small demonstration dataset for finetuning. Demonstrations are generated by a rule-based scripted oracle. For each episode, object poses and the robot initial configuration are perturbed around nominal values, invalid configurations such as object collisions are rejected, and the scene is allowed to settle before execution.

For block-lifting tasks, the oracle moves above the target block, descends to grasp it, closes the gripper, and lifts it to a fixed height. For the button-pressing task, the oracle moves above the button and presses it downward with the gripper closed. We retain only successful rollouts verified by the CALVIN task oracle. In our experiments, we collect 50 demonstrations per task and use them to finetune the evaluated policies.

These 200 demonstrations contain only standard single-instruction episodes and no mid-execution instruction switches. Thus, the evaluated policies learn the four underlying manipulation tasks without being explicitly trained on the switching behavior evaluated by the benchmark.

Switching-state construction. To construct switching states, we sample intermediate robot configurations along a scripted reference path for each source task. The path moves from the default robot pose toward a pre-grasp or pre-contact pose and then toward the corresponding interaction pose. We sample a progress value along this path, solve for the corresponding robot joint configuration using inverse kinematics, and save the settled robot and scene state. Object poses are sampled near fixed nominal locations with small perturbations, and all states are generated with random seed 0.

We divide the reference path into early, middle, and late progress intervals. Using a 30-step reference schedule, these intervals are reported as the switching-step ranges $[9, 15]$, $[15, 21]$, and $[21, 27]$. These ranges indicate the progress of the saved state along the source-task reference path; they do not imply that the evaluated policy executes the source instruction online for the corresponding number of actions. States from later ranges lie further along the source-task behavior and generally make redirection toward a new target more difficult.

Evaluation protocol. For each switching range, we generate 20 object configurations and construct one intermediate state for each of the four possible source tasks. Each intermediate state is paired with the three distinct target tasks, yielding

$$20 \times 4 \times 3 = 240$$

instruction-switching episodes per range. The state-generation scripts additionally produce 80 diagonal source–target controls in which the instruction remains unchanged. These no-switch controls are excluded from the switch-only results in Table 1 and from the reported Switch Success Rate.

At evaluation time, the policy is initialized directly from the saved intermediate state and receives only the target instruction. It may execute at most 180 environment actions. An episode is considered successful only if the CALVIN task oracle verifies completion of the target task; completing the source task or another non-target task does not count as success. All policies and methods are evaluated from exactly the same saved states.

C.3 CALVIN-CUSTOM-SWITCH-RANDOM DETAILS

The standard CALVIN-Custom-Switch benchmark samples object poses as small perturbations around fixed nominal locations. As a result, demonstration trajectories of the same task tend to occupy relatively concentrated regions of the workspace, which may induce strong and spatially localized trajectory priors. This raises a natural question: does the difficulty of mid-execution instruction switching persist when such fixed object-centered spatial regularities are removed, and can CAS remain effective under randomized object placements?

To study this question, we construct *CALVIN-Custom-Switch-Random*, a qualitatively different variant in which block locations are sampled over a broader workspace region without fixed nominal centers. Unlike the concentration-controlled variants, which preserve task-specific nominal object locations and vary only the perturbation range, the *random* regime substantially changes the spatial organization of the resulting task trajectories, as illustrated in Fig. 13 (d). Consequently, task trajectories are no longer tightly associated with fixed object-centered regions of the workspace.

Randomized object placements and demonstrations. Instead of sampling block poses as perturbations around task-specific nominal locations, we uniformly sample the center of each block over the entire valid tabletop region. This removes the fixed spatial association between each block and a particular region of the workspace, while ensuring that sampled block locations remain within the valid manipulation area. We generate training demonstrations under this randomized placement distribution using the same scripted-oracle procedure as in CALVIN-Custom-Switch. For each of the four tasks, we collect 100 successful demonstration trajectories, yielding 400 training trajectories in total.

C.4 LIBERO-GOAL-SWITCH DETAILS

Task suite and shared initial condition. LIBERO-Goal-Switch is built on LIBERO-Goal, which contains ten language-conditioned manipulation tasks sharing the same scene layout. We use a common physical initial condition for every episode: the first official initial state associated with LIBERO-Goal task 0, whose instruction is *"open the middle drawer of the cabinet"*. This association is used only to identify the stored physical state; task 0 otherwise has no privileged role in the evaluation. Reusing the same state across all source–target pairs ensures that every online rollout starts from an identical robot configuration and object arrangement.

Online switching protocol. After loading the shared initial state, we execute ten no-op actions to allow the simulator to stabilize. These actions are not predicted by the policy and are not counted toward the switching step. We denote the resulting observation by V_0. The policy then receives the source instruction and executes exactly t_s environment actions, reaching the policy-dependent switching observation V_{t_s}. At this point, the active instruction is replaced by the target instruction. For the main evaluation, we consider

$$t_s \in \{10, 15, 20, 25\},$$

covering progressively later stages of source-task execution.

All evaluated policies predict action chunks rather than individual actions. At the switching boundary, any remaining source-conditioned actions in the current chunk are discarded, and the policy replans from V_{t_s} under the updated target instruction. For policies that additionally maintain context across action chunks, such as LingBot-VA, we reset the accumulated cross-chunk context at the switching boundary, so that the target phase starts from the current observation and the updated instruction without retaining source-conditioned temporal context. All evaluated methods use the unmodified base policy before the switch; CAS and other post-switch interventions are activated only after the target instruction is issued.

Table 6: Average LIBERO success rates (%) under reported clean evaluation and inference-time modality interventions. Results are averaged over LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long.

	Clean	Intervened Modality		
Model	Reported	Vision	Instruction	Proprioception
OpenVLA-OFT	97.1	0.0	60.3	41.7
π_0	94.2	1.7	19.3	7.9
$\pi_{0.5}$	96.9	1.9	55.6	–
Fast-WAM	97.6	0.0	46.0	86.6
LingBot-VA†	98.5	0.1	58.7	–

Note. Clean results are taken from the corresponding papers or released checkpoints rather than re-evaluated in our intervention runs. A dash indicates that the standard policy does not consume an independent proprioceptive input. †For LingBot-VA, only LIBERO-Long uses the official released checkpoint in our intervention experiments. LIBERO-Spatial, LIBERO-Object, and LIBERO-Goal use community-released suite-specific checkpoints.

Task pairs and success metric. We enumerate all ordered pairs of distinct source and target tasks. With ten tasks, each switching step contains

$$10 \times 9 = 90$$

online instruction-switching episodes. Source–target pairs are ordered, so switching from task i to task j and switching from task j to task i are treated as two distinct switching scenarios. Each ordered pair is evaluated once at each value of t_s, yielding a total of 360 episodes per policy.

After the switch, the policy may execute at most 300 environment actions. Success is determined solely by the official LIBERO completion condition for the target task; completing the source task or any other non-target task does not count as success. For each switching step, we report the mean success rate over all 90 ordered off-diagonal task pairs:

$$\text{Switch Success Rate}(t_s) = \frac{1}{90} \sum_{(i,j):\, i \neq j} \mathbf{1}[\text{target task } j \text{ is completed after switching from task } i] \,.$$

D Analyses of Mid-Execution Instruction Switching

D.1 Analysis of Visual Priors

D.1.1 Inference-Time Modality Interventions

Experimental setup. We evaluate inference-time modality interventions on LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. Each suite contains 10 tasks, and we evaluate all 50 official initial states for each task, resulting in 500 episodes per model–suite–intervention condition and 2,000 episodes per model–intervention condition across the four suites. For each policy, we preserve its native evaluation protocol and modify only the modality under study; all non-intervened inputs, environment dynamics, action generation and post-processing, rollout horizon, and success criteria remain unchanged. Thus, the intervention conditions are controlled within each policy, while different policies retain their respective standard inference procedures. No CAS steering or additional counterfactual policy queries are used in these experiments.

For the *vision intervention*, we replace all visual observations provided to the policy with the same spatially constant RGB image. The image has constant pixel value $(128, 118, 109)$, obtained from the channel-wise mean of a reference LIBERO observation. For policies that consume both a third-person view and a wrist-camera view, both visual inputs are replaced by this same constant image at every policy query. The original task instruction and, when applicable, the real-time proprioceptive state remain unchanged. The environment itself continues to evolve and render normally; only the visual observations supplied to the policy are replaced.

For the *instruction intervention*, every task-specific instruction is replaced by the fixed instruction "remain still.", while the visual observations and other available inputs remain unchanged. The environment task and its success criterion are not modified: an episode is still counted as successful only

Table 7: Suite-level LIBERO success rates (%) under reported clean evaluation and inference-time modality interventions. Each suite contains 10 tasks with 50 evaluation episodes per task.

Model	Spatial	Object	Goal	Long	Avg.
Clean (reported)					
OpenVLA-OFT	97.6	98.4	97.9	94.5	97.1
π_0	96.8	98.8	95.8	85.2	94.2
$\pi_{0.5}$	98.8	98.2	98.0	92.4	96.9
Fast-WAM	98.2	100.0	97.0	95.2	97.6
LingBot-VA†	98.5	99.6	97.2	98.5	98.5
Vision intervention					
OpenVLA-OFT	0.0	0.0	0.0	0.0	0.0
π_0	0.0	0.0	6.8	0.0	1.7
$\pi_{0.5}$	0.0	0.0	7.4	0.0	1.9
Fast-WAM	0.0	0.0	0.0	0.0	0.0
LingBot-VA	0.4	0.0	0.0	0.0	0.1
Instruction intervention					
OpenVLA-OFT	58.4	96.2	9.2	77.4	60.3
π_0	26.2	30.2	1.6	19.0	19.3
$\pi_{0.5}$	67.4	66.6	12.8	75.6	55.6
Fast-WAM	40.0	86.6	4.2	53.2	46.0
LingBot-VA	82.6	58.0	9.8	84.2	58.7
Proprioception intervention					
OpenVLA-OFT	7.2	34.6	31.4	93.4	41.7
π_0	6.2	4.0	17.0	4.4	7.9
Fast-WAM	82.8	89.8	95.6	78.2	86.6

Note. Clean results are taken from the corresponding papers or released checkpoints rather than re-evaluated in our intervention runs. For the intervention experiments, all non-intervened inputs and model-specific inference settings are kept unchanged within each policy. $\pi_{0.5}$ and LingBot-VA are omitted from the proprioception block because their standard inference configurations do not consume an independent proprioceptive input. †For LingBot-VA, only LIBERO-Long uses the official released checkpoint in our intervention experiments. LIBERO-Spatial, LIBERO-Object, and LIBERO-Goal use community-released suite-specific checkpoints.

if the policy completes the original LIBERO task. Therefore, high success under this intervention indicates that task-specific behavior can persist despite removing the original task-specific language, whereas low success alone does not imply that the robot actually remains stationary.

For the *proprioception intervention*, we freeze the proprioceptive input at its value from the first policy query of each episode. After the model's standard stabilization procedure, we record the 8-D state consisting of the end-effector position, end-effector orientation, and gripper joint positions, and repeatedly provide this fixed state at all subsequent policy queries. The physical robot state and visual observations continue to evolve normally. This intervention therefore removes online proprioceptive updates rather than removing proprioception entirely. We apply it to OpenVLA-OFT, π_0, and Fast-WAM. The standard inference configurations of $\pi_{0.5}$ and LingBot-VA do not consume an independent proprioceptive input, so no corresponding intervention is reported for these models.

Model-specific evaluation details. For Fast-WAM, we use the base `libero_uncond_2cam224` checkpoint used by the official LIBERO evaluation, rather than an optional IDM variant. We follow its native evaluation protocol with random seed 42, 30 stabilization steps after environment reset, replanning every 10 environment steps, and an action horizon of 32. The same checkpoint and inference settings are used across the vision, instruction, and proprioception interventions, with only the intervened modality changed.

For LingBot-VA, checkpoint availability differs across LIBERO suites. In our intervention experiments, LIBERO-Long uses the official released `lingbot-va-posttrain-libero-long` checkpoint. Official released checkpoints for the other three suites were not available in the evaluation setup we used; therefore, LIBERO-Spatial, LIBERO-Object, and LIBERO-Goal are evaluated

using community-released suite-specific LingBot-VA checkpoints. All LingBot-VA intervention conditions within a given suite use the same corresponding checkpoint. We retain the model's native inference mechanism, including its action/video history and KV cache, and intervene only on the external input modality under study. LingBot-VA does not consume robot proprioception as an independent input in this inference configuration.

Modality intervention results. The modality interventions reveal a strikingly consistent dependence on visual input. Under the reported clean settings, all five policies achieve high average LIBERO success rates, ranging from 94.2% to 98.5%. In contrast, replacing the visual observations with a spatially constant image reduces the average success rate to at most 1.9% for every evaluated policy. OpenVLA-OFT and Fast-WAM drop to 0.0%, LingBot-VA to 0.1%, and π_0 and $\pi_{0.5}$ to 1.7% and 1.9%, respectively. This collapse is also highly consistent across individual suites: almost all model–suite combinations achieve zero success under the vision intervention, with the main residual successes occurring on LIBERO-Goal for π_0 and $\pi_{0.5}$, which achieve 6.8% and 7.4%, respectively. We inspect these successful rollouts and find that they do not resemble normal task-conditioned execution. Instead, the policy repeatedly produces nearly the same motion, continuously translating toward the stove while rotating in a nearly fixed direction. This repetitive motion occasionally contacts the stove knob and accidentally satisfies the "turn on the stove" success condition. The residual success under vision intervention therefore largely reflects coincidental task completion rather than coherent execution based on the visual scene.

Replacing the task instruction produces a markedly different outcome. Substantial task success remains for several policies even though they no longer receive the original task-specific language. OpenVLA-OFT, $\pi_{0.5}$, Fast-WAM, and LingBot-VA retain average success rates of 60.3%, 55.6%, 46.0%, and 58.7%, respectively, while π_0 retains 19.3%. The residual performance is strongly suite-dependent. For example, OpenVLA-OFT still achieves 96.2% on LIBERO-Object and 77.4% on LIBERO-Long, while LingBot-VA achieves 82.6% on LIBERO-Spatial and 84.2% on LIBERO-Long. In contrast, success under the instruction intervention is much lower on LIBERO-Goal for all models, ranging from 1.6% to 12.8%. These results do not imply that language is unimportant; rather, they show that removing the original task-specific instruction does not necessarily eliminate recognizable task-like manipulation behavior.

