DEFLECT: TEMPORAL COUNTERFACTUAL PREFERENCE LEARNING FOR DELAY-ROBUST ASYNCHRONOUS VLAS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) policies increasingly rely on asynchronous inference to hide model latency behind ongoing robot motion. During inference, the robot continues executing its current action chunk while the next chunk is generated from the observation available at inference start. By the time the new chunk is executed, the robot or scene may have changed, so an action that fits the prediction-time state can be misaligned with the execution-time state. Training-time methods modify the policy input or supervised target based on the expected execution delay, but do not directly supervise how actions should change as the scene evolves between prediction and execution. Runtime repair methods improve chunk continuity or modify actions during deployment, but can still leave actions aligned with the prediction-time state rather than the execution-time state. Our key insight is that execution-time observations are unavailable when the online prediction is made but are available in recorded trajectories during offline training, making the prediction–execution mismatch observable during training. We propose DEFLECT, an offline post-training framework that turns this temporal contrast into preference supervision for a policy acting from stale inputs. Across Kinetix, LIBERO, and three real-robot tasks, DEFLECT improves delay robustness across the evaluated VLA policies and settings. At the longest supported delay, it improves success by 4.2 percentage points over VLASH on Kinetix and by 22.7 points over naive asynchronous execution with GR00T N1.7 on LIBERO.

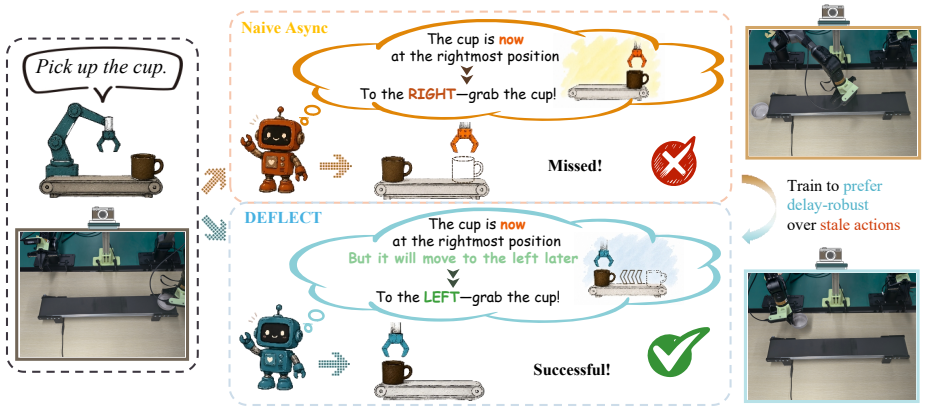


Figure 1: **DEFLECT lets an asynchronous VLA act more robustly from a stale observation.** The task is to grasp a cup on a moving conveyor; between observation and execution the cup travels from right to left. *Top* (Naive Async): the policy reaches where the cup was seen, so the grasp misses. *Bottom* (DEFLECT): given the same stale observation, the policy anticipates the upcoming motion and reaches the new position.

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.

1 INTRODUCTION

Vision-Language-Action (VLA) policies have shown strong performance across a wide range of robot manipulation and locomotion tasks (Brohan et al., 2023; Kim et al., 2024; Black et al., 2024; Intelligence et al., 2025). Many of these policies are deployed synchronously and predict actions as *action chunks* (Zhao et al., 2023): at each decision step the model outputs a short sequence of actions, the robot executes them in order, and the policy only runs again to produce the next chunk after the current one is finished. This execution scheme makes robot motion choppy: after each chunk, the robot must wait for the next inference to complete, during which it cannot respond to scene changes. To remove this pause, *asynchronous inference* has become a common deployment strategy for real-time VLA execution (Tang et al., 2025; Jiang et al., 2025; Black et al., 2026), in which the robot keeps executing the current chunk while the model computes the next one in parallel. Because each action chunk typically takes longer to execute than a single inference pass, the next chunk can be computed before the current one finishes, effectively hiding inference latency behind ongoing robot motion. This overlap also allows the policy to issue new chunks more frequently than stop-and-go execution, improving responsiveness to scene changes.

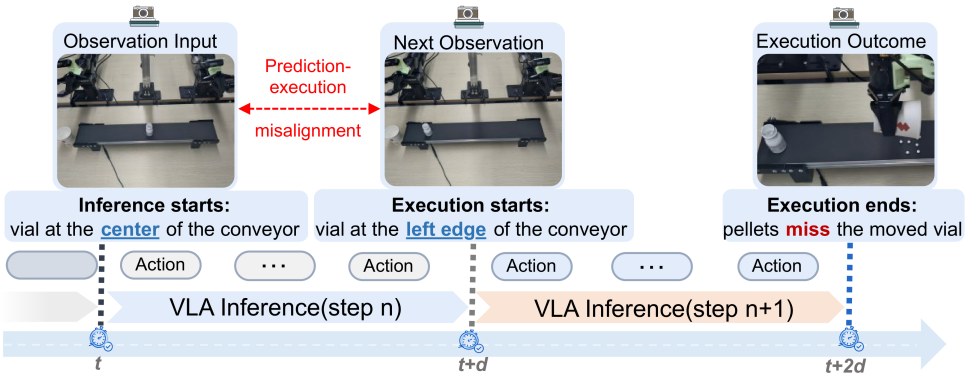


Figure 2: **Async-VLA inference under deployment-relevant latency: a real-robot example.** A robot dispenses pellets into a target vial on a moving conveyor. Top: robot frames at t, $t + d$, and $t + 2d$. Bottom: consecutive asynchronous VLA inference cycles. The policy starts inference at time t when the vial is centered, but the resulting actions execute after delay d, when the vial has moved. By execution end, the stale action chunk releases pellets where the vial was, causing a miss.

However, asynchronous inference also creates a prediction–execution mismatch: the next chunk is computed from an earlier observation but executed after the robot and scene have changed. The resulting action chunk may therefore no longer fit the state the robot actually faces when execution begins. As latency grows, this mismatch accumulates over continuous action execution: each step may look reasonable on its own, but the robot drifts from the correct trajectory and eventually fails the task. Figure 2 shows the severity of this problem: at prediction time, the vial lies at the center of the conveyor, and the predicted action chunk targets pellet release at that position. By the time the chunk is executed, the vial has moved to the other end of the conveyor, causing the pellets to miss the target. Existing asynchronous strategies (Tang et al., 2025; Black et al., 2026; Liu et al., 2025; Sendai et al., 2025) can soften this mismatch, but they do not address the central issue: the action chunk should match the execution-time world, not the prediction-time view.

We propose DEFLECT, an offline post-training framework for delay-robust asynchronous VLAs. Our key idea is to turn the information advantage of future observations into temporal preference supervision for a policy that must act from stale inputs. A frozen reference VLA generates paired chunks from future and stale observations. The policy learns to prefer the future-generated chunk under its unchanged deployment input. Future observations are used only for pair construction. The controller and inference budget remain unchanged. Across two simulated benchmarks and three real-robot tasks, DEFLECT improves asynchronous VLA control across the whole delay range its base policy supports, and the margin is largest where latency bites hardest: at the longest such

delay it gains $+4.2$ percentage points over VLASH on Kinetix and up to $+22.7$ points over naive asynchronous execution on a VLA backbone.