The proprioception intervention exhibits substantially greater variation across policies. OpenVLA-OFT retains 41.7% average success, π_0 drops to 7.9%, whereas Fast-WAM remains at 86.6%. Thus, the contribution of online proprioceptive updates appears strongly policy-dependent. Notably, π_0 is also sensitive to the proprioception intervention, suggesting that current robot-state information can play an important role in its behavior in addition to visual input. This observation is compatible with our KNN-based design in CAS, which uses the current end-effector position to estimate proximity to task trajectories.

In contrast, the near-complete failure under visual intervention is remarkably consistent across all five evaluated architectures.

Interpreting residual success under instruction intervention. We further inspect why substantial success can remain under the instruction intervention. Except for LIBERO-Goal, the other three LIBERO suites contain visual scene or configuration cues that can often provide information about which task is likely to be executed. Such visual cues may therefore partially explain the relatively high residual success rates when task-specific language is replaced. Although these visual cues can be task-discriminative, they do not necessarily drive the policy toward the very task behavior associated with the original environment task. For example, by inspecting rollouts on LIBERO-Object, we observe many cases in which the policy continues to exhibit a recognizable task-like manipulation behavior after the instruction is replaced by "remain still.", but the resulting behavior corresponds to a different task from the one originally specified.

Importantly, our interpretation that robot policies are strongly influenced by visual priors is not based solely on the success-rate differences across modality interventions. Success rate only indicates whether the resulting behavior happens to satisfy the original task predicate; it does not fully characterize what behavior the policy is attempting to execute. A failed episode under instruction intervention may still contain a highly recognizable task-like manipulation behavior, whereas, as dis-

cussed above, a successful episode under vision intervention may result from accidental interaction with the environment.

Behavioral analysis. Our main evidence also comes from direct inspection and visualization of policy behavior. Consistent with the observation in Sec. 2.2, under instruction intervention, policies frequently continue to exhibit recognizable task-like manipulation behaviors, as if still executing a plausible task. This remains true even in cases where the resulting behavior corresponds to a task different from the one originally specified. In contrast, under vision intervention, such task-likeness is substantially degraded, manifesting as erratic motions, repetitive motion along a nearly fixed direction, or movements with noticeably reduced magnitudes.

Below, we visualize representative rollouts for all five evaluated policies under three conditions: clean inference, instruction intervention, and vision intervention (Figs. 7, 8, 9, 10, and 11). We visualize each rollout using five frames sampled every 20 environment steps, at steps 0, 20, 40, 60, and 80. At 20 fps, these frames correspond to video timestamps 0, 1, 2, 3, and 4 seconds, respectively. These examples illustrate the contrast in behavioral task-likeness across input conditions.

For OpenVLA-OFT (Fig. 7), the rollout under instruction intervention closely resembles the clean rollout, with the robot successfully reaching and grasping the correct object in both cases. Under vision intervention, however, the gripper instead repeatedly translates toward the right without exhibiting a recognizable manipulation behavior. For π_0 (Fig. 8), the instruction-intervened policy still reaches and grasps the correct object, although it does not successfully complete the lift. In contrast, under vision intervention, the gripper moves only slowly toward the right. Although this direction is broadly consistent with motion toward the target, the movement magnitude is substantially reduced compared with normal task execution. For $\pi_{0.5}$ (Fig. 9), the first grasp attempt under instruction intervention is slightly misaligned, but the policy nevertheless approaches and continues interacting with the correct target. Its behavior under vision intervention is qualitatively different, exhibiting erratic motion without a clear task-directed manipulation pattern. Fast-WAM (Fig. 10) shows a similar pattern under instruction intervention: despite an initially misaligned grasp, the robot continues to approach the correct target. With vision intervention, however, it repeatedly moves toward the basket while simultaneously rotating, rather than carrying out a recognizable task-specific manipulation sequence. Finally, for LingBot-VA (Fig. 11), the instruction-intervened policy exhibits a particularly informative failure mode: it successfully grasps an object, but not the one specified by the original task, effectively carrying out the behavior of a different task. Under vision intervention, the gripper instead initially drifts toward the right and slightly downward.

Across these examples, replacing the task instruction does not necessarily eliminate structured manipulation behavior: the policy may still approach and interact with meaningful objects, successfully grasp the intended object, or even execute a coherent but different task. In contrast, replacing visual input substantially degrades this task-likeness, producing behaviors such as nearly unidirectional drift, greatly reduced motion, erratic movement, or repetitive translation and rotation. These qualitative observations complement the success-rate results and provide direct behavioral evidence for the strong influence of visual priors on policy execution.

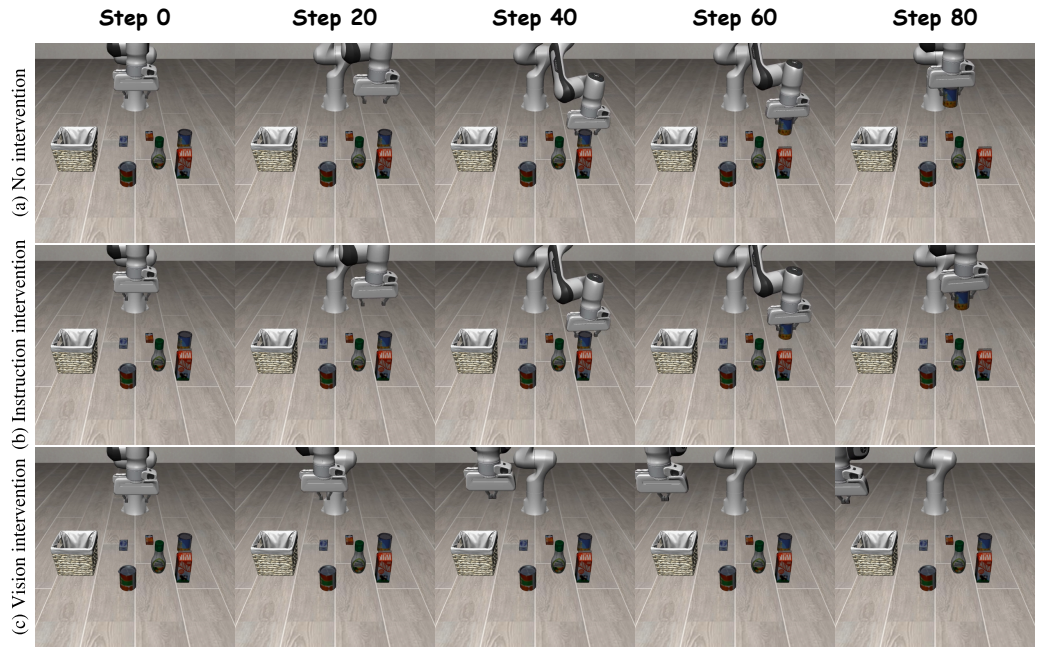


Figure 7: **Representative rollout sequences for OpenVLA-OFT.** Under instruction intervention, the policy behaves similarly to clean inference and successfully grasps the correct object, whereas under vision intervention the gripper repeatedly translates toward the right without exhibiting a recognizable manipulation behavior.

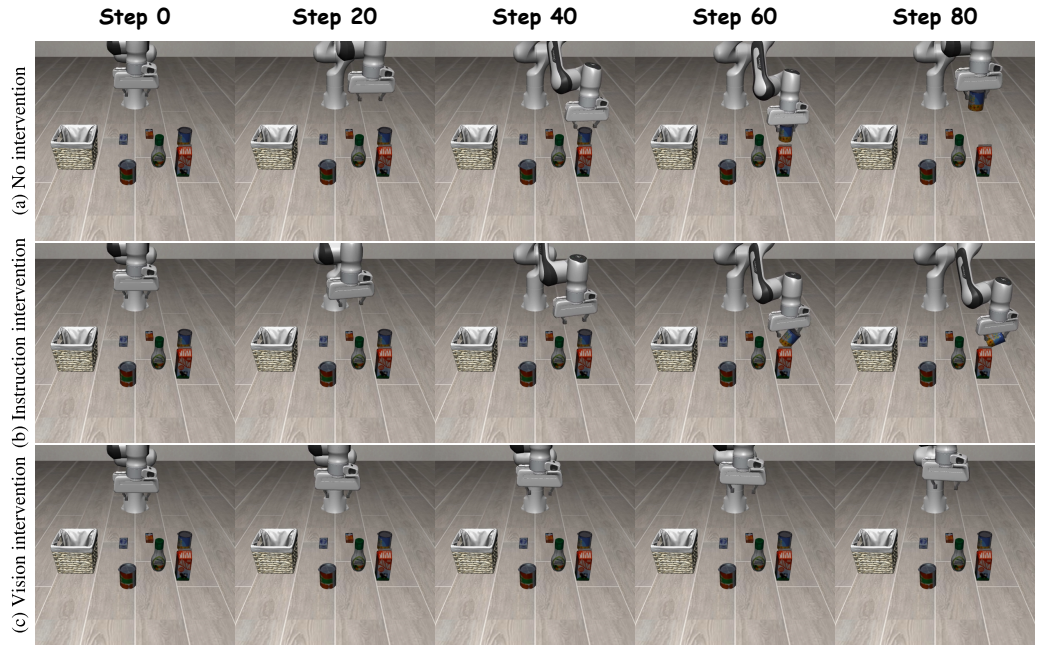


Figure 8: **Representative rollout sequences for π_0.** Under instruction intervention, the policy still reaches and grasps the correct object, although it does not complete the lift. Under vision intervention, the gripper moves slowly toward the right with substantially reduced motion magnitude.

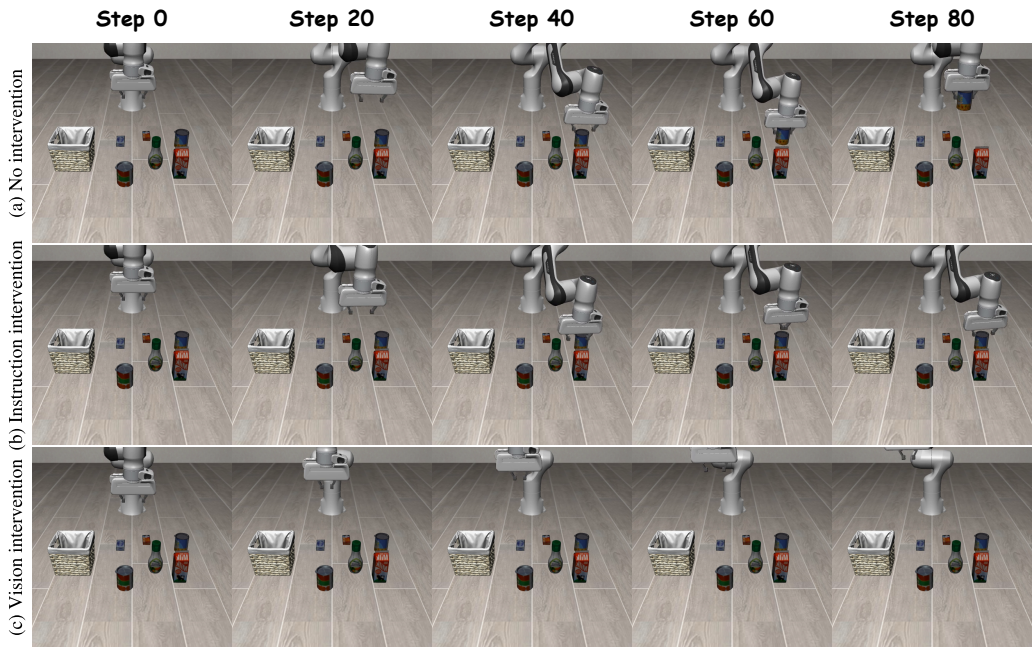


Figure 9: **Representative rollout sequences for $\pi_{0.5}$.** Under instruction intervention, the first grasp attempt is slightly misaligned, but the policy still approaches and interacts with the correct target. Under vision intervention, the policy instead exhibits erratic motion without a clear task-directed manipulation pattern.

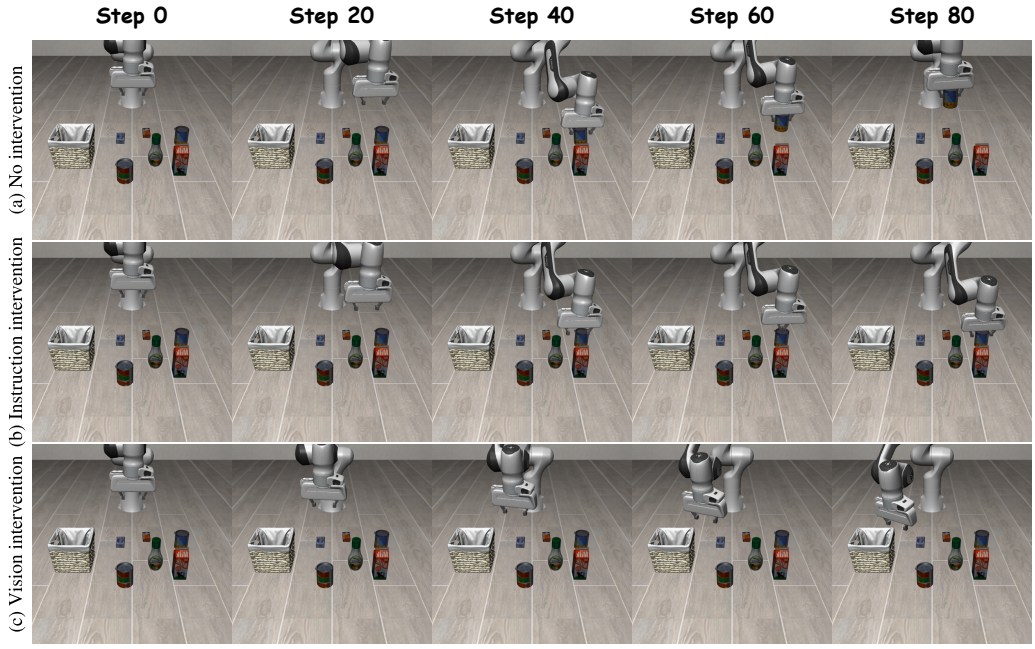


Figure 10: **Representative rollout sequences for Fast-WAM.** Under instruction intervention, the first grasp attempt is misaligned, but the policy continues to approach the correct target. Under vision intervention, the robot repeatedly moves toward the basket while rotating, substantially degrading the recognizable task-like manipulation behavior.

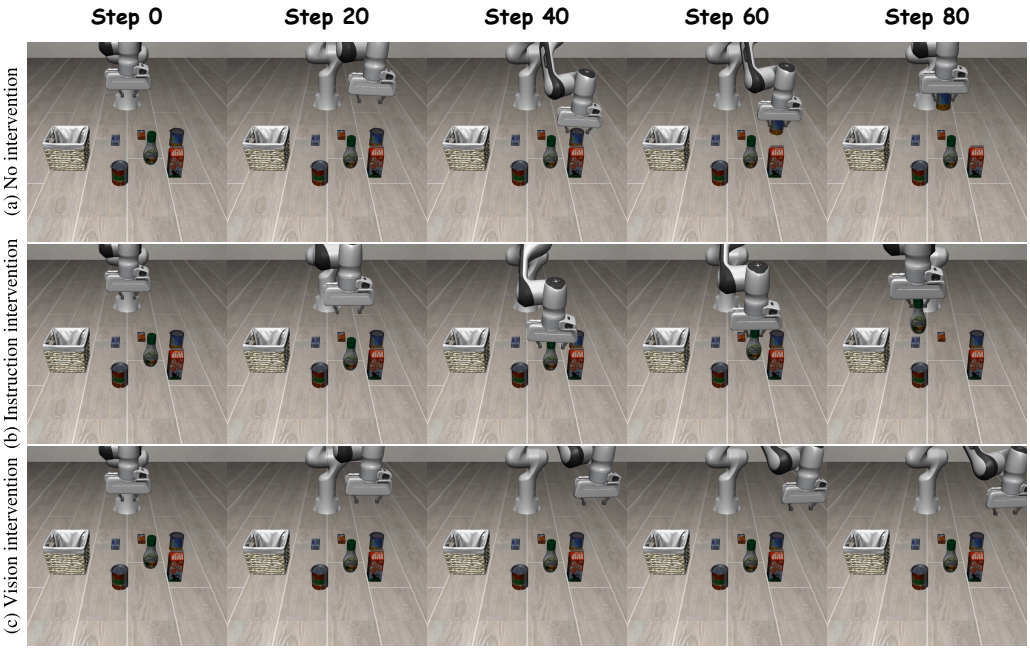


Figure 11: **Representative rollout sequences for LingBot-VA.** Under instruction intervention, the policy still executes a coherent manipulation behavior but grasps an object corresponding to a different task. Under vision intervention, the gripper instead drifts toward the right and slightly downward without exhibiting comparable task-like structure.

D.1.2 MID-EXECUTION STOP TEST AFTER TRAINING ON THE "REMAIN STILL" INSTRUCTION

Experimental setup. We further test whether OpenVLA-OFT can reliably halt an ongoing behavior when the "remain still" instruction is explicitly included in training. After adding explicit remain-still supervision, we evaluate online instruction switching from each of the ten LIBERO-Goal source tasks to "remain still" at different stages of execution, and measure how much the robot continues to move after the switch.

Training with explicit remain-still supervision. We augment the OpenVLA-OFT training data with 100 explicit remain-still trajectories. Each trajectory contains 20 environment steps and is initialized from a robot state sampled across the workspace covered by LIBERO-Goal demonstrations, with the sampled states distributed across the ten LIBERO-Goal task contexts. Throughout each trajectory, the six end-effector translation and rotation action dimensions are set to zero, while the gripper is kept in its initial open state. Thus, the training data explicitly associates the instruction "remain still" with zero commanded end-effector motion.

To retain supervision for the original LIBERO-Goal tasks during this additional training, we also include 50 original LIBERO-Goal demonstration trajectories, with five demonstrations sampled from each of the ten tasks. We finetune OpenVLA-OFT on this mixed dataset and use the resulting Step-55,000 checkpoint for evaluation.

Online mid-execution stop evaluation. We evaluate the trained policy using an online instruction-switching protocol. For each of the ten LIBERO-Goal tasks, the policy first executes the corresponding source instruction from the common initial state. We then switch the instruction to "remain still" at

$$t_s \in \{5, 10, 15, 20, 25\},$$

resulting in $10 \times 5 = 50$ switching rollouts in total. At the switching boundary, any unexecuted source-conditioned actions in the current action chunk are discarded, and the policy immediately replans under the "remain still" instruction. We then continue execution for 100 environment steps. OpenVLA-OFT executes an open-loop action chunk of eight environment actions before each new policy query. All movement statistics below are computed only over the post-switch portion of the rollout.

Evaluation metrics. Let p_0 denote the end-effector position at the switching state and p_t its position after t post-switch environment steps. We measure two complementary quantities. First, the cumulative end-effector path length over a horizon H is

$$L_H = \sum_{t=1}^{H} \|p_t - p_{t-1}\|_2,$$

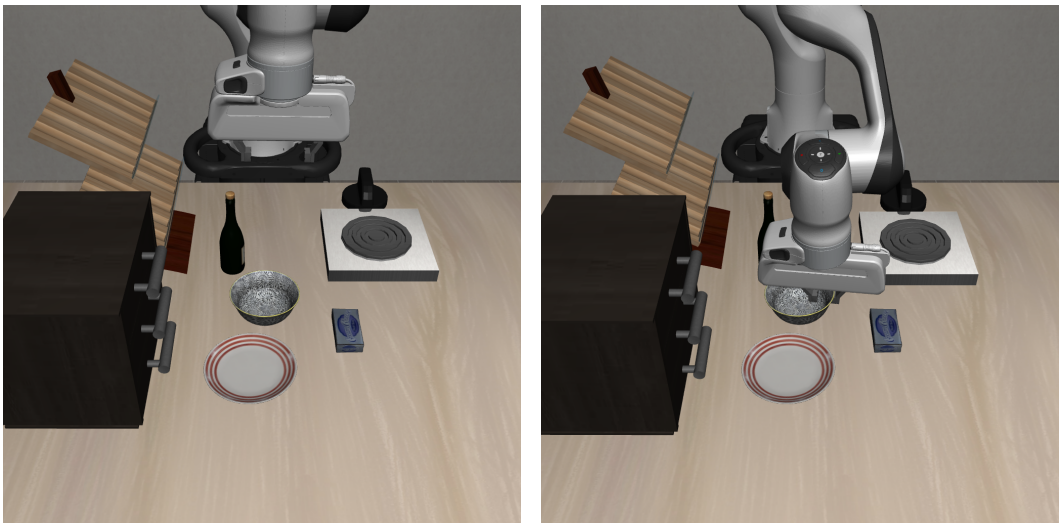
which measures the total amount of motion after the stop instruction is issued. Second, we measure the final displacement from the switching state,

$$D_H = \|p_H - p_0\|_2.$$

The latter distinguishes substantial displacement from small local oscillations around the switching state. A policy that reliably follows the "remain still" instruction should keep both quantities close to zero.

Results. Despite explicit remain-still supervision, the policy continues to move substantially after the instruction switch. Table 8 summarizes the post-switch motion over horizons of 20, 50, and 100 environment steps.

The 20-step horizon is particularly informative because it matches the length of the remain-still trajectories used during training. Even over this same horizon, the end effector travels 7.85 cm on average and ends 7.07 cm away from the switching state. The motion continues to accumulate over longer execution: after 50 steps, the average cumulative path reaches 20.44 cm, and after 100 steps it reaches 41.29 cm, with an average final displacement of 23.20 cm.



(a) At the instruction switch. (b) After 100 post-switch steps.

Figure 12: Representative mid-execution stop failure after explicit training on the "remain still" instruction. The source instruction is "put the bowl on the plate." At the switching state, the instruction is changed to "remain still" (left). Despite the updated instruction, the robot continues to move substantially over the following 100 environment steps (right), resulting in a cumulative end-effector path length of 30.74 cm and a final displacement of 23.81 cm from the switching state.

Table 8: Post-switch end-effector motion after switching to "remain still" following explicit remain-still training. Results are aggregated over 50 rollouts across the ten LIBERO-Goal source tasks and five switching steps.

	Cumulative Path Length L_H (cm)			Final Displacement D_H (cm)		
Post-Switch Horizon	Mean	Median	Std.	Mean	Median	Std.
20 steps	7.85	5.49	6.20	7.07	5.10	5.86
50 steps	20.44	18.37	14.07	12.68	12.45	7.23
100 steps	41.29	31.66	21.48	23.20	22.26	6.34

The large final displacement also shows that the measured motion cannot be explained solely by small action jitter around the switching state. At the 100-step horizon, every one of the 50 rollouts has both a cumulative path length and a final displacement greater than 10 cm. The minimum cumulative path length across all rollouts is 11.13 cm, and even the minimum final displacement is 10.04 cm.

Qualitative example. Figure 12 shows a representative rollout whose source instruction is "put the bowl on the plate." At the switching state, the robot has already progressed toward the bowl under the source instruction. The instruction is then changed to "remain still." Nevertheless, over the subsequent 100 environment steps, the robot moves substantially away from the switching state rather than remaining stationary. For this rollout, the cumulative end-effector path length is 30.74 cm and the final displacement from the switching state is 23.81 cm. Among rollouts with bowl-related source instructions, we select this example because its cumulative path length and final displacement are close to the aggregate levels across all 50 rollouts, rather than because it exhibits unusually large motion.

Interpretation. These results show that explicit remain-still supervision alone does not make the policy reliably stop under online instruction switching. Even after the policy is directly trained to associate "remain still" with zero commanded end-effector motion, substantial movement persists when the instruction is issued from states that already reflect partial progress along another task.

Table 9: **Wrong-task completion as a percentage of all switching episodes.** Each switching-range column reports the percentage of all 240 episodes in that range for which the target task fails but a non-target task is completed. Overall aggregates all four switching ranges.

Policy	$[3, 9]$	$[9, 15]$	$[15, 21]$	$[21, 27]$	Overall
OpenVLA-OFT	9.2	12.9	18.8	20.8	15.4
π_0	0.0	7.5	16.7	31.3	13.9
$\pi_{0.5}$	3.3	12.9	12.9	16.7	11.5
Fast-WAM	1.3	0.4	0.8	2.1	1.1
LingBot-VA	0.8	2.9	5.4	0.0	2.3

Table 10: **Wrong-task completion as a percentage of target-task failures.** Each switching-range column reports the percentage of target-task failures in that range for which the robot completes a non-target task. Overall aggregates target-task failures across all four switching ranges.

Policy	$[3, 9]$	$[9, 15]$	$[15, 21]$	$[21, 27]$	Overall
OpenVLA-OFT	11.8	14.0	20.8	22.6	17.5
π_0	0.0	8.5	17.0	31.4	17.0
$\pi_{0.5}$	9.5	15.2	13.7	16.7	14.6
Fast-WAM	2.0	0.4	0.8	2.1	1.3
LingBot-VA	2.5	4.4	5.8	0.0	3.1

Taken together with the modality-intervention results in Appendix D.1.1, which show that disrupting visual input substantially degrades structured task-like behavior, these findings support the interpretation that visual priors contribute to behavioral persistence during execution. In particular, visually grounded task-progress cues may favor continuation of an ongoing manipulation pattern, making the policy less responsive to updated instructions and thereby contributing to failures under mid-execution instruction switching.

D.2 Non-Target Task Completion in Failed Instruction Switching

We further quantify how often failed instruction-switching episodes result in the completion of a non-target task. We analyze the CALVIN-Custom online switching evaluations used in Observation 1, covering four switching ranges, $t_s \in [3, 9], [9, 15], [15, 21], [21, 27]$. Each policy is evaluated on 240 episodes per switching range, yielding 960 episodes in total.

We define a *wrong-task completion* as an episode in which the target task is not completed, but the robot completes a different task. This captures a strong form of trajectory hijacking in which the policy does not merely drift toward a non-target trajectory, but ultimately completes a non-target task. It is therefore a conservative indicator of trajectory hijacking, since partial drift toward a non-target trajectory need not result in completion of that task.

Tables 9 and 10 show that non-target task completion occurs across multiple policy architectures and switching ranges. For OpenVLA-OFT, π_0, and $\pi_{0.5}$, wrong-task completion accounts for 17.5%, 17.0%, and 14.6% of target-task failures overall, respectively. Thus, a non-negligible fraction of switching failures do not simply end without completing a task; instead, the robot completes a task other than the switched target.

For several policies, wrong-task completion also becomes more frequent at later switching stages. The trend is particularly pronounced for π_0, for which wrong-task completion rises from 0.0% of target-task failures in $[3, 9]$ to 31.4% in $[21, 27]$. OpenVLA-OFT shows a similar increase from 11.8% to 22.6%. This pattern is consistent with later switching states lying further along the source-task behavior, where redirecting execution toward a new target becomes more challenging.

The frequency of complete wrong-task execution varies substantially across policies, with Fast-WAM and LingBot-VA exhibiting this outcome less frequently. Importantly, a low wrong-task completion rate does not imply that the remaining switching failures are unrelated to trajectory hijacking. Our definition of trajectory hijacking also includes partial drift toward a non-target trajectory, which

Table 11: **Inference-time efficiency on CALVIN-Custom-Switch with π_0.** We compare the base policy, CAS, and PCD over the same 240 episodes for $t_s \in [15, 21]$ on a single NVIDIA H100 GPU, with a maximum episode length of 180 environment actions. CAS is evaluated using serial counterfactual policy queries. *Avg. Inference Latency per Replan* sums the wall-clock time of all policy queries issued within one replan. Values in parentheses denote the inference latency relative to the base policy.

Method	Avg. Episode Time	Avg. Inference Latency per Replan
Base	14.14 s	102.2 ms (1.00×)
+ CAS	12.13 s	157.4 ms (1.54×)
+ PCD	38.15 s	788.3 ms (7.71×)

may disrupt target-task execution without culminating in completion of another task. Overall, these results provide quantitative evidence that complete execution of a non-target task is a recurring failure pattern in mid-execution instruction switching, complementing the qualitative examples in Fig. 3.

E Additional Experiments and Analyses of Inference-Time CAS

E.1 Inference-Time Efficiency Analysis

We further analyze the inference-time efficiency of CAS on CALVIN-Custom-Switch using π_0 for $t_s \in [15, 21]$. We compare the base policy, CAS, and PCD over the same 240 evaluation episodes on a single NVIDIA H100 GPU. All methods use the same maximum episode length of 180 environment actions.