2 Background and Motivation

2.1 Asynchronous VLA Deployment

We denote the inference delay in control steps by d, so a prediction initiated at timestep t becomes available at $t+d$. At inference start, the policy receives the visual observation o_t, the proprioceptive state s_t, and the language instruction l_t. For a policy b obtained through supervised fine-tuning (SFT), let $\mathcal{C}_b$ denote how it constructs its input during asynchronous deployment. We use standard SFT for training on the current observation-state input and the expert action chunk beginning at the same timestep. Delay-aware SFT instead adjusts the supervised input or target to account for the expected execution latency.

Let $\mathcal{I}_t$ collect the observation, robot state, instruction, and any controller-side signals used by b. The deployment input $c_{t,b}^{\text{dep}}$ for policy b at control step t is given by

$$c_{t,b}^{\text{dep}} = \mathcal{C}_b(\mathcal{I}_t). \tag{1}$$

For example, a standard SFT policy deployed with naive asynchronous execution (Naive Async) uses $c_{t,b}^{\text{dep}} = (o_t, s_t, l_t)$, while VLASH is a representative delay-aware SFT method and uses $c_{t,b}^{\text{dep}} = (o_t, \hat{s}_{t+d}, l_t)$, where $\hat{s}_{t+d}$ is obtained by rolling forward the actions already committed for execution (Tang et al., 2025). DEFLECT post-trains policy b with the same deployment input and supervised target. We omit the index b when it is clear from context.

2.2 Limitations of Existing Async VLA Strategies

Existing asynchronous execution strategies mitigate parts of the latency problem, but leave unresolved a more fundamental misalignment: the policy aligns actions with the world observed at prediction time, while the robot needs actions that remain valid at execution time. As shown in Figure 3, an ideal zero-latency oracle would predict the next chunk $a'_{0:3}$ from the execution-time context (o_3, s_3), but this oracle is unavailable in real deployments: inference for $a'_{0:3}$ starts at s_1 while the robot is still executing the current action chunk a, and finishes only at s_3. Naive Async therefore predicts from the stale input (o_1, s_1), so its chunk follows the scene at inference start. Delay-aware SFT methods account for the expected execution latency during supervised training. VLASH (Tang et al., 2025) is one representative example, using the rolled-forward deployment context (o_1, s_3) shown in Figure 3. However, both standard SFT and delay-aware SFT learn to imitate a single supervised action target for each input, rather than explicitly ranking execution-time alternatives. Runtime repair methods such as RTC (Black et al., 2026), BID (Liu et al., 2025), and A2C2 (Sendai et al., 2025) improve chunk continuity. However, they leave a semantic-timestamp error: an action may be plausible for the prediction-time state yet misaligned with the execution-time state.

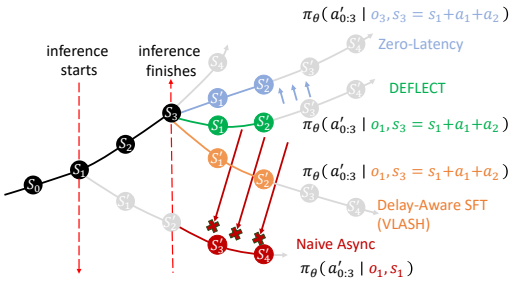


Figure 3: **Asynchronous deployment contexts under latency.** Naive Async uses (o_1, s_1), while VLASH represents delay-aware SFT with (o_1, s_3). The zero-latency oracle uses the unavailable (o_3, s_3). DEFLECT retains its base SFT policy's deployment input (o_1, s_3). It learns to favor future-generated over stale-generated chunks, shifting its actions toward the zero-latency reference.

2.3 Our Insight

Our key insight is that future observations are unavailable when an action must be predicted online, but are available in recorded trajectories during offline training. They therefore need not be part

of the deployment input to influence the policy's behavior. Instead, later observations can provide information about which actions are better aligned with the execution-time state and serve as a basis for learning action preferences, as shown in Figure 3.

This temporal contrast comes from the observation timestamp. A zero-latency policy would condition on the execution-time observation (o_{t+d}, s_{t+d}), whereas the asynchronous policy must act from the stale prediction-time view. DEFLECT uses this contrast to define a temporal counterfactual preference. The chunk generated from the future observation is labeled as preferred over the chunk generated from the stale view. It then learns this preference under the same deployment input used at test time, without changing the runtime interface.

3 DEFLECT

Figure 4 provides an overview of DEFLECT. We first construct temporal counterfactual supervision from offline expert trajectories, as shown in Figure 4a. The input at t represents the stale prediction-time scene, while the input at $t+\delta$ represents a future execution-time scene, where δ is sampled from the deployment-relevant delay range. A frozen reference VLA is queried on these two timestamped inputs with shared sampling randomness, so the resulting contrast mainly reflects the stale-versus-future input difference rather than sampling noise.

Figure 4b illustrates how DEFLECT uses this temporal contrast to train the policy under the deployment-time input c_t^{dep} of its base policy, which is also used at test time. The stale and future observations are used only to define the offline preference signal, not as separate optimization inputs. For the VLA backbones, the VLM backbone is frozen, and only the action expert is updated with the objective

$$\lambda_{\mathrm{SFT}}\mathcal{L}_{\mathrm{FM}} + \lambda_{\mathrm{DPO}}\mathcal{L}_{\text{FM-DPO}},$$

where the Supervised Fine-Tuning (SFT) flow-matching term anchors the policy to the expert action manifold, and the Flow-Matching Direct Preference Optimization (FM-DPO) term favors the candidate generated from the future observation over the one generated from the stale observation.

After training, DEFLECT leaves the asynchronous runtime interface unchanged. The resulting policy uses the same deployment input and inference budget as the SFT policy it refines, without access to future observations.

3.1 Constructing Temporal Counterfactual Preference Pairs

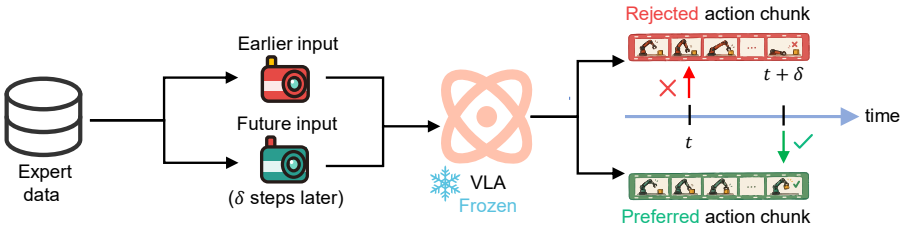
DEFLECT constructs temporal preference pairs independently of the deployment input used by the SFT policy it refines. Let b denote this policy. We use d for the deployment-time inference delay and δ for the temporal contrast horizon used to construct a training pair. For each demonstration timestep t, we sample a temporal contrast horizon δ from the deployment-relevant delay range and query the corresponding frozen reference policy $\pi_{\mathrm{ref},b}$ using the same language instruction l_t and sampling randomness ξ:

$$A_t^+ = \pi_{\mathrm{ref},b}\big((s_{t+\delta}, o_{t+\delta}), l_t; \xi\big), \qquad A_t^- = \pi_{\mathrm{ref},b}\big((s_t, o_t), l_t; \xi\big). \tag{2}$$