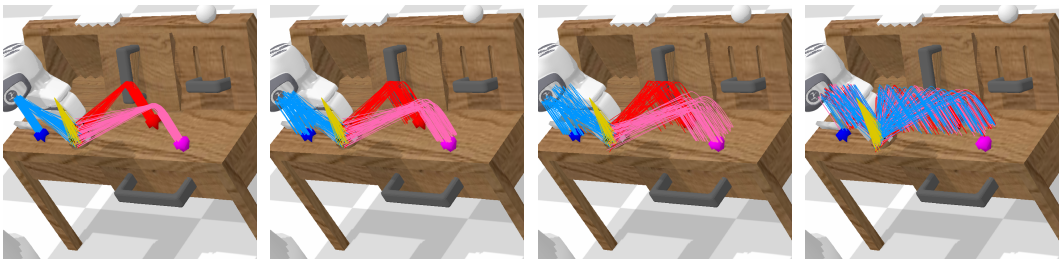
Efficiency of CAS. CAS may query the policy under multiple task instructions at the same state, including the target instruction and the instructions of selected hijacking tasks. These instruction-conditioned action predictions are independent and can therefore be issued in a single batched policy call. In principle, such batching can substantially reduce the additional sequential policy-query latency introduced by counterfactual queries. In our current implementation, however, we use a simpler serial implementation and query the policy separately for different task instructions.

Even without such parallelization, CAS introduces relatively modest additional policy-query cost. As shown in Table 11, CAS has a lower average episode time, 12.13 s compared with 14.14 s for the base policy. Since all methods use the same maximum episode length of 180 environment actions, this lower average episode time is consistent with CAS more frequently completing the target task before reaching the episode horizon.

The reported per-replan timing measures the accumulated latency of all policy queries within each replan, including the base target-conditioned query and any additional counterfactual queries. We therefore report this metric as *Avg. Inference Latency per Replan*. CAS increases the average inference latency per replan from 102.2 ms for the base policy to 157.4 ms, corresponding to only $1.54\times$ the base latency. In comparison, PCD has an average inference latency of 788.3 ms per replan, or $7.71\times$ the base latency.

Across all CAS replans in this evaluation, CAS makes 1.60 policy queries on average, including the target-conditioned query. Equivalently, in addition to the single target-conditioned query required by the base policy, CAS introduces only 0.60 counterfactual policy queries per replan on average. This low query count results from the selective nature of CAS: rather than querying every non-target task, CAS only queries tasks whose trajectory distributions are sufficiently close to the current state and thus exert strong interfering priors.

Scaling to larger task sets. One potential concern is that, under a serial implementation, the policy-query cost of CAS could increase if many counterfactual tasks need to be queried. Importantly, however, the number of counterfactual policy queries is determined by the number of locally interfering trajectories rather than directly by the total number of tasks. Even as the overall task set grows, CAS only queries non-target tasks whose KNN distance to the current state falls below the



(a) **Narrow** (b) **Medium** (c) **Wide** (d) **Random**

Figure 13: Visualization of task trajectory distributions under different object-pose regimes in CALVIN-Custom-Switch. In *narrow*, *medium*, and *wide*, object poses are sampled as perturbations around fixed nominal centers, with progressively larger perturbation ranges producing increasingly dispersed trajectories. The *random* regime instead samples object positions over a broader workspace region without fixed nominal centers, resulting in a qualitatively different trajectory structure.

selection threshold. Therefore, the number of counterfactual queries need not grow proportionally with the size of the task set.

In settings where many trajectories strongly overlap around the current state, the computational budget can be further controlled by reducing the KNN distance threshold δ, thereby retaining only the strongest interfering trajectory priors. The selected counterfactual queries can also be evaluated in parallel through batched policy inference, as discussed above. Finally, the lightweight Source-only CAS variant introduced in Sec. 3.2 uses only the known source task and avoids querying additional candidate tasks, while still providing substantial improvements (Sec. 4.4).

E.2 Generality across Diverse Trajectory Regimes

The strength and structure of trajectory priors may depend on the object-pose distribution used to generate demonstrations. When object poses vary only slightly around fixed nominal locations, demonstrations tend to form narrow and highly concentrated trajectory distributions. As object-pose variation increases, the resulting trajectories become more dispersed, potentially changing the interference patterns among tasks. A qualitatively different setting arises when object positions are sampled over a broad workspace region without fixed nominal centers, where the resulting task trajectories no longer follow the same object-centered spatial structure.

To evaluate whether CAS depends on a particular trajectory geometry, we consider two complementary variations of CALVIN-Custom-Switch. First, we construct three concentration-controlled regimes, *narrow*, *medium*, and *wide*, by progressively increasing the object-pose perturbation range around fixed nominal centers. Second, we construct a *random* regime in which object positions are sampled over a broader workspace region without fixed nominal centers. For each regime, the object-pose distribution used for training is aligned with that used for evaluation, so these experiments isolate the effect of demonstration-trajectory structure rather than train–test distribution shift. Fig. 13 visualizes the resulting trajectory distributions.

E.2.1 Varying Trajectory Concentration

We first study whether CAS depends on narrow and highly concentrated trajectory distributions. The *narrow*, *medium*, and *wide* settings share the same nominal object locations but use progressively larger object-pose perturbation ranges. The *narrow* setting corresponds to CALVIN-Custom-Switch-Fixed used in the main experiments and uses the smallest perturbation range. The *medium* and *wide* settings introduce increasingly larger object-pose variation, producing progressively more dispersed task trajectories, as illustrated in Fig. 13 (a)–(c).

Table 12 reports the results across the three trajectory-concentration regimes. CAS consistently improves switching success across all concentration levels and switching ranges. For early switches in the $[9, 15]$ range, CAS improves success from 22.1% to 73.8% in the narrow setting, from 12.9%

Table 12: Success rates (%) of π_0 on CALVIN-Custom-Switch under different levels of trajectory concentration. The *narrow* setting corresponds to CALVIN-Custom-Switch-Fixed used in the main experiments.

Switch range	Method	Trajectory concentration		
		Narrow	Medium	Wide
$[9, 15]$	π_0	22.1	12.9	12.1
	π_0+CAS	**73.8**	**62.9**	**65.8**
$[15, 21]$	π_0	6.7	2.1	5.4
	π_0+CAS	**45.4**	**40.8**	**43.8**
$[21, 27]$	π_0	0.0	0.0	0.0
	π_0+CAS	**20.4**	**8.8**	**10.4**

Table 13: Success rates (%) under the random trajectory regime on CALVIN-Custom-Switch-Random. Object positions are randomized over a broader workspace region rather than sampled as perturbations around nominal centers.

Switch range	Policy	Base	CAS
$[9, 15]$	OFT	1.7	**4.6**
	π_0	33.3	**60.0**
	$\pi_{0.5}$	44.6	**65.4**
	Fast-WAM	38.7	**52.9**
	LingBot-VA	33.7	**34.5**
$[15, 21]$	OFT	1.3	**2.1**
	π_0	8.8	**34.2**
	$\pi_{0.5}$	22.5	**50.4**
	Fast-WAM	19.1	**38.7**
	LingBot-VA	10.4	**13.7**
$[21, 27]$	OFT	1.3	**1.7**
	π_0	2.1	**14.2**
	$\pi_{0.5}$	6.3	**37.1**
	Fast-WAM	4.5	**22.0**
	LingBot-VA	1.2	**7.0**

to 62.9% in the medium setting, and from 12.1% to 65.8% in the wide setting. The gains remain substantial for later and more challenging switches. In the $[15, 21]$ range, CAS improves success by 38.7, 38.7, and 38.4 percentage points in the narrow, medium, and wide settings, respectively. In the latest $[21, 27]$ range, the base policy fails across all three regimes with 0.0% success, whereas CAS still achieves 20.4%, 8.8%, and 10.4% success.

These results show that the effectiveness of CAS is not restricted to highly concentrated trajectory distributions. CAS remains effective as demonstrations become increasingly dispersed, suggesting that its benefit does not depend on a particular level of trajectory concentration.

E.2.2 Randomized Object Placements

We next consider a qualitatively different trajectory regime in which object positions are sampled over a broader workspace region without fixed nominal centers. Unlike the concentration-controlled settings above, which preserve task-specific nominal object locations while varying only the perturbation range, the *random* regime substantially changes the spatial organization of the resulting task trajectories, as illustrated in Fig. 13 (d). This setting provides a stronger test of whether CAS depends on object-centered spatial regularities or a particular trajectory geometry.

Table 14: Switch-only success rates (%) under scene appearance shifts for π_0. Each variant contains 600 episodes covering 12 ordered off-diagonal source–target pairs with 50 repetitions per pair. The CAS steering strength is selected on D0 and kept fixed for D1–D3.

Scene variant	π_0	$\pi_0 + \text{CAS}$
D0 (original)	13.2	**55.0**
D1 (mild)	13.7	**55.8**
D2 (moderate)	14.5	**55.2**
D3 (strong)	13.7	**53.8**

Table 13 reports the results under the random trajectory regime across five policy architectures. CAS improves switching success over the corresponding base policy for every evaluated policy and switching range.

For π_0, CAS improves success from 33.3% to 60.0% in the $[9, 15]$ range, from 8.8% to 34.2% in the $[15, 21]$ range, and from 2.1% to 14.2% in the $[21, 27]$ range. For $\pi_{0.5}$, the corresponding improvements across the three ranges are from 44.6% to 65.4%, from 22.5% to 50.4%, and from 6.3% to 37.1%, respectively. CAS also consistently improves Fast-WAM, increasing success from 38.7% to 52.9%, from 19.1% to 38.7%, and from 4.5% to 22.0% across the three progressively later switching ranges. For LingBot-VA, the gain is modest in the $[9, 15]$ range, from 33.7% to 34.5%, but becomes more pronounced in the later ranges, improving from 10.4% to 13.7% in $[15, 21]$ and from 1.2% to 7.0% in $[21, 27]$. Although the absolute improvements for OFT are smaller, CAS still improves over the corresponding baseline in all three switching ranges.

These results show that CAS does not rely on task trajectories being organized around fixed nominal object locations. Even when object placements are broadly randomized and the resulting trajectory geometry changes substantially, CAS consistently improves switching performance across multiple policy architectures. Together with the trajectory-concentration study above, these findings demonstrate that CAS remains effective across both increasingly dispersed trajectory distributions and qualitatively different randomized trajectory structures.

E.3 Robustness to Scene Appearance Shifts

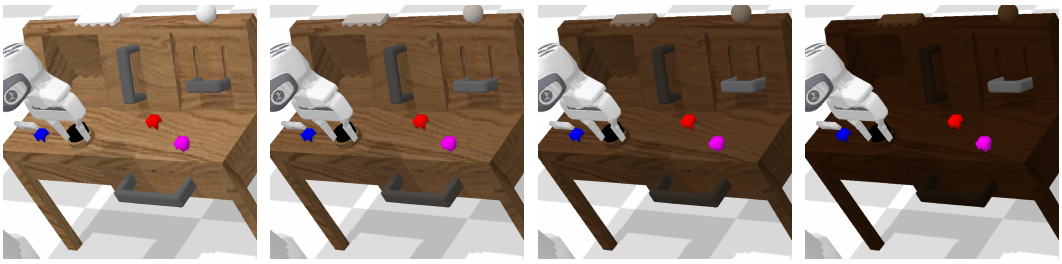
Visual trajectory priors may be affected by the visual consistency between training and evaluation environments. In our main experiments, the scene appearance at evaluation time matches that used during training. To test whether CAS remains effective under visual appearance shifts, we construct a separate scene-appearance-shift evaluation using π_0.

As shown in Fig. 14, we create four appearance variants based on the CALVIN-Custom-Switch scene. We denote the original training scene as *D0* and the progressively darkened variants as *D1–D3*. All four variants preserve the same task semantics, object layout, and scene geometry; only the appearance of the wood-textured background surfaces is changed. The policy is trained only on D0 and evaluated on all four variants.

To isolate the effect of scene appearance, we construct a separate set of state-matched switching episodes shared across D0–D3. Switching states are sampled at normalized reference progress $u \sim \mathcal{U}[0.4, 0.6]$, corresponding to nominal reference steps $[12, 18]$ on the same 30-step reference scale used in CALVIN-Custom-Switch. For each scene variant, we evaluate all 12 ordered off-diagonal source–target task pairs with 50 episodes per pair, yielding 600 switching episodes per variant. The same underlying robot states, object configurations, source–target assignments, and sampled reference-progress values are used across D0–D3; only the scene appearance differs. Success is determined solely by completion of the target task, and we report the switch-only success rate over these 600 episodes.

The CAS steering strength is selected using D0 and then kept fixed for all shifted variants D1–D3. Thus, no CAS hyperparameter is retuned for the appearance-shift conditions.

Table 14 reports the results. The base π_0 policy exhibits similar success rates across D0–D3, ranging from 13.2% to 14.5%. CAS substantially improves switching performance in every scene variant, with success rates remaining highly stable between 53.8% and 55.8% across appearance-



(a) D0 (original) (b) D1 (mild) (c) D2 (moderate) (d) D3 (strong)

Figure 14: Scene appearance variants used for evaluating robustness to visual train–test mismatch. D0 is the original CALVIN-Custom-Switch scene used for training in our main experiments, while D1–D3 progressively darken the background appearance. The task semantics, object layout, and scene geometry remain unchanged across variants.

Table 15: Robustness to inference random seeds on LIBERO-Goal-Switch. We evaluate $\pi_{0.5}$ at $t_s = 25$ over five random seeds. Each seed contains the same 90 ordered source–target task pairs. CAS consistently improves over the base policy across all evaluated seeds. Seed 7 corresponds to the setting used in the main evaluation.

Method	Seed 0	Seed 1	Seed 2	Seed 3	Seed 7	Mean $\pm$ Std.
Base	35.6	27.8	36.7	35.6	34.4	34.0 ± 3.6
+ CAS	37.8	46.7	47.8	45.6	44.4	$\mathbf{44.4 \pm 3.9}$

shift strengths. Notably, CAS achieves 55.0% on the original D0 scene and 53.8% even under the strongest D3 shift.

These results show that the effectiveness of CAS does not rely on exact visual matching between the training and evaluation scenes. Because the physical states, task assignments, and reference-progress samples are matched across D0–D3, the consistent improvements indicate that counterfactual action steering remains effective even when the visual appearance of the environment differs from that seen during training.

E.4 Robustness Across Evaluation Random Seeds

We further evaluate whether the improvement from CAS is robust to randomness during policy inference. We focus on $\pi_{0.5}$ on LIBERO-Goal-Switch at the challenging switching step $t_s = 25$. In addition to seed 7 used in the main evaluation, we repeat the same evaluation with four additional random seeds, $\{0, 1, 2, 3\}$. For each seed, we evaluate the same 90 ordered source–target task pairs, while keeping all other evaluation and CAS settings unchanged.

As shown in Table 15, CAS consistently outperforms the corresponding base policy across all five random seeds. Averaged over seeds $\{0, 1, 2, 3, 7\}$, the base policy achieves a success rate of 34.0%, while CAS achieves 44.4%, corresponding to an average improvement of 10.4 percentage points. The improvement is observed for every evaluated seed, indicating that the benefit of CAS is not specific to the particular random seed used in the main evaluation.

E.5 Additional Ablation Analysis

E.5.1 Steering-Weight Sweep for CAS Variants.

We sweep the fixed steering weight for the two simplified variants in Table 5. Specifically, we evaluate $w \in \{0.1, 0.2, 0.4, 0.6, 0.8\}$ for both Source-only CAS and All non-target Fixed. The complete results are shown in Fig. 15.

Remarkably, *every tested steering weight for both variants outperforms the base policy*. This consistent improvement across the entire sweep demonstrates that counterfactual steering provides a

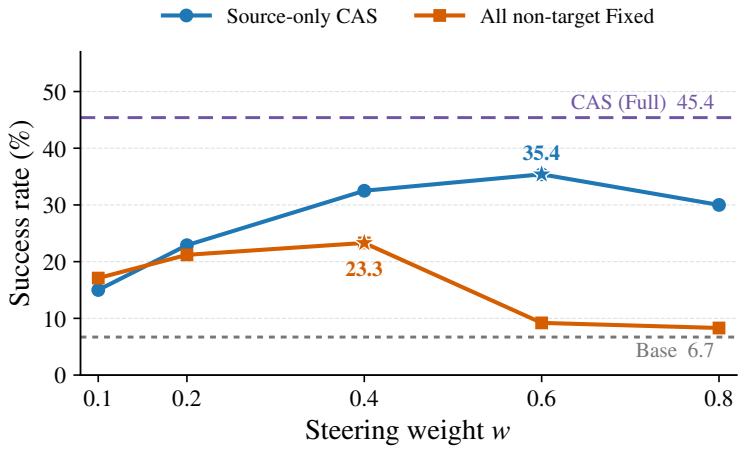


Figure 15: **Steering-weight sweep for CAS variants.** We vary the fixed steering weight $w \in \{0.1, 0.2, 0.4, 0.6, 0.8\}$ for Source-only CAS and All non-target Fixed. Remarkably, every tested setting of both variants outperforms the base policy. Source-only CAS achieves its best Switch SR of 35.4% at $w = 0.6$, while All non-target Fixed peaks at 23.3% at $w = 0.4$. Full CAS achieves 45.4%, outperforming all tested fixed-weight settings. Dashed horizontal lines show the base policy and full CAS for reference.

robust corrective signal rather than relying on a narrowly tuned steering strength. Source-only CAS achieves its best Switch SR of 35.4% at $w = 0.6$, while All non-target Fixed peaks at 23.3% at $w = 0.4$; these best-performing settings are reported in Table 5.

The sweep further reveals different sensitivity to the steering strength. Source-only CAS remains consistently effective across a broad range of w, further demonstrating the robustness of the lightweight source-only strategy. In contrast, All non-target Fixed degrades substantially as w becomes large, consistent with over-steering when the same correction strength is indiscriminately applied to multiple weakly relevant counterfactual tasks. Crucially, no tested global fixed steering weight closes the gap to full CAS: full CAS achieves a Switch SR of 45.4%, outperforming every fixed-weight setting of both simplified variants and exceeding the best one by 10.0 percentage points. These results show that the advantage of full CAS cannot be recovered by simply retuning a global steering strength, supporting the need for its state-dependent counterfactual steering design.

E.5.2 Hyperparameter Ablations

We further study the effect of the three key hyperparameters in CAS: the number of nearest neighbors K, the KNN distance threshold δ, and the steering strength λ. Specifically, K controls the local neighborhood used to estimate the distance between the current end-effector position and each task trajectory distribution, while δ determines which non-target trajectories are selected as potential hijacking tasks. The steering strength λ controls the overall magnitude of the counterfactual correction applied to the target-conditioned action. When $\lambda = 0$, the counterfactual correction vanishes and CAS reduces exactly to the base policy without action steering.

Unless otherwise specified, we use $K = 30$, $\delta = 0.1$, and $\lambda = 0.03$ as the default setting, which achieves a Switch SR of 45.4%. Starting from this configuration, we vary one hyperparameter at a time while keeping the other two fixed.

As shown in Fig. 16, CAS maintains substantial improvements over the base policy across a broad range of hyperparameter settings. Increasing λ from zero progressively strengthens counterfactual steering and leads to substantial gains in switching performance. In contrast, the performance remains highly stable across different choices of K, indicating that CAS does not require a precise neighborhood size for estimating local trajectory proximity. Varying δ changes how selectively CAS includes non-target trajectories as potential hijacking tasks, yet CAS remains substantially better than the base policy across all tested values. The best tested δ achieves a Switch SR of 49.2%.

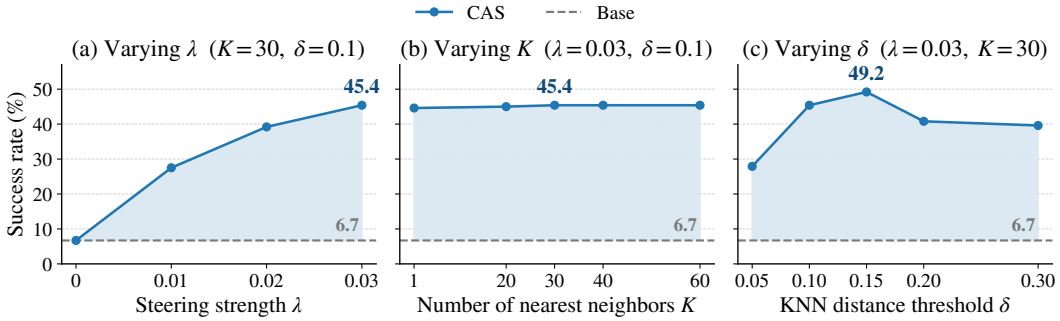
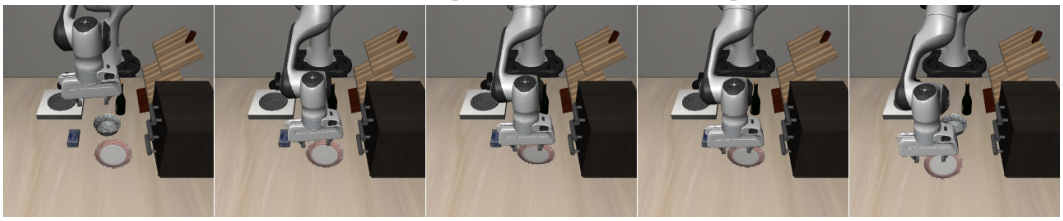


Figure 16: Hyperparameter ablations of CAS on CALVIN-Custom-Switch for $t_s \in [15, 21]$. Starting from the default setting $K = 30$, $\delta = 0.1$, and $\lambda = 0.03$, we vary one hyperparameter at a time while keeping the other two fixed: (a) steering strength λ, (b) number of nearest neighbors K, and (c) KNN distance threshold δ. The dashed line denotes the base policy without CAS.

(a) Without CAS: Suboptimal interaction causes the plate to tilt



(b) With CAS: Improved redirection enables a smoother push

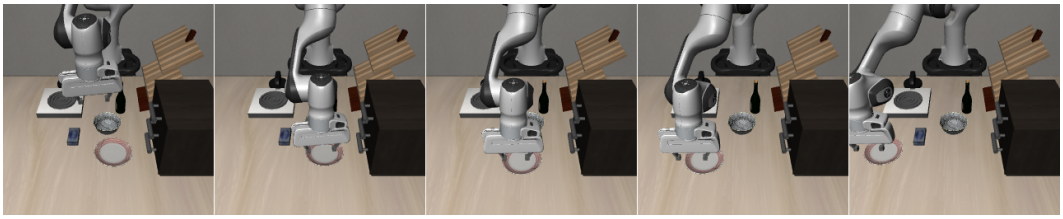


Figure 17: **Successful rollouts for a switch from *put the bowl on the stove* to *push the plate to the front of the stove*.** Both rollouts ultimately complete the target task. Without CAS, the policy remains more affected by the source-task trajectory prior and approaches the plate from a less suitable interaction position. As a result, one side of the plate lifts during the push, making the manipulation less smooth. With CAS, the gripper redirects toward a more suitable position for interacting with the plate and pushes it more smoothly toward the target location. Thus, although both rollouts succeed, CAS yields a cleaner post-switch manipulation trajectory.

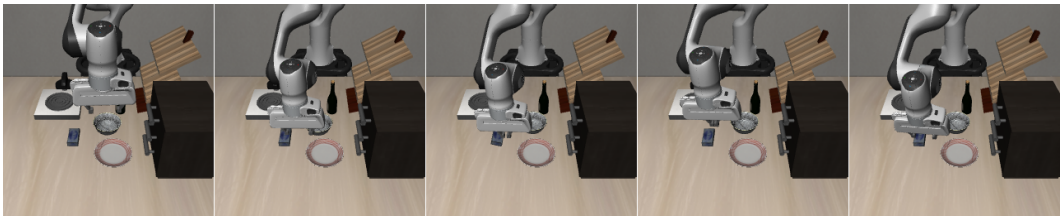
Overall, these results suggest that CAS is effective across a reasonably broad range of choices for K, δ, and λ, rather than relying on a narrowly tuned hyperparameter configuration.

F Additional Experiments and Analyses of CAS-Guided Self-Training

F.1 Qualitative Analysis of Higher-Quality CAS-guided Rollouts

To better understand why successful CAS-guided rollouts provide more effective supervision for self-training, we qualitatively compare successful rollouts collected with and without CAS during the one-round self-training data collection on LIBERO-Goal-Switch with $\pi_{0.5}$ at $t_s = 25$. We deliberately focus on cases where both rollouts ultimately complete the switched target task. These

(a) Without CAS: Insufficient redirection leads to a missed first grasp



(b) With CAS: Improved redirection enables a successful first grasp

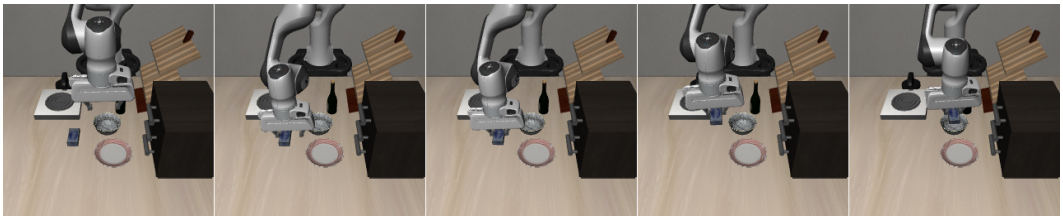
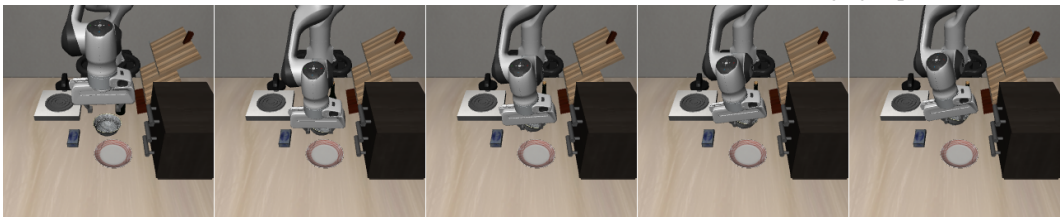


Figure 18: **Successful rollouts for a switch from *push the plate to the front of the stove* to *put the cream cheese in the bowl*.** Both rollouts eventually complete the target task. Without CAS, the initial post-switch redirection toward the cream cheese is insufficient, causing the first grasp attempt to miss. The policy must then re-approach the object and perform additional corrective motions before succeeding. With CAS, the gripper redirects more effectively toward the cream cheese and succeeds on the first grasp attempt, yielding a cleaner trajectory with fewer unnecessary intermediate actions.

(a) Without CAS: Insufficient redirection leads to an awkward front-edge grasp



(b) With CAS: Improved redirection enables a stable side grasp

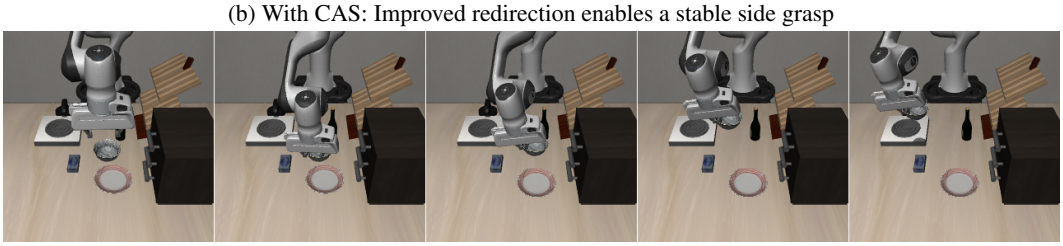


Figure 19: **Successful rollouts for a switch from *push the plate to the front of the stove* to *put the bowl on the stove*.** Both rollouts ultimately succeed. Without CAS, the policy does not redirect sufficiently toward a favorable grasp configuration and awkwardly catches the front edge of the bowl before lifting it. With CAS, the gripper is redirected toward a more suitable grasp pose and stably grasps the side of the bowl, resulting in more direct progress toward the updated task. Despite identical binary success outcomes, the two rollouts therefore exhibit clearly different post-switch manipulation quality.

examples therefore reveal differences in the quality of successful trajectories that are not captured by the binary task-success metric.

Figures 17–19 show three representative examples. In each figure, the first displayed frame corresponds to the instruction-switching state, followed by several key frames from the post-switch rollout. The frames shown with and without CAS are temporally aligned for direct comparison; the last displayed frame is not necessarily the final frame of the episode.

Case analysis. Figure 17 shows a switch from *put the bowl on the stove* to *push the plate to the front of the stove*. Although both rollouts eventually complete the target task, their post-switch behaviors differ substantially. Without CAS, the gripper does not sufficiently redirect toward a suitable interaction configuration for the plate. It consequently contacts and pushes the plate from an unfavorable position, causing one side of the plate to tilt upward and making the subsequent manipulation less smooth. With CAS, the counterfactual correction redirects the gripper toward a more suitable interaction position, leading to a smoother and more direct push toward the target location. Thus, even though both rollouts are successful, the CAS-guided rollout provides a cleaner transition from the switching state toward the updated task.