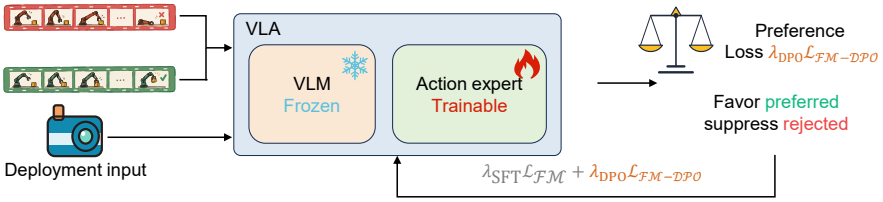
A_t^+ is generated from the future observation, whereas A_t^- is generated from the stale observation. We label A_t^+ as preferred over A_t^-. This label is based on the observation timestamps, not on the outcomes of executing the two chunks. It guides training but does not mean that A_t^+ performs better in every pair. The future observation is used only to construct the offline preference and is never provided as an input to the trainable policy. Because both candidates use the same reference policy and sampling randomness, their difference mainly comes from the observation timestamp. The contrast horizon δ is sampled from the deployment-relevant delay range and independently of any base-policy-specific variable used to construct the deployment input. The pair therefore provides relative temporal supervision rather than a direct regression target for one particular deployment context.

3.2 Preference Optimization under the Deployment-Time Input

DEFLECT retains the deployment input and supervised target of the SFT policy it refines. Let $\mathcal{D}$ denote the offline demonstration dataset, and let $\mathcal{S}_b$ denote the sampler used to form the original



(a) Temporal counterfactual data preparation. Here t denotes the prediction-time step, and δ is the sampled temporal contrast horizon, so the future input corresponds to $t+\delta$.



(b) Preference optimization under the deployment input. The objective $\lambda_{\text{SFT}}\mathcal{L}_{\text{FM}} + \lambda_{\text{DPO}}\mathcal{L}_{\text{FM-DPO}}$ balances the SFT flow-matching anchor and the DPO preference loss.

Figure 4: **DEFLECT Overview.** (a) A frozen reference VLA contrasts earlier and future inputs from offline trajectories to construct temporal counterfactual supervision. (b) DEFLECT freezes the VLM and trains the action expert to favor chunks generated from future observations over those generated from stale observations, while preserving the expert action manifold.

SFT examples for policy b. For each timestep t, it returns an SFT sample

$$\left(c_{t,b}^{\text{dep}}, A_{t,b}^{\text{exp}}\right) \sim \mathcal{S}_b(\mathcal{D}, t), \tag{3}$$

where $c_{t,b}^{\text{dep}}$ is the input used by policy b during deployment and $A_{t,b}^{\text{exp}}$ is its corresponding expert SFT target. DEFLECT reuses this input–target pair during post-training and adds the temporal preference objective.

Although A_t^+ and A_t^- are generated from different timestamps, DEFLECT evaluates both candidates under the same $c_{t,b}^{\text{dep}}$. This shared context turns the temporal pair into the decision faced by the deployed policy.

Following DPO and its adaptations to diffusion and flow-matching models (Rafailov et al., 2023; Wallace et al., 2024; McAllister et al., 2026), we estimate the per-example flow-matching objective using one sampled (τ, ϵ) and use the resulting loss as a surrogate score for negative log likelihood. Let θ denote the trainable policy parameters and $v_\theta(\cdot)$ its predicted flow velocity field. For an action chunk A, conditioning input c, flow time $\tau \in [0,1]$, and Gaussian noise ϵ, define

$$A_\tau = \tau A + (1-\tau)\epsilon, \qquad \mathcal{L}_{\text{FM}}(\theta; A, c, \tau, \epsilon) = \|v_\theta(A_\tau, \tau, c) - (A-\epsilon)\|_2^2. \tag{4}$$

The same sampled (τ, ϵ) is reused across both candidates, the trainable and reference policies, and the SFT anchor. Define

$$\begin{aligned} \Delta_\theta^\pm &= \mathcal{L}_{\text{FM}}(\theta; A_t^\pm, c_{t,b}^{\text{dep}}, \tau, \epsilon) - \mathcal{L}_{\text{FM}}(\pi_{\text{ref},b}; A_t^\pm, c_{t,b}^{\text{dep}}, \tau, \epsilon), \\ \mathcal{L}_{\text{FM-DPO}}(\theta) &= -\log\sigma\left(\beta\left[\Delta_\theta^- - \Delta_\theta^+\right]\right), \end{aligned} \tag{5}$$

where σ is the sigmoid function and β controls the preference strength. This objective provides a reference-calibrated flow-matching surrogate for the DPO log-ratio margin. Under the deployment input of policy b, it favors the candidate generated from the future observation over the candidate generated from the stale observation.

The final objective combines this relative preference with the original supervised target:

$$\mathcal{L}_{\text{DEFLECT}}(\theta) = \lambda_{\text{SFT}}\mathcal{L}_{\text{FM}}(\theta; A_{t,b}^{\text{exp}}, c_{t,b}^{\text{dep}}, \tau, \epsilon) + \lambda_{\text{DPO}}\mathcal{L}_{\text{FM-DPO}}(\theta). \tag{6}$$

Here $\lambda_{\mathrm{SFT}} \geq 0$ weights the supervised term and $\lambda_{\mathrm{DPO}} \geq 0$ the preference term. The SFT term preserves the supervised behavior of policy b, while the preference term introduces a future-aware temporal correction. When $\lambda_{\mathrm{DPO}} = 0$, Eq. (6) reduces to continued SFT of the same policy. For policies deployed with Naive Async, one checkpoint serves all evaluated delay settings using the same input form (o_t, s_t, l_t). This setup yields a correction shared across the training delay range. At deployment, DEFLECT retains the context constructor, asynchronous controller, and inference budget of the SFT policy it refines.

4 EXPERIMENTS

We design our experiments and analyses to answer the following four questions:

1. Does DEFLECT improve task success over naive asynchronous execution and strong baselines in simulation and the real world, without adding inference-time computation?
2. Does DEFLECT improve over Continued SFT and Future SFT, and which design choices are important for its performance?
3. Does DEFLECT improve both standard and delay-aware SFT policies across different VLA backbones and simulated and real-robot settings?
4. How does DEFLECT's benefit change with inference delay, and how sensitive is it to the contrast-horizon range used to construct preference pairs?