Figure 18 shows a switch from *push the plate to the front of the stove* to *put the cream cheese in the bowl*. Without CAS, the gripper does not redirect sufficiently toward the cream cheese. This results in a misaligned first grasp attempt, after which the policy must re-approach the object and perform additional corrective motions before eventually succeeding. With CAS, the gripper redirects more directly toward the cream cheese and completes the grasp on the first attempt. The resulting trajectory therefore avoids the failed intermediate attempt and unnecessary corrective motions observed without CAS.

Figure 19 shows another switch from *push the plate to the front of the stove*, this time to *put the bowl on the stove*. Without CAS, the gripper reaches the bowl from an awkward pose, catches its front edge, and lifts it using an undesirable grasp. With CAS, the gripper reaches a more suitable target-task grasp pose, allowing it to stably grasp the side of the bowl. The temporally aligned frames also show more direct progress toward the updated task in the CAS-guided rollout, consistent with its more appropriate post-switch redirection and grasp configuration.

Implications for self-training. Across all three examples, the policy ultimately completes the switched target task both with and without CAS. However, successful rollouts collected without CAS can still exhibit signs of trajectory hijacking, such as insufficient redirection, suboptimal interaction configurations, failed intermediate attempts, or unnecessary corrective motions. CAS mitigates these issues and produces more effective de-hijacking behavior by providing sufficient post-switch redirection toward the updated target task, even in episodes that would eventually succeed without CAS. These examples highlight an important distinction between *task success* and *trajectory quality*. Binary task success treats the paired rollouts above as equivalent, despite substantial differences in the post-switch behaviors they provide as supervision. In particular, CAS-guided successful rollouts provide cleaner examples of how the policy should redirect from a state influenced by a non-target trajectory prior toward the updated target task. This qualitative analysis therefore provides complementary evidence for the higher training value of CAS-guided rollouts, consistent with the self-training results in Table 2, where CAS-ST consistently outperforms Vanilla-ST with the same number of retained trajectories.

F.2 Instruction-Conditioned Action Visualization

To further analyze the effect of self-training on instruction-conditioned action predictions, we compare instruction-conditioned actions before and after two rounds of CAS-ST, as in Fig. 3 (d). Specifically, we query the original and two-round CAS-ST checkpoints from the same robot states and visualize the actions predicted under different LIBERO-Goal task instructions. The robot states are sampled from intermediate states encountered during task-execution rollouts. To avoid clutter from displaying all ten tasks simultaneously, we select the same four tasks as in Fig. 3 (c,d): *put the wine bottle on top of the cabinet*, *put the bowl on top of the cabinet*, *put the cream cheese in the bowl*, and *turn on the stove*. To provide a visual reference for the task-consistent motion directions, we additionally show one official demonstration trajectory for each selected task in Fig. 20.

These trajectories are intended to provide only a coarse reference for interpreting the action arrows below. In particular, the visualized arrows correspond to 3D end-effector translational actions projected onto the 2D camera plane, and thus motion components perpendicular to the image plane

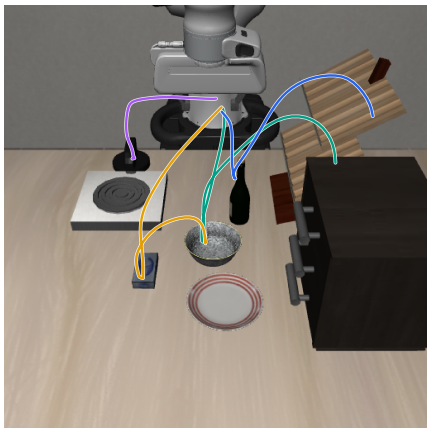
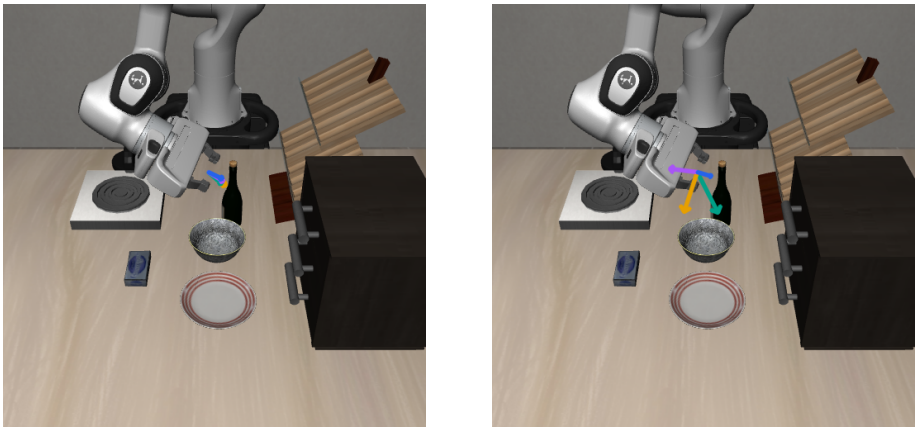


Figure 20: Official demonstration trajectories for the four LIBERO-Goal tasks used in our action visualization: *put the wine bottle on top of the cabinet*, *put the bowl on top of the cabinet*, *put the cream cheese in the bowl*, and *turn on the stove*. The trajectory colors are consistent with those used in the action visualizations and provide visual references for the corresponding task-consistent motion directions.

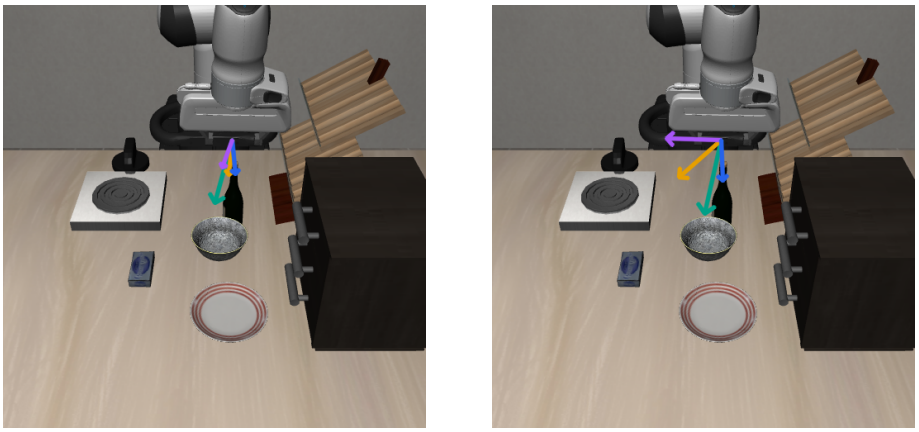


(a) Base checkpoint (b) Two-round CAS-ST

Figure 21: Instruction-conditioned actions for the same robot state before and after two rounds of CAS-guided self-training. Colors correspond to *wine bottle*, *bowl*, *cream cheese*, and *stove* tasks. Under the base checkpoint, the four actions nearly overlap and point toward the wine-bottle region. After self-training, the actions become clearly separated and point broadly toward their corresponding task-relevant regions.

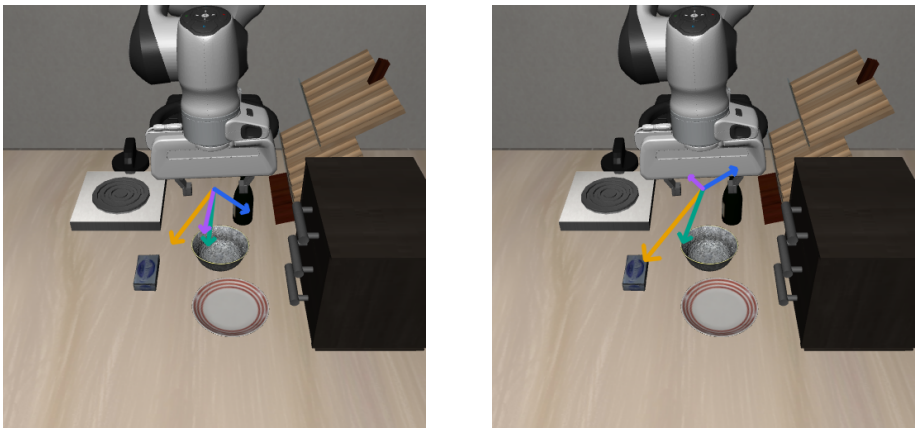
are not visible. In addition, task-consistent actions need not point directly toward the target object, since grasping may first require moving the gripper above the object before descending to grasp it. We therefore focus primarily on the coarse task-dependent motion directions. Despite these limitations, the contrasts between the two checkpoints are clear in the examples below: after two rounds of CAS-guided self-training, the predicted actions are generally better aligned with their corresponding task directions, while spurious overlap among actions for different task instructions is substantially reduced.

Case 1. As shown in Fig. 21, the base checkpoint predicts nearly overlapping actions for all four task instructions, with all arrows pointing broadly toward the wine-bottle region. After two rounds of CAS-ST, the actions become clearly separated: the purple action points toward the stove knob, the blue action toward the wine bottle, the orange action toward the cream-cheese region to the left



(a) Base checkpoint (b) Two-round CAS-ST

Figure 22: Another example of instruction-conditioned action predictions at the same robot state. Colors correspond to *wine bottle*, *bowl*, *cream cheese*, and *stove* tasks. The base checkpoint predicts very similar actions across all four instructions, whereas the two-round CAS-ST checkpoint produces clearly separated, task-dependent action directions.



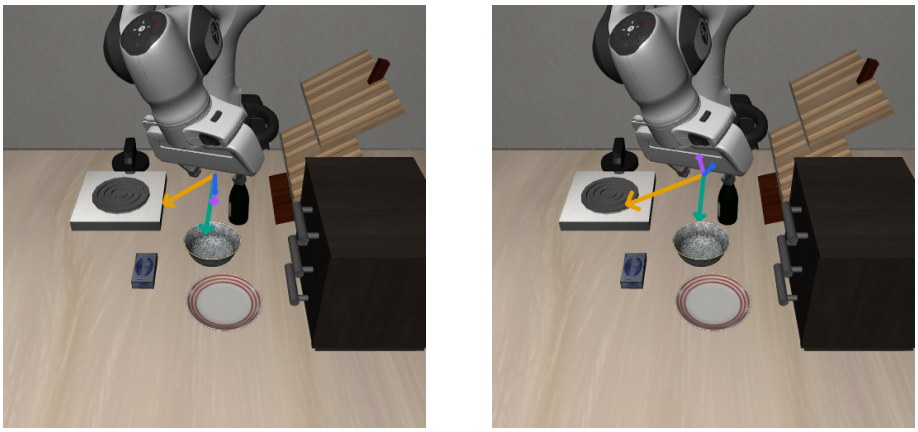
(a) Base checkpoint (b) Two-round CAS-ST

Figure 23: Instruction-conditioned actions in a state where the base checkpoint already exhibits partial task-dependent separation. Colors correspond to *wine bottle*, *bowl*, *cream cheese*, and *stove* tasks. While some base actions are task-consistent, others remain misaligned. After two rounds of CAS-ST, all four actions become more clearly separated and better aligned with their corresponding task directions.

of the bowl, and the green action toward the bowl. The resulting directions are substantially more task-dependent and better aligned with the corresponding tasks.

Case 2. In Fig. 22, the actions predicted for the four task instructions are very similar under the base checkpoint and all point approximately toward the wine bottle. After two rounds of CAS-ST, the actions become clearly separated: the purple, blue, orange, and green actions point broadly toward the stove knob, wine bottle, cream cheese, and bowl, respectively.

Case 3. In Fig. 23, the base checkpoint already exhibits some task-dependent action separation. The orange and green actions are broadly consistent with the cream-cheese and bowl tasks, respectively. However, the purple action, which should move toward the stove-knob region, instead points toward the bowl, while the blue action is poorly aligned with the wine-bottle direction. After two



(a) Base checkpoint (b) Two-round CAS-ST

Figure 24: Instruction-conditioned actions in a state where the base checkpoint confuses a subset of task instructions. Colors correspond to *wine bottle*, *bowl*, *cream cheese*, and *stove* tasks. In the base checkpoint, the wine-bottle and stove actions are biased toward the bowl region. After two rounds of CAS-ST, these actions shift toward directions more consistent with their respective tasks.

rounds of CAS-ST, all four actions become more clearly separated and better aligned with their corresponding task directions.

Case 4. As shown in Fig. 24, the orange and green actions predicted by the base checkpoint are broadly consistent with the cream-cheese and bowl tasks, respectively. In contrast, both the blue and purple actions are biased toward the bowl region rather than toward the wine bottle and stove. After two rounds of CAS-ST, the blue action redirects toward the wine bottle, while the purple action shifts toward the rear-left region, which is more consistent with the stove task.

Summary. Across these examples, CAS-guided self-training leads to clearer task-dependent action separation and generally better alignment with the corresponding task directions. Notably, even in challenging cases where the base checkpoint predicts highly similar actions across different instructions, these overlaps are substantially reduced after two rounds of CAS-ST.

F.3 Self-Training Performance under Different Training Budgets

We further evaluate Vanilla-ST and CAS-ST under different finetuning budgets on CALVIN-Custom-Switch using Fast-WAM. The self-training data construction and finetuning setup follow Appendix H.3; here, we vary only the number of finetuning steps.

As shown in Table 16, CAS-ST consistently outperforms Vanilla-ST across all evaluated training budgets. Notably, CAS-ST reaches 42.9% success with only 4k finetuning steps, already substantially outperforming Vanilla-ST trained for 20k steps (21.3%). This further demonstrates the higher training value and finetuning efficiency of CAS-guided rollouts.

F.4 Pair-Level Analysis of CAS-Guided Self-Training

We further examine the CALVIN-Custom-Switch self-training results at the individual source–target pair level. We follow the self-training setup described in Appendix H.3, using the $[9, 15]$ switching range for both data collection and evaluation. Here, we compare the pair-level difference in the number of retained self-training trajectories with the corresponding evaluation success rate.

CAS-ST can perform better with fewer training trajectories. Table 17 highlights two source–target pairs for which CAS-ST uses fewer self-training trajectories than Vanilla-ST, while achieving higher evaluation success. For Red $\rightarrow$ Button, CAS-ST uses four fewer trajectories while improving the success rate from 70.0% to 100.0%. Similarly, for Button $\rightarrow$ Red, CAS-ST uses seven fewer

Table 16: Self-training performance (%) of Fast-WAM on CALVIN-Custom-Switch under different finetuning budgets. CAS-ST with only 4k finetuning steps already substantially outperforms Vanilla-ST trained for 20k steps.