4.1 SIMULATION EVALUATION

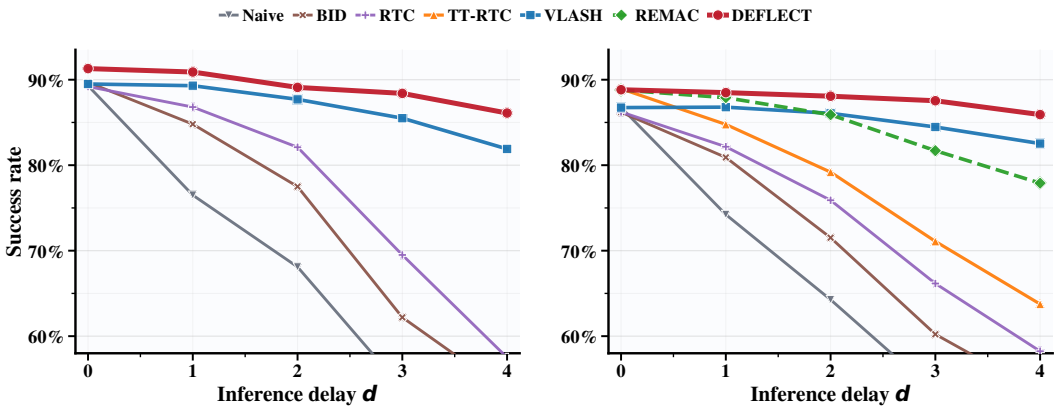
We evaluate DEFLECT on two simulated robot-control benchmarks: Kinetix (Matthews et al., 2025) and LIBERO (Liu et al., 2023). Kinetix focuses on fast reactions in highly dynamic environments, while LIBERO evaluates image-based manipulation across a diverse set of tasks.

4.1.1 KINETIX

Experimental Setup. Kinetix (Matthews et al., 2025) contains 12 dynamic control tasks involving contact-rich manipulation and locomotion, including catching, launching, and balancing. Since the robot and the environment can change substantially during a single inference call, Kinetix provides a direct test of robustness to prediction–execution delay. Following RTC (Black et al., 2026) and VLASH (Tang et al., 2025), we use an action-chunking flow policy with an action-chunk length of $H = 8$ and a four-layer MLP-Mixer architecture. All methods in the standard comparison use the same released VLASH checkpoint, action-chunk length, and asynchronous execution protocol. Following the VLASH evaluation convention, we use an execution horizon of $K = \max(1, d)$, meaning that K actions are executed from each chunk before switching to the next prediction. We evaluate simulated inference delays $d \in \{0, 1, 2, 3, 4\}$. We report the average success rate across all 12 tasks, with 1,024 rollouts per task for each delay and method. DEFLECT is post-trained for 24 epochs.

Baselines. We compare DEFLECT against the following baselines and controls:

- **Naive.** This baseline runs policy inference in parallel with action execution and switches to the newly predicted chunk as soon as it becomes available. It does not explicitly correct for the stale observation used to predict the new chunk.
- **RTC.** Real-Time Chunking (Black et al., 2026) keeps the actions that are already committed for execution and inpaints the remaining part of the new chunk. This repairs chunk transitions but requires additional computation during deployment.
- **BID.** Bidirectional Decoding (Liu et al., 2025) resamples action chunks at test time using forward-contrast and backward-coherence criteria. Like RTC, it modifies the runtime sampling procedure and adds inference-time computation.
- **VLASH.** VLASH (Tang et al., 2025) is a training-time asynchronous baseline that rolls the proprioceptive state forward to the expected execution time while retaining the visual observation captured at inference start. For this benchmark, its released checkpoint serves as the delay-aware SFT initialization for DEFLECT.



(a) Standard protocol ($K = \max(1, d)$) (b) REMAC protocol (K averaged)

Figure 5: **Kinetix success rate** under the two evaluation conventions. Averaging over all legal horizons $K \in [\max(1, d),\ 8 - d]$ in (b) lowers every method's absolute values relative to (a).

We additionally compare with TT-RTC (Black et al., 2025) and REMAC (Wang et al., 2026) in a separate protocol-matched experiment. REMAC reports results by averaging over all valid execution horizons, whereas the standard VLASH protocol evaluates the single horizon $K = \max(1, d)$. We therefore evaluate all methods under REMAC's convention for this comparison rather than mixing results from the two protocols.

Results. As shown in Figure 5(a), DEFLECT achieves the best performance under the VLASH protocol, improving the average success rate from 86.8% to 89.2% over VLASH. At $d = 4$, DEFLECT reaches 86.1%, outperforming VLASH by 4.2 percentage points. Figure 5(b) shows that the conclusion remains unchanged under the REMAC protocol: DEFLECT achieves the highest average success rate of 87.8% and exceeds REMAC by 8.0 points at $d = 4$. Ablations show that keeping the SFT loss and scoring both candidates with the same deployment input improve task success (Appendix F). Even when preference pairs use observations only one or two control steps apart, DEFLECT still outperforms VLASH by 3.2 percentage points at a four-step inference delay (Appendix E).

4.1.2 LIBERO

Experimental Setup. We evaluate DEFLECT on LIBERO (Liu et al., 2023), a widely used VLA benchmark containing four task suites: Spatial, Object, Goal, and LIBERO-10, with ten tasks in each suite. We evaluate three SFT-trained VLA policies: $\pi_{0.5}$, SmolVLA, and GR00T N1.7. The $\pi_{0.5}$ policy uses a delay-aware SFT initialization following VLASH, while SmolVLA and GR00T N1.7 retain their standard SFT initializations and use Naive Async at deployment. DEFLECT preserves the deployment input and supervised target of each policy. We follow the standard evaluation protocol of each backbone, using execution horizons of $K = 5$ for $\pi_{0.5}$ and SmolVLA, and $K = 8$ for GR00T N1.7. We evaluate delays $d \in \{1, 2, 3, 4\}$, using 500 episodes per suite and delay for all three backbones.

Results. As shown in Table 1(a), DEFLECT substantially improves the delay robustness of GR00T N1.7. Its gains grow with the delay: $+1.35$ percentage points at $d = 1$, $+6.15$ at $d = 2$, $+16.05$ at $d = 3$, and $+22.70$ at $d = 4$. As a result, the average success rate increases from 74.20% to 85.76%.

Future SFT performs best at $d = 1$, but its success decreases more sharply as delay grows. DEFLECT exceeds it by 2.85 percentage points on average and 7.35 points at $d = 4$, suggesting that preference learning improves delay robustness beyond directly learning future-generated targets.

Table 1(b) further shows positive gains across all three SFT policies. DEFLECT improves the delay-aware $\pi_{0.5}$ policy by $+0.74$ percentage points on average, standard-SFT SmolVLA by $+10.49$, and standard-SFT GR00T N1.7 by $+11.56$. Because the backbones and evaluation pro-

Table 1: **LIBERO success rates (%) across VLA policies.** Avg. denotes the mean over delays $d \in \{1, 2, 3, 4\}$, measured in control steps. In (b), the base policy is VLASH for $\pi_{0.5}$ and Naive Async for SmolVLA and GR00T N1.7. Gains are relative to the base policy and reported in percentage points (pp).

(a) GR00T N1.7: Delay sweep and SFT baselines

Method	$d = 1$	$d = 2$	$d = 3$	$d = 4$	Avg.
Naive Async	92.25	84.90	69.25	50.40	74.20
Continued SFT ($\lambda_{\text{DPO}} = 0$)	91.65	85.40	75.55	62.25	78.71
Future SFT	**95.20**	90.10	80.60	65.75	82.91
DEFLECT	93.60	**91.05**	**85.30**	**73.10**	**85.76**

(b) Gains across VLA backbones and SFT initializations

		Avg. success rate (%)		Gain (pp)	
Backbone	SFT initialization	Base policy	DEFLECT	Avg.	$d = 4$
$\pi_{0.5}$	Delay-aware (VLASH)	95.54	96.28	+0.74	+1.40
SmolVLA	Standard	49.77	60.26	+10.49	+14.00
GR00T N1.7	Standard	74.20	85.76	+11.56	+22.70

tocols differ, we evaluate the improvement over each policy's own SFT baseline rather than directly comparing gain magnitudes across backbones. The consistent improvements show that DEFLECT can refine both standard and delay-aware SFT policies across VLA backbones.

4.2 Real-World Evaluation

To evaluate DEFLECT in real-world settings, we deploy an in-house $\pi_{0.5}$ policy on a bimanual AgileX PiperX platform with two 6-DoF arms. The robot is equipped with three Intel RealSense D435i cameras, including one overhead camera and one wrist camera on each arm. Our system is built on LeRobot (Cadene et al., 2026). The comparison covers three stages of the same $\pi_{0.5}$ training pipeline: standard SFT with Naive Async, delay-aware SFT following VLASH, and DEFLECT post-training initialized from the delay-aware SFT policy. We evaluate task success in dynamic multi-stage manipulation and responsiveness to rapid environmental changes.

4.2.1 Task Success and Reactivity

Experimental Setup. We evaluate $\pi_{0.5}$ with an action-chunk length of $H = 50$ and 10 denoising steps. The robot operates at 30 Hz, and the asynchronous controller requests a new prediction when half of the current action queue remains. We compare $\pi_{0.5}$ with Naive Async execution, VLASH, and DEFLECT. All methods use the same demonstrations, VLA backbone, asynchronous controller, and inference budget.

We evaluate the following three tasks:

- **Conveyor-I.** The left arm places a cup on a moving conveyor, and the right arm retrieves it before it leaves the workspace.
- **Conveyor-II.** The left arm places a cup on the conveyor. The right arm then selects the red cube rather than the green cube, places it into the moving cup, and retrieves the cup.
- **Whack-a-Mole.** The robot strikes targets that appear briefly at random positions before they retract.

For each task, we conduct 30 trials per method. We report full-task success rates for Conveyor-I and Conveyor-II, where every sub-goal must be completed, and the average number of targets struck during each 30-second Whack-a-Mole trial. Detailed training configurations and per-sub-goal results are provided in the appendix.

Results. As shown in Figure 6, DEFLECT outperforms both $\pi_{0.5}$ with Naive Async and VLASH across all three tasks. On Conveyor-I, DEFLECT achieves a full-task success rate of 96.7%, com-

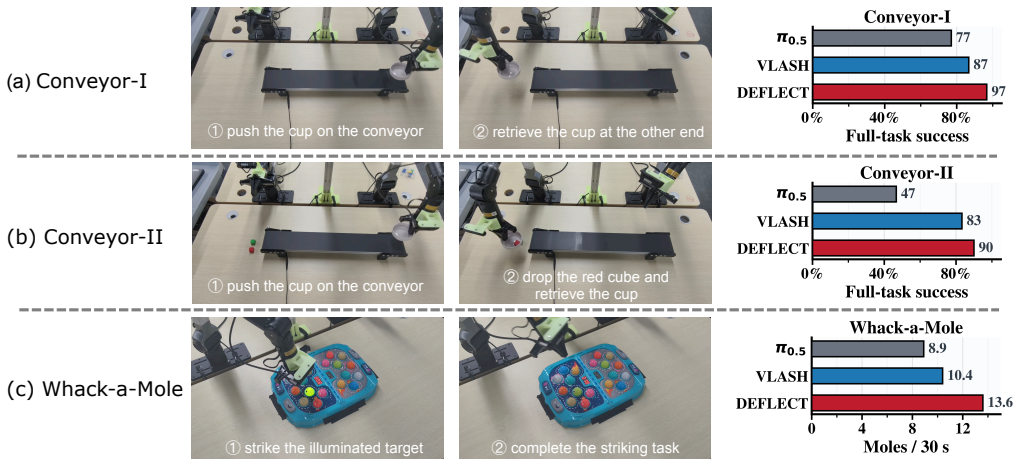


Figure 6: **Real-robot tasks and results.** (a) *Conveyor-I*, (b) *Conveyor-II*, and (c) *Whack-a-Mole*. The conveyor tasks report full-task success, while Whack-a-Mole reports the average number of targets struck per 30-second trial.

Table 2: **Model inference latency of $\pi_{0.5}$ across GPUs.** The action expert uses 10 denoising steps. Equivalent delay is the total inference latency rounded up to whole control steps at 30 Hz.

GPU	Inference latency (ms)			Equivalent delay (control steps)
	Vision encoder + VLM prefill	Action expert	**Total**	
RTX 5090	38.3	18.8	**57.1**	2
RTX 4090	56.9	21.9	**78.8**	3
RTX 5070	116.6	35.4	**152.0**	5

pared with 86.7% for VLASH and 76.7% for $\pi_{0.5}$. On Conveyor-II, DEFLECT reaches 90.0%, compared with 83.3% for VLASH and 46.7% for $\pi_{0.5}$. On Whack-a-Mole, DEFLECT strikes an average of 13.6 targets per 30-second trial, outperforming VLASH at 10.4 and $\pi_{0.5}$ at 8.9. These results show that DEFLECT improves both multi-stage manipulation and reactive control without additional inference-time computation.

4.2.2 Model Inference Latency

Table 2 reports the inference latency of $\pi_{0.5}$ across three GPUs. On the RTX 5090 used in our real-robot experiments, model inference takes 57.1 ms; it takes 78.8 ms on the RTX 4090 and 152.0 ms on the RTX 5070. Rounding these latencies up to whole control periods at 30 Hz gives two steps for the RTX 5090, three for the RTX 4090, and five for the RTX 5070. These hardware latencies are comparable in scale to the multi-step delays evaluated in simulation. FASTER (Lu et al., 2026) reduces the $\pi_{0.5}$ time-to-first-action to 62.1 ms on an RTX 4090, still nearly two control periods at 30 Hz. Such residual latency motivates delay-robust control alongside faster inference, which is the focus of DEFLECT.

5 Conclusion

We present DEFLECT, an offline post-training framework for delay-robust asynchronous Vision-Language-Action models with flow-matching action heads. By contrasting action chunks generated from future and stale observations under the same deployment input, DEFLECT learns to reduce the prediction–execution mismatch. Across Kinetix, LIBERO, and real-robot tasks, DEFLECT improves asynchronous control, with larger gains at longer delays and no additional inference-time computation. These results support temporal preference learning for more reliable VLA control in dynamic environments.

REPRODUCIBILITY STATEMENT

All components needed to reproduce our results are specified in the paper. The DEFLECT objective is given in closed form in Eqs. (2)–(6), and one full gradient step, including preference-pair construction, the shared (τ, ϵ) sampling protocol, and the joint SFT+DPO update, is written out as pseudocode in Algorithm 1 (Appendix C). Appendix B lists the reference checkpoints, trainable-parameter counts, optimizer, schedule, batch size, learning rate, and both loss weights for each benchmark (Table 3), together with the flow-matching loss-reduction convention that the DPO margin depends on. Evaluation protocols and episode counts are given in Appendix B. Both benchmarks (Kinetix (Matthews et al., 2025) and LIBERO (Liu et al., 2023)) are public, and we use their standard reward, termination, episode-length, and success conventions without modification. The real-robot experiments depend on our specific bimanual hardware setup, described in Appendix H; those results are reproducible only on comparable hardware.