Finetuning Steps	Vanilla-ST	CAS-ST
4k	17.5	42.9
8k	18.8	48.8
12k	20.8	51.3
16k	19.6	53.8
20k	21.3	50.0

Table 17: Pair-level cases where CAS-ST uses fewer retained self-training trajectories than Vanilla-ST but achieves higher evaluation success. Δ Traj. denotes the difference in retained trajectories (CAS-ST $-$ Vanilla-ST), and Δ Succ. denotes the success-rate difference in percentage points.

	Δ Traj.	Success Rate (%)		
Source $\rightarrow$ Target	CAS-ST $-$ Vanilla-ST	Vanilla-ST	CAS-ST	Δ Succ. (pp)
Red $\rightarrow$ Button	-4	70.0	100.0	$+30.0$
Button $\rightarrow$ Red	-7	65.0	90.0	$+25.0$

trajectories while improving the success rate from 65.0% to 90.0%. Thus, for these pairs, the higher CAS-ST performance is observed despite using fewer retained self-training trajectories.

Pairs with more CAS-ST trajectories show no regression. Table 18 reports all seven source–target pairs for which CAS-ST contains more retained self-training trajectories than Vanilla-ST. CAS-ST achieves a higher success rate on six of these pairs and matches Vanilla-ST on the remaining pair; none shows lower performance. The additional number of retained trajectories ranges from three to eleven across individual pairs. For example, Red $\rightarrow$ Blue has six additional CAS-ST trajectories and improves from 0.0% to 45.0%, while Pink $\rightarrow$ Button also has six additional trajectories and improves from 10.0% to 95.0%. Across all seven pairs, CAS-ST contains 49 more retained trajectories, with a pooled success rate of 42.9% compared with 3.6% for Vanilla-ST.

Overall, this pair-level analysis provides a complementary view of the aggregate self-training results. For some task pairs, CAS-ST achieves higher success despite using fewer training trajectories; meanwhile, among all pairs where CAS-ST uses more trajectories, we observe either improved or unchanged pair-level performance.

F.5 Further Generalization to Later Switches

We further evaluate the extrapolation of CAS-guided self-training to later switching settings on LIBERO-Goal-Switch. Following the self-training setup in Appendix H.3, all self-training trajectories are collected at $t_s = 25$. In addition to the unseen later switch at $t_s = 30$ reported in Table 2, we further evaluate the same finetuned policies at $t_s = 35$, extending the evaluation further beyond the switching setting used for data collection.

As shown in Table 19, CAS-ST consistently outperforms Vanilla-ST at both unseen later switching steps for both $\pi_{0.5}$ and Fast-WAM. At $t_s = 35$, CAS-ST achieves 41.1% and 22.2% success for $\pi_{0.5}$ and Fast-WAM, respectively, exceeding Vanilla-ST by 6.7 and 6.6 percentage points. These additional results show that CAS-ST continues to outperform Vanilla-ST at a later unseen switching step.

F.6 Self-Training on CALVIN-Custom-Switch-Random

We further examine CAS-guided self-training on *CALVIN-Custom-Switch-Random*, introduced in Appendix C.3, where object placements are broadly randomized without fixed nominal locations. As discussed above, such randomization substantially changes the spatial organization of task tra-

Table 18: All source–target pairs where CAS-ST has more retained self-training trajectories than Vanilla-ST. CAS-ST improves six pairs and ties one, with no observed regression. Δ Traj. denotes the difference in retained trajectories (CAS-ST − Vanilla-ST), and Δ Succ. denotes the success-rate difference in percentage points.

	Δ Traj.	Success Rate (%)		
Source → Target	CAS-ST − Vanilla-ST	Vanilla-ST	CAS-ST	Δ Succ. (pp)
Red → Pink	+8	0.0	20.0	+20.0
Red → Blue	+6	0.0	45.0	+45.0
Pink → Red	+3	15.0	60.0	+45.0
Pink → Blue	+8	0.0	75.0	+75.0
Pink → Button	+6	10.0	95.0	+85.0
Blue → Pink	+7	0.0	0.0	0.0
Button → Pink	+11	0.0	5.0	+5.0
Overall	+49	3.6	42.9	+39.3

Table 19: Self-training performance (%) on LIBERO-Goal-Switch at unseen later switching steps. All self-training trajectories are collected at $t_s = 25$. The $t_s = 30$ results are also reported in Table 2, while $t_s = 35$ provides an additional evaluation farther beyond the self-training regime.

	$\pi_{0.5}$		Fast-WAM	
Method	$t_s = 30$	$t_s = 35$	$t_s = 30$	$t_s = 35$
Base	27.8	25.5	16.7	14.4
Vanilla-ST	32.2	34.4	21.1	15.6
CAS-ST	40.0	41.1	27.8	22.2

jectories and may consequently weaken the task-associated trajectory priors induced at particular states. We therefore investigate whether the advantage of CAS-guided self-training persists under this randomized object-placement setting.

As shown in Table 20, inference-time CAS substantially improves the success rate from 8.8% to 34.2%. More importantly, CAS-ST reaches 27.9% success without using CAS at inference time, while Vanilla-ST provides no improvement over the base policy (8.3% vs. 8.8%). These results show that CAS-guided rollouts remain effective self-training data even under fully randomized trajectory distributions, and provide substantially more useful supervision than vanilla successful rollouts in this setting.

G Beyond Instruction Switching: Trajectory-Biased Initial States

CAS is introduced in this work to address trajectory hijacking under mid-execution instruction switching. However, interfering trajectory priors may also arise when the robot is initialized from a configuration that is already strongly biased toward particular task trajectories. This raises a natural question: can CAS remain effective when the robot starts from a trajectory-biased initial state that is not induced by instruction switching? To study this complementary setting, we construct trajectory-biased initial states and evaluate whether CAS can improve target-task execution from these challenging initial configurations. As an auxiliary analysis beyond our primary instruction-switching setting, we focus on the VLA policies OpenVLA-OFT, π_0, and $\pi_{0.5}$.

G.1 Construction of Trajectory-Biased Initial States

We construct trajectory-biased initial states on LIBERO-Goal. For each task i, we aggregate the end-effector positions from its official demonstration trajectories into a task-specific trajectory point cloud D_i, following the notation in Sec. 3.2. Given a candidate robot configuration with end-effector

Table 20: Self-training results on CALVIN-Custom-Switch-Random in the $[15, 21]$ switching-step range. CAS-ST substantially improves π_0, while Vanilla-ST does not provide a meaningful gain.

Method	Success Rate (%)
Base	8.8
+ CAS	**34.2**
Vanilla-ST	8.3
CAS-ST	27.9

Table 21: Success rates (%) from trajectory-biased initial states on LIBERO-Goal. CAS consistently improves all evaluated VLA policies and outperforms PCD where applicable.

Policy	Method	Success Rate (%)
OFT	Base	25.5
	+ CAS	**39.5**
π_0	Base	30.2
	+ PCD	31.2
	+ CAS	**40.5**
$\pi_{0.5}$	Base	74.0
	+ PCD	76.2
	+ CAS	**78.6**

position $\mathbf{p} \in \mathbb{R}^3$, we measure its proximity to task i using the average distance to its K nearest demonstration points:

$$d_i(\mathbf{p}) = \frac{1}{K} \sum_{\mathbf{q} \in \mathrm{KNN}(\mathbf{p}, D_i)} \|\mathbf{p} - \mathbf{q}\|_2.$$

A smaller $d_i(\mathbf{p})$ indicates that the current end-effector position lies closer to the demonstrated trajectory distribution of task i. We sort the distances to the ten task trajectories as

$$d_{(1)}(\mathbf{p}) \le d_{(2)}(\mathbf{p}) \le \cdots \le d_{(10)}(\mathbf{p}).$$

To generate candidate states, we perturb the robot arm around its default configuration and allow the simulator to settle. We retain a candidate state only if

$$d_{(10)}(\mathbf{p}) < \tau, \qquad d_{(4)}(\mathbf{p}) - d_{(1)}(\mathbf{p}) > \delta_{\mathrm{gap}},$$

where we set $\tau = 0.12$ and $\delta_{\mathrm{gap}} = 0.03$. The first condition keeps the end-effector within a region well covered by the demonstrated task trajectories, ensuring that its proximity to different trajectory distributions remains meaningful. The second condition enforces a pronounced imbalance among these trajectory proximities, such that the current end-effector position is substantially closer to a small subset of task trajectories than to the others. We use $d_{(4)}$ rather than $d_{(2)}$ because several task trajectories can partially overlap in the workspace. Together, these criteria construct initial states in which a particular subset of trajectory priors is already dominant before any task execution begins, creating challenging cases for following a potentially conflicting target instruction.

Following this procedure, we construct 42 trajectory-biased initial states. For each initial state, we evaluate all ten LIBERO-Goal target instructions, yielding 420 evaluation episodes for each policy–method configuration. The same fixed set of 42 initial states and target-instruction assignments is shared across all evaluated methods.

G.2 Inference-Time Evaluation

Main results. We first evaluate whether CAS can directly improve target-task execution from trajectory-biased initial states at inference time. All methods are evaluated on the same 420 state–target episodes constructed from the 42 initial states described above. As shown in Table 21, CAS

Table 22: Success rates (%) from trajectory-biased initial states under different OpenVLA-OFT checkpoints and observation configurations.

Checkpoint	Observation	Base	+ CAS
Official LIBERO-Goal OFT	Third-person + wrist	25.5	39.5
Our reproduced OFT	Third-person + wrist	40.0	51.9
Our reproduced OFT	Third-person only	32.9	41.2

Table 23: Self-training performance (%) on the 42 trajectory-biased initial states on LIBERO-Goal. CAS-ST consistently outperforms Vanilla-ST across all three evaluated VLA policies.

Method	OFT	π_0	$\pi_{0.5}$
Base	25.5	30.2	74.0
+ Vanilla-ST	33.1	49.8	78.8
+ CAS-ST	**48.8**	**53.1**	**80.7**

consistently improves all three evaluated VLA policies. For OpenVLA-OFT, CAS increases the success rate from 25.5% to 39.5%; for π_0, from 30.2% to 40.5%; and for $\pi_{0.5}$, from 74.0% to 78.6%. For π_0 and $\pi_{0.5}$, we additionally compare against the training-free PCD baseline, which provides only modest improvements over the corresponding base policies, while CAS achieves consistently stronger performance. These results show that CAS remains effective when trajectory bias is already present in the initial robot configuration, even without any preceding source-task execution or instruction switch.

Robustness across checkpoints and observation settings. We further examine whether the benefit of CAS under trajectory-biased initial states is specific to a particular OpenVLA-OFT checkpoint or observation configuration. In addition to the official LIBERO-Goal OFT checkpoint used above, we finetune an OFT model ourselves on the same task suite and evaluate CAS on this reproduced checkpoint. We also consider a more constrained single-view setting in which the wrist camera is removed and the policy receives only the third-person observation.

As shown in Table 22, CAS consistently improves target-task success across all evaluated settings. On the official OFT checkpoint, CAS improves the success rate from 25.5% to 39.5%. On our reproduced OFT checkpoint, CAS improves performance from 40.0% to 51.9%. Even in the single-view setting without the wrist camera, CAS increases the success rate from 32.9% to 41.2%. These results suggest that the benefit of CAS under trajectory-biased initial states is not specific to a single checkpoint or camera configuration.

G.3 CAS-Guided Self-Training

Main results. We next examine whether CAS-guided trajectories can provide more effective supervision for improving the underlying policy under trajectory-biased initial states. We compare CAS-guided self-training (CAS-ST) with vanilla self-training (Vanilla-ST). Both methods finetune the same base policy using successful trajectories collected from the same distribution of trajectory-biased initial states, with collection states kept separate from the 42 states used for evaluation. Vanilla-ST and CAS-ST use exactly the same collection episodes, with matched initial robot states, object configurations, and target instructions. The only difference during data collection is whether CAS is enabled. Since CAS yields more successful rollouts, we uniformly downsample the CAS success pool to match the number of retained Vanilla-ST trajectories. Both resulting policies are evaluated without CAS at inference time, so the comparison reflects what is learned from the collected trajectories rather than any inference-time steering effect.

As shown in Table 23, CAS-ST consistently outperforms Vanilla-ST across all three evaluated VLA policies. For OpenVLA-OFT, CAS-ST improves the success rate from 25.5% to 48.8%, compared with 33.1% for Vanilla-ST. For π_0, CAS-ST reaches 53.1%, compared with 49.8% for Vanilla-ST, while for $\pi_{0.5}$ it achieves 80.7%, compared with 78.8%. Consistent with our findings under mid-execution instruction switching, these results further show that successful CAS-guided trajectories

Table 24: Self-training performance (%) on the 42 trajectory-biased initial states under different finetuning budgets. CAS-ST is substantially more finetuning-efficient: with only 10k finetuning steps, it already outperforms Vanilla-ST trained for 50k steps.

Method	Finetuning Steps	Success Rate (%)
Base	–	25.5
+ Vanilla-ST	10k	27.9
+ Vanilla-ST	30k	34.1
+ Vanilla-ST	50k	33.1
+ CAS-ST	10k	41.7
+ CAS-ST	30k	47.9
+ CAS-ST	50k	**48.8**

provide more effective self-training data than successful trajectories collected by the base policy alone, even when the trajectory bias is already present in the initial robot state.

Finetuning efficiency. We further examine whether the advantage of CAS-guided self-training persists across different finetuning budgets. All experiments use OpenVLA-OFT as the base policy and follow the same self-training protocol described above; Vanilla-ST and CAS-ST differ only in whether the successful training trajectories are collected with or without CAS guidance.

As shown in Table 24, CAS-ST consistently outperforms Vanilla-ST across all evaluated finetuning budgets. With only 10k finetuning steps, CAS-ST already reaches 41.7% success, substantially exceeding the 33.1% achieved by Vanilla-ST after 50k steps. Increasing the finetuning budget further improves CAS-ST to 47.9% at 30k steps and 48.8% at 50k steps, whereas Vanilla-ST provides substantially smaller gains and does not improve further from 30k to 50k steps. Consistent with our findings under mid-execution instruction switching, these results show that CAS-guided trajectories provide more effective supervision and enable substantially more efficient policy improvement. The same trend therefore holds when the challenging trajectory bias is already present in the robot's initial configuration rather than being induced by a preceding source-task execution.

Iterative self-training and standard-task performance. Since CAS-guided self-training can be applied iteratively, we further perform a second round of CAS-ST with OpenVLA-OFT. For this second round, we generate a new set of trajectory-biased initial states using the same construction procedure described above, while ensuring that these states are disjoint from the 42 states used for evaluation. We collect successful CAS-guided trajectories from these newly generated states and mix them with the original official LIBERO-Goal demonstration trajectories for further finetuning.