AI USE STATEMENT

In this work, we used generative AI tools in three roles, and we describe all of them in full.

For writing, we used them for grammar correction, phrasing, and copy-editing of author-written text.

For code, we used an AI coding assistant throughout the experimental pipeline rather than for plotting alone. The preference-pair collection, post-training, and evaluation scripts, the cluster job-submission and monitoring tooling, and the analysis and figure-plotting code were written with AI assistance. The assistant was also used to launch these jobs, to parse the resulting run logs, and to compute the summary statistics and significance tests we report. In the course of that work it also proposed diagnostic experiments and ablations, several of which we ran and several of which we rejected.

We also used generative AI tools to search for and summarize related literature. Every work cited in this paper was then located and read by at least one author, and we did not rely on a model's summary for any claim we attribute to prior work. The research question and the DEFLECT objective are the authors' own. All AI-assisted text was reviewed and edited by the authors. All AI-assisted code was read by at least one author, and every number reported in this paper is the output of executing that code on the public benchmarks or on our own robot, checked against the raw run logs rather than transcribed from a model's summary. We take responsibility for the final content of this work, including all text, claims, and artifacts.

REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Kevin Black, Allen Z. Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking, 2025. URL https://arxiv.org/abs/2512.05964.

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Remi Cadene, Simon Alibert, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, et al. Lerobot: An open-source library for end-to-end robot learning. In *International Conference on Learning Representations*, volume 2026, pp. 122398–122417, 2026.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, et al. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Yuhua Jiang, Shuang Cheng, Yan Ding, Feifei Gao, and Biqing Qi. Asyncvla: Asynchronous flow matching for vision-language-action models. *arXiv preprint arXiv:2511.14148*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 1(2):4, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

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

Jiaming Liu, Hao Chen, Zhuoyang Liu, Pengju An, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, et al. Hybridvla: Collaborative diffusion and autoregression in a unified vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 80668–80694, 2026.

Yuejiang Liu, Jubayer Hamid, Annie Xie, Yoonho Lee, Max Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via guided test-time sampling. In *International Conference on Learning Representations*, volume 2025, pp. 4594–4627, 2025.

Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. Faster: Rethinking real-time flow vlas. *arXiv preprint arXiv:2603.19199*, 2026.

Wuyang Luan, Junhui Li, Weiguang Zhao, Wenjian Zhang, Tieru Wu, and Rui Ma. Snapflow: One-step action generation for flow-matching vlas via progressive self-distillation. *arXiv preprint arXiv:2604.05656*, 2026.

Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. *arXiv preprint arXiv:2410.23208*, 2025.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. In *International Conference on Learning Representations*, volume 2026, pp. 36352–36372, 2026.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224*, 2025.

Juyi Sheng, Ziyi Wang, Peiming Li, and Mengyuan Liu. Mp1: Meanflow tames policy learning in 1-step for robotic manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18532–18539, 2026.

Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8228–8238. IEEE, 2024.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. In *International Conference on Learning Representations*, volume 2026, pp. 120161–120179, 2026.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: toward fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3988–3995, 2025.

Ge Yan, Jiyue Zhu, Yuquan Deng, Shiqi Yang, Ri-Zhao Qiu, Xuxin Cheng, Marius Memmel, Ranjay Krishna, Ankit Goyal, Xiaolong Wang, et al. Maniflow: A general robot manipulation policy via consistency flow training. *arXiv preprint arXiv:2509.01819*, 2025.

Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems*, 37:56619–56643, 2024.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

A Related Work

Efficient VLA inference. Recent VLAs commonly predict action chunks using diffusion or flow matching (Zhao et al., 2023; Black et al., 2024; Intelligence et al., 2025). To improve real-time deployment, DeeR-VLA reduces backbone computation, SnapFlow reduces flow sampling steps, and FASTER prioritizes and streams near-term actions (Yue et al., 2024; Luan et al., 2026; Lu et al., 2026). These methods shorten the time before actions become available, but do not eliminate the prediction–execution mismatch. Nonzero latency remains, and later actions in a chunk are still conditioned on an earlier observation, so they may become stale before execution. DEFLECT instead learns actions that remain aligned with the scene after this residual delay.

Asynchronous action-chunk execution. Runtime methods such as Temporal Ensembling, BID, RTC, and A2C2 improve chunk transitions or correct actions during deployment (Zhao et al., 2023; Liu et al., 2025; Black et al., 2026; Sendai et al., 2025). Training-time methods including VLASH, TT-RTC, AsyncVLA, and REMAC adapt the policy to delayed execution through future-state conditioning, action-prefix conditioning, selective refinement, or masked chunk correction (Tang et al., 2025; Black et al., 2025; Jiang et al., 2025; Wang et al., 2026). DEFLECT differs by explicitly contrasting action candidates generated from future and stale observations under the same deployable input. This supervision is constructed offline and leaves the asynchronous runtime unchanged.

B Implementation Details

Preference-pair generation. Both the earlier input (s_t, o_t) and the later input $(s_{t+\delta}, o_{t+\delta})$ are taken from the same recorded expert trajectory. A frozen copy $\pi_{\mathrm{ref},b}$ of the SFT policy b generates A^+ from the later input and A^- from the earlier input using identical Gaussian noise and the same ODE schedule. These chunks are model predictions, not the expert actions recorded in the dataset. The expert actions are retained for the SFT anchor (Algorithm 1). We set the maximum temporal contrast horizon to $d_{\max}{=}4$ on both benchmarks and sample $\delta \sim \mathrm{Unif}\{1, \ldots, d_{\max}\}$; $\delta{=}0$ is excluded because A^+ and A^- would coincide and degenerate the pair. The original supervised sampler $\mathcal{S}_b$ provides the deployment input and SFT target of each policy (Eq. (3)). For standard-SFT SmolVLA and GR00T N1.7, it uses the current input and the current expert action chunk. For the VLASH-based Kinetix, $\pi_{0.5}$, and real-robot settings, it follows the original delay-aware context and target construction, including a context delay sampled from $\{0, \ldots, 4\}$. The temporal contrast horizon δ is sampled independently of any context delay used by $\mathcal{S}_b$. Deployment delays are evaluated over $d \in \{0, \ldots, 4\}$. For LIBERO, the frozen reference policy processes earlier and later inputs from recorded expert trajectories, and its outputs are cached as preference pairs before post-training. Training samples these cached pairs with replacement. With $2{,}000$ gradient steps, $\pi_{0.5}$ uses $16{,}000$ pairs per suite at batch size 8, and GR00T N1.7 uses $8{,}000$ at batch size 4, counting repeated uses. These counts describe training usage, not the number of distinct pairs in the cache. For Kinetix, preference chunks are generated as needed during training from inputs in recorded expert trajectories, rather than loaded from a precomputed cache.

Future SFT. For the GR00T N1.7 baseline, Future SFT uses the future-generated chunk A_t^+ as a supervised target under the deployment input $c_{t,b}^{\mathrm{dep}}$, applying the flow-matching loss of Eq. (4) to this target. The stale-generated chunk A_t^- and the preference loss of Eq. (5) are not used. We keep the original expert-target SFT term alongside the loss on the future-generated targets.

Training. We use AdamW with a $1{,}000$-step linear warmup followed by cosine decay to 1% of the peak learning rate. VLA post-training on LIBERO and the real-robot tasks runs for $2{,}000$ gradient steps. Kinetix uses the 24-epoch schedule. We set $\beta{=}1.0$ and $\lambda_{\mathrm{SFT}}{=}1.0$. Model-specific settings are listed in Table 3. The DPO weight λ_{DPO} is tuned per benchmark. On Kinetix we sweep $\lambda_{\mathrm{DPO}} \in \{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$ and select 0.02 from the robust band $\lambda \in [0.01, 0.05]$. On LIBERO we use $\lambda_{\mathrm{DPO}}{=}0.1$.

Flow-matching loss reduction. For each example, the flow-matching loss of Eq. (4) sums squared errors over the chunk-horizon and action-dimension axes. Using a per-element mean instead of a sum divides the preference margin by the number of included elements, changing the effect of β

unless it is adjusted accordingly. For example, the divisors are 16 for the Kinetix configuration with shape 8×2 and 350 for the $\pi_{0.5}$ LIBERO configuration with shape 50×7. One flow time τ and one Gaussian noise sample ϵ are drawn per preference pair and shared across both candidates, the trainable and reference policies, and the SFT anchor. The outer DPO loss in Eq. (5) is averaged over the minibatch.

Table 3: **Post-training settings for each benchmark and backbone.** The warmup lasts 1,000 gradient steps in all settings. VLA post-training runs for 2,000 gradient steps, while Kinetix uses 24 epochs. Real-robot data collection is described in Appendix H.

	Kinetix	LIBERO			Real robot
Backbone	Flow policy	$\pi_{0.5}$	GR00T N1.7	SmolVLA	$\pi_{0.5}$
SFT initialization	VLASH	VLASH	Standard	Standard	VLASH
Trainable parameters	3M	693M	1620M	97M	693M
Training schedule	24 epochs	2,000 steps	2,000 steps	2,000 steps	2,000 steps
Batch size	512	8	4	8	8
Peak learning rate	3×10^{-4}	1×10^{-6}	1×10^{-5}	1×10^{-6}	1×10^{-5}
Chunk length H	8	50	16	50	50
DPO weight λ_{DPO}	0.02	0.1	0.1	0.1	0.02

All main-paper numbers, figures, and ablations on Kinetix use the 24-epoch cosine setting. The VLM backbone is frozen for LIBERO, while Kinetix uses full fine-tuning.

Evaluation. On LIBERO, each method is evaluated with 500 episodes per suite and delay for all three backbones, using MuJoCo 3.3.7. On Kinetix, each method is evaluated with 1,024 rollouts per task for each delay and execution-horizon setting. Kinetix results are averaged over all 12 tasks.

Benchmark protocols. We use the standard reward, termination, episode-length, success criterion, and action-space conventions from Kinetix (Matthews et al., 2025) and LIBERO (Liu et al., 2023) without modification. No task curriculum, action rescaling, or reward shaping is introduced for either benchmark.

Baselines. Main-table baselines (Naive, VLASH, RTC, BID) are all evaluated on the released VLASH checkpoint. RTC and BID are test-time chunk-level schedulers that replace the naive asynchronous rollover. RTC inpaints the residual chunk prefix at inference boundaries, and BID resamples chunks under forward-contrast and backward-coherence criteria. Both rely on the residual chunk prefix from the previous prediction, which shrinks to 1–3 steps at $d \geq 5$ and becomes too short to support either method. Other concurrent VLA work includes architectural variants (HybridVLA (Liu et al., 2026), WorldVLA (Cen et al., 2025), OpenVLA-OFT (Kim et al., 2025), TinyVLA (Wen et al., 2025), π_0-FAST (Pertsch et al., 2025), ManiFlow (Yan et al., 2025)) and inference-acceleration methods (DeeR-VLA (Yue et al., 2024), SnapFlow (Luan et al., 2026), MP1 (Sheng et al., 2026)). These target architectural design or inference cost rather than delay-robust execution and are orthogonal to our refinement.

C Algorithm: One Gradient Step of DEFLECT

Algorithm 1 spells out one gradient step of DEFLECT: pair construction, shared (τ, ϵ) sampling, the FM-loss evaluation, and the joint SFT+DPO update. In the LIBERO implementation, preference pairs are precomputed from recorded expert trajectories and loaded from the cache during post-training, rather than regenerated at every gradient step.

Algorithm 1 One gradient step of DEFLECT.

Require: SFT-trained policy b; its original supervised sampler $\mathcal{S}_b$; frozen reference copy $\pi_{\mathrm{ref},b}$; trainable parameters θ initialized from b; demonstration dataset $\mathcal{D}$; maximum temporal contrast horizon $d_{\max}$; loss weights $\lambda_{\mathrm{SFT}}, \lambda_{\mathrm{DPO}}, \beta$.

1: Sample timestep t and retrieve instruction l_t.
2: Draw the original base-policy training sample $(c_{t,b}^{\mathrm{dep}}, A_{t,b}^{\mathrm{exp}}) \sim \mathcal{S}_b(\mathcal{D}, t)$.
3: Sample temporal contrast horizon $\delta \sim \mathrm{Unif}\{1, \ldots, d_{\max}\}$. $\{\delta = 0$ gives $A^+ = A^-\}$
4: Sample shared reference-policy randomness ξ.
5: Generate $A^+ \leftarrow \pi_{\mathrm{ref},b}((s_{t+\delta}, o_{t+\delta}), l_t; \xi)$ and $A^- \leftarrow \pi_{\mathrm{ref},b}((s_t, o_t), l_t; \xi)$.
6: Sample $\tau \sim \mathrm{Unif}[0,1]$ and Gaussian ϵ, shared across the preference pair and SFT anchor, and construct $\mathbf{x}_\tau^\pm = \tau A^\pm + (1-\tau)\epsilon$ and $\mathbf{x}_\tau^{\mathrm{exp}} = \tau A_{t,b}^{\mathrm{exp}} + (1-\tau)\epsilon$.
7: Query $v_\theta^\pm = v_\theta(\mathbf{x}_\tau^\pm, c_{t,b}^{\mathrm{dep}}, \tau)$ and the corresponding reference velocities $v_{\mathrm{ref}}^\pm$.
8: Compute $r_\theta^\pm = -\|v_\theta^\pm - (A^\pm - \epsilon)\|_2^2$ and $r_{\mathrm{ref}}^\pm = -\|v_{\mathrm{ref}}^\pm - (A^\pm - \epsilon)\|_2^2$.
9: $\mathcal{L}_{\mathrm{DPO}} \leftarrow -\log\sigma\big(\beta\left[(r_\theta^+ - r_\theta^-) - (r_{\mathrm{ref}}^+ - r_{\mathrm{ref}}^-)\right]\big)$.
10: $\mathcal{L}_{\mathrm{FM}} \leftarrow \|v_\theta(\mathbf{x}_\tau^{\mathrm{exp}}, c_{t,b}^{\mathrm{dep}}, \tau) - (A_{t,b}^{\mathrm{exp}} - \epsilon)\|_2^2$.
11: Update θ using $\nabla_\theta[\lambda_{\mathrm{SFT}}\mathcal{L}_{\mathrm{FM}} + \lambda_{\mathrm{DPO}}\mathcal{L}_{\mathrm{DPO}}]$.

D Flow-Matching Loss as a Likelihood Surrogate for DPO

Section 3.2 uses the flow-matching objective as a tractable surrogate for the action log likelihood required by DPO. We clarify here the distinction between the expected flow-matching objective, its variational interpretation, and the Monte Carlo estimator used in training.

Expected flow-matching objective. For an action chunk A, conditioning input c, flow time $\tau \sim \mathrm{Unif}[0,1]$, and Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$, define

$$\begin{aligned} \ell_{\mathrm{FM}}(\theta; A, c, \tau, \epsilon) &= \|v_\theta(\tau A + (1-\tau)\epsilon, \tau, c) - (A - \epsilon)\|_2^2, \\ \overline{\mathcal{L}}_{\mathrm{FM}}(\theta; A, c) &= \mathbb{E}_{\tau,\epsilon}\left[\ell_{\mathrm{FM}}(\theta; A, c, \tau, \epsilon)\right]. \end{aligned} \tag{7}$$

Under the Gaussian-source, linear-interpolant formulation, prior work motivates this objective through a variational lower bound on the conditional log likelihood: Diffusion-DPO substitutes the diffusion ELBO for the log likelihood (Wallace et al., 2024), and McAllister et al. (2026) derive an analogous bound for this flow family while retaining a θ-dependent gap. We therefore use $-\overline{\mathcal{L}}_{\mathrm{FM}}$ as an implicit likelihood score. We do not identify it with the exact value of $\log \pi_\theta(A \mid c)$.

Reference-calibrated preference surrogate. Define

$$r_\theta^{\mathrm{FM}}(A \mid c) = -\overline{\mathcal{L}}_{\mathrm{FM}}(\theta; A, c). \tag{8}$$

The corresponding pairwise margin is

$$\begin{aligned} M_{\mathrm{FM}} = \ & \left[r_\theta^{\mathrm{FM}}(A^+ \mid c) - r_\theta^{\mathrm{FM}}(A^- \mid c)\right] \\ & - \left[r_{\mathrm{ref}}^{\mathrm{FM}}(A^+ \mid c) - r_{\mathrm{ref}}^{\mathrm{FM}}(A^- \mid c)\right]. \end{aligned} \tag{9}$$

This has the same reference-calibrated pairwise structure as the standard DPO log-ratio margin, but should generally be interpreted as its flow-matching surrogate. Under the stronger assumption that the expected flow-matching objective differs from the exact negative log likelihood only by the same model-independent offset for the trainable and reference policies, these offsets cancel and M_{FM} coincides with the DPO log-ratio margin. Our optimization does not require this stronger equality.

Monte Carlo realization. During training, we estimate $\overline{\mathcal{L}}_{\mathrm{FM}}$ using one sampled pair (τ, ϵ):

$$\widehat{\mathcal{L}}_{\mathrm{FM}}(\theta; A, c) = \ell_{\mathrm{FM}}(\theta; A, c, \tau, \epsilon), \qquad \mathbb{E}[\widehat{\mathcal{L}}_{\mathrm{FM}}] = \overline{\mathcal{L}}_{\mathrm{FM}}. \tag{10}$$

The same (τ, ϵ) is shared across A^+ and A^- and across the trainable and reference policies. This common-random-number construction reduces the variance of the pairwise difference, but it does

not remove Monte Carlo error or turn the estimator into an exact likelihood ratio. Since $-\log\sigma(z)$ is convex in z, by Jensen's inequality the expected single-sample training loss upper-bounds the loss evaluated at the expected margin, the same estimator structure used in Diffusion-DPO (Wallace et al., 2024).

Scope. Accordingly, DEFLECT optimizes a reference-calibrated flow-matching surrogate motivated by the variational likelihood interpretation. We do not claim exact recovery of the action log likelihood or of the original DPO log-ratio objective.

E Generalization across Contrast Horizons

We compare three temporal contrast-horizon settings. **Model A** uses $\delta \in \{1, 2\}$, while the main **Model B** uses $\delta \in \{1, \ldots, 4\}$. **Model C** uses $\delta \in \{1, \ldots, 7\}$ and also changes the asynchronous interval to 8. At deployment delay d=4, Model A improves success over VLASH by 3.2 percentage points despite using contrast horizons of at most two steps during training. This supports generalization beyond the training contrast horizons. Model C achieves 88.2% average success, compared with 89.2% for Model B. Because Model C changes both the contrast-horizon range and the asynchronous interval, this difference cannot be attributed to the contrast-horizon range alone.

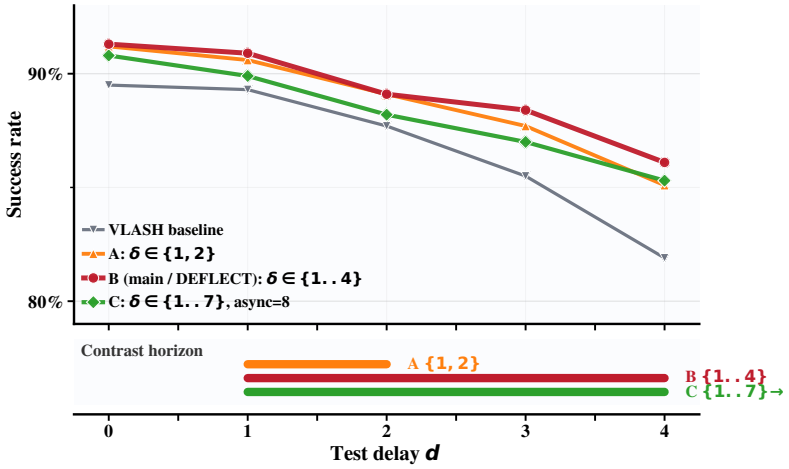


Figure 7: **Generalization beyond the training contrast horizons.** Model A beats VLASH at every deployment delay despite never using a contrast horizon $\delta \geq 3$ in its pairs. The strip below marks each variant's contrast-horizon range; Model C's extends past the plotted range (arrow).

Table 4: **Contrast-horizon generalization** on Kinetix (success rate %). Column subscripts give each variant's δ-range; rows are deployment delays. Model C also changes the asynchronous interval to 8.

d	VLASH	$A_{\{1,2\}}$	$B_{\{1,\ldots,4\}}$	$C_{\{1,\ldots,7\}}$
0	89.5	91.2	91.3	90.8
1	89.3	90.6	90.9	89.9
2	87.7	89.1	89.1	88.2
3	85.5	87.7	88.4	87.0
4	81.9	85.1	86.1	85.3
avg	86.8	88.7	**89.2**	88.2

F Design Ablations

Two design choices define DEFLECT: the SFT anchor is kept alongside the preference term, and both candidates are scored under the same deployment input rather than each under its own genera-

tion context. Table 5 isolates both on Kinetix; every variant starts from the same VLASH checkpoint and shares the training schedule of the main run.

Removing the SFT anchor reduces average success to 15.1% and success at d=4 to 10.6%, showing that the preference objective alone is insufficient in this setting