The second round provides an additional improvement on the trajectory-biased initial-state evaluation, increasing the success rate from 48.8% after one round of CAS-ST to 51.7% after two rounds. This result shows that additional CAS-guided recovery data can continue to improve the policy beyond a single round of self-training.

Importantly, the resulting policy also maintains strong performance under the standard task setting. On the standard LIBERO-Goal benchmark, it achieves **98.6%** success, exceeding the **97.9%** success rate reported for the official OpenVLA-OFT checkpoint.

H Experimental Details

H.1 Mid-Execution Instruction Switching Evaluation Details

Switch-boundary handling. We define the switching step t_s at the environment-action level. At the switch, we discard any unexecuted source-conditioned actions in the current action chunk, and the policy replans from the current observation under the target instruction. For LingBot-VA, we additionally clear its cached temporal state to eliminate explicit carryover from the source phase.

Table 25: Prediction horizons and actions executed per policy query in the Observation 1 online evaluations. For LingBot-VA, $12/16$ denotes 12 actions in the first chunk after a reset and 16 actions in subsequent streaming chunks.

Policy	Predicted horizon	LIBERO actions/query	CALVIN actions/query
OpenVLA-OFT	8	8	8
π_0	50	5	5
$\pi_{0.5}$	10	5	5
Fast-WAM	32	10	5
LingBot-VA	$12/16$	$12/16$	$12/16$

LIBERO-Goal online evaluation. We evaluate fully online instruction switching with $t_s \in \{0, 5, 10, 15, 20, 25\}$. At $t_s = 0$, the policy executes the target instruction directly from the common initial state. For $t_s > 0$, it first executes the source instruction for exactly t_s environment actions and then switches to the target instruction.

For each t_s, we evaluate all 90 ordered pairs of distinct source and target tasks and report the mean target-task success rate. The initial state, simulator stabilization, 300-action target-phase budget, and success criterion follow Appendix C.4.

CALVIN-Custom online evaluation. The CALVIN-Custom results in Fig. 2 use an online switching protocol, unlike the state-matched CALVIN-Custom-Switch protocol used for the main method evaluation. We use the same four tasks as in CALVIN-Custom-Switch.

For each nonzero switching range, we use 20 held-out initial configurations and all 12 ordered source–target task pairs, yielding 240 episodes per range. We consider four progressively later switching ranges: $t_s \in [3, 9]$, $[9, 15]$, $[15, 21]$, and $[21, 27]$. These ranges correspond to progress intervals $[0.1, 0.3]$, $[0.3, 0.5]$, $[0.5, 0.7]$, and $[0.7, 0.9]$ on a 30-step reference scale. The reference scale determines only the switching time; each policy executes its own source-conditioned prefix online.

After the switch, the policy has up to 180 environment actions to complete the target task. For $t_s = 0$, we directly execute each target task from 50 held-out initial configurations, yielding 200 episodes in total.

Inference settings. All results use the base policies without CAS or other post-switch interventions. On LIBERO-Goal, we follow each policy's standard evaluation setup. Table 25 summarizes the prediction horizons and the number of actions executed per policy query on both LIBERO-Goal and CALVIN-Custom. These settings are fixed across switching times.

H.2 Real-World Experiment Details

We provide additional implementation details for the real-robot experiments reported in Table 3. The experiments are conducted on an AIRBOT Play 6-DoF robot arm equipped with a parallel-jaw gripper. We evaluate the same mid-execution instruction-switching setting as in the main text, where the robot is first instructed to lift one colored block and the instruction is then switched to lifting the other block. We test both switching directions, blue-to-red and red-to-blue, at switching steps $t_s = 20$ and $t_s = 30$, with 20 trials for each setting. For a fair comparison, the base policy and CAS use matched object placements.

The deployed policy is a π_0 flow-matching policy initialized from the π_0 base checkpoint and fine-tuned on our real-world block-lifting demonstrations. We train a single checkpoint jointly on two tasks, *pick and lift the red block* and *pick and lift the blue block*. For each task, we collect 50 successful demonstration trajectories using a leader-follower teleoperation setup. The policy is finetuned for 60k gradient steps with a batch size of 32, using two NVIDIA H100 GPUs. The policy takes two RGB observations as input, including a static third-person camera view and a wrist camera view. Since the real-robot policy follows the standard π_0 observation interface, unused camera inputs are masked out.

Unlike the simulation experiments, where policies output delta end-effector actions, the real-robot policy predicts delta joint actions for the six arm joints, together with a gripper command. We reconstruct absolute joint targets by adding the predicted joint deltas to the current joint configuration. CAS is applied only to the arm-action dimensions, while the gripper command is directly taken from the target-conditioned action. This follows the same principle as in simulation, where the gripper dimension is excluded from counterfactual steering. The gripper command is then discretized into open or close using a fixed threshold.

For CAS, hijacking tasks are selected using the same KNN-based trajectory-distance criterion as in simulation. We use the KNN distance threshold $\delta = 0.03$ for all real-world experiments. The steering strength is set according to the switching step: $\lambda = 0.3$ for $t_s = 20$ and $\lambda = 0.8$ for $t_s = 30$. A larger steering strength is used for later switches because the robot has progressed further along the source-task trajectory, making trajectory hijacking stronger and requiring more aggressive correction. The final CAS-corrected joint action is sent to the robot controller for execution.

H.3 Self-Training Details

Controlled comparison. Our main one-round self-training comparison disentangles two benefits of CAS: a higher yield of successful switching rollouts and, more importantly, greater training value of the resulting trajectories. *Base* denotes the original task-finetuned checkpoint. Vanilla-ST finetunes Base on successful switching rollouts collected without CAS, whereas CAS-ST finetunes the same Base checkpoint on successful CAS-guided rollouts. Both finetuned policies are evaluated without CAS, so the comparison reflects what is learned from the collected trajectories rather than inference-time steering.

For the main one-round self-training comparison, we roll out the policies in settings that follow the same task-pair and switching protocol as LIBERO-Goal-Switch and CALVIN-Custom-Switch, while keeping the collection episodes disjoint from those used for evaluation. We retain only rollouts that successfully complete the switched target task.

Under the same collection settings, CAS consistently yields a larger pool of successful trajectories than the Base policy. To compare the training value of these trajectories under the same data budget, we retain all successful Vanilla rollouts and uniformly sample an equal-sized subset from the CAS success pool using seed 0. Vanilla-ST and CAS-ST are therefore trained on the same number of self-training trajectories with matched finetuning settings.

Self-training data collection. For LIBERO-Goal-Switch, we collect self-training trajectories at $t_s = 25$ using 450 distinct official LIBERO-Goal initial states. We cover all 90 ordered source–target task pairs defined in Appendix C.4, with five initial states assigned to each pair. Vanilla and CAS use identical episode–state pairings. For $\pi_{0.5}$, we use environment seed 7, while Fast-WAM uses inference seeds 42 and 43.

For CALVIN-Custom-Switch, we generate collection states following the switching-state construction in Appendix C.2, using seed 42. We collect trajectories in the $[9, 15]$ switching range, with 40 states for each of the 12 ordered source–target task pairs, yielding 480 rollout attempts per policy. Vanilla and CAS use the same saved intermediate states.

For both benchmarks, only the post-switch portion of each retained successful trajectory is used for self-training, and all retained samples are conditioned on the target instruction I_{tgt}. Table 26 summarizes the collection settings and resulting training-data composition.

CAS rollout settings. For CAS-guided data collection in the main one-round experiments, we use $K = 30$ for the trajectory distance $d_i(p_t)$ and set the hijacking-task threshold to $\delta = 0.1$. The setting-specific steering strengths λ are reported in Table 26; all other CAS inference settings follow Sec. 3.

Finetuning settings. For the main one-round comparison, Vanilla-ST and CAS-ST are initialized from the same Base checkpoint and use identical finetuning settings within each policy–benchmark comparison. Table 27 summarizes the training configurations.

Table 26: Self-training data collection and dataset composition for the main one-round results. Vanilla-ST and CAS-ST use matched numbers of retained self-training trajectories.

	LIBERO-Goal-Switch		CALVIN-Custom-Switch	
Setting	$\pi_{0.5}$	Fast-WAM	$\pi_{0.5}$	Fast-WAM
Collection switching setting	$t_s = 25$	$t_s = 25$	$[9, 15]$	$[9, 15]$
Collection rollout attempts (each)	450	2×450	480	480
Retained self-training trajectories (each)	148	221	104	80
Official demonstrations	500	500	–	–
Total training trajectories (each)	648	721	104	80
λ	0.003	0.0013	0.018	0.035
Reported checkpoint	10k	10k	20k	20k

Table 27: Finetuning settings for the main one-round Vanilla-ST and CAS-ST comparisons. Settings are matched within each policy–benchmark comparison; reported checkpoints are listed as LIBERO / CALVIN.

Setting	$\pi_{0.5}$	Fast-WAM
Initialization	Task-finetuned base checkpoint	Task-finetuned base checkpoint
Fine-tuning objective	Full-parameter policy fine-tuning	Joint video–action fine-tuning
Global batch size	32	32
Training seed	42	42
Optimizer	AdamW, gradient clip 1.0	AdamW
Peak learning rate	5×10^{-5}	1×10^{-4}
Warmup	10k steps	1k steps; cosine decay
EMA decay	0.999	–
Video/action loss weights	–	$1/1$
Reported checkpoint	10k / 20k	10k / 20k

Iterative CAS self-training. We further study whether CAS-guided self-training can be applied iteratively to progressively improve recovery from later and more challenging switching states. Round 1 corresponds to the $\pi_{0.5}$ CAS-ST model on LIBERO-Goal-Switch described above: starting from the task-finetuned $\pi_{0.5}$ checkpoint, we collect CAS-guided trajectories at $t_s = 25$ and finetune the policy to obtain the reported 10k checkpoint.

For Round 2, we initialize both data collection and subsequent finetuning from the Round-1 10k checkpoint and move the collection switch to the later setting $t_s = 40$. We perform 450 CAS-guided rollout attempts with steering strength $\lambda = 0.002$. Among these rollouts, 154 successfully complete the switched target task, and all 154 successful trajectories are retained. As before, only the post-switch portion of each retained successful trajectory is used for self-training and is conditioned on the target instruction I_{tgt}.

The Round-2 finetuning dataset consists of the same 500 official LIBERO-Goal demonstrations together with the 154 newly collected Round-2 CAS trajectories, for a total of 654 training trajectories. Importantly, we do not replay the 148 retained Round-1 self-training trajectories in the Round-2 finetuning dataset. Instead, Round 1 influences the second round through two complementary paths. First, the Round-1 10k checkpoint initializes the Round-2 policy. Second, the post-switch portions of the 148 retained Round-1 CAS trajectories are added to the task-trajectory library used for KNN-based hijacking detection during Round-2 CAS data collection. Thus, Round 2 uses an updated trajectory library containing recovery experience from the preceding round, while its policy finetuning data contain only the official demonstrations and the newly collected Round-2 CAS trajectories.

We finetune the Round-1 checkpoint for an additional 10k steps using the same $\pi_{0.5}$ optimization configuration as in Round 1. The resulting Round-2 checkpoint is evaluated without CAS, so improvements after the second round reflect recovery behavior incorporated into the policy rather than inference-time steering.

Evaluation. For the main one-round comparison, we evaluate both the data-collection setting and an unseen later-switch setting: $t_s = 25$ and $t_s = 30$ for LIBERO-Goal-Switch, and $[9, 15]$ and $[15, 21]$ for CALVIN-Custom-Switch.

With matched numbers of self-training trajectories, CAS-ST consistently outperforms Vanilla-ST across both benchmarks and both policies, at both the data-collection settings and the unseen later switches. Because CAS is disabled during evaluation, these gains reflect recovery behaviors learned from the CAS-guided trajectories rather than inference-time steering. The consistent improvements at unseen later switches further demonstrate that the learned de-hijacking behavior generalizes beyond the switching regime used for self-training.

For iterative CAS-ST, we evaluate the Round-1 and Round-2 checkpoints at $t_s \in \{40, 50, 60\}$ on LIBERO-Goal-Switch. These switching steps are all later than the Round-1 collection setting at $t_s = 25$. For Round 2, $t_s = 40$ matches its data-collection setting, while $t_s = 50$ and $t_s = 60$ are unseen later switching conditions. Both checkpoints are evaluated without CAS.

H.4 PCD Implementation Details

PCD comparison method. Policy Contrastive Decoding (PCD) (Wu et al., 2026) is a training-free inference-time method that contrasts the action distribution predicted from the original observation with that predicted from an object-masked observation. The object-masked observation removes task-relevant objects from the image, so the resulting contrast is intended to amplify actions that depend on the task object rather than on spurious visual cues. We apply PCD to the flow-matching-based policies π_0 and $\pi_{0.5}$ using KDE-based probabilistic modeling over sampled actions. We keep the original PCD decoding procedure unchanged and set the contrast strength to $\alpha = 0.2$, following the setting used for π_0 in the original paper.

Adapting PCD to our benchmarks. The main adaptation needed in our setting is the construction of object-masked observations. PCD requires identifying the task-relevant objects to be removed, but our benchmarks provide only RGB observations and natural-language instructions at test time. We therefore implement an instruction-grounded open-vocabulary detection pipeline. Given an instruction, we first extract object names using a rule-based parser tailored to the instruction patterns in LIBERO-Goal. Single-object commands such as *open X*, *push X*, or *pick up X* return the manipulated object; placement commands such as *put X on Y* or *stack X on Y* return both the manipulated object and the receptacle; and compound instructions joined by "and" are split into sub-commands, from which all referenced objects are collected.

The extracted object phrases are then normalized before being passed to the detector. We remove articles and location modifiers after the object noun; for example, *the bowl between the plate and the ramekin* is rewritten as *bowl*. Since bare LIBERO object names are often difficult for open-vocabulary detectors to ground reliably, we further apply a small hand-built alias map that rewrites them into more visually grounded queries, such as *bowl* to "black bowl", *plate* to "white plate", *block* to "wooden cube", and drawer-specific names to "drawer". The resulting queries are sent to Grounding DINO for object detection, followed by segmentation and inpainting to construct the contrast observation. We also preserve the robot arm during masking, so that only task objects are removed. If no valid object is parsed or detected at a step, we use the original observation as the contrast observation, in which case PCD reduces to the base policy for that step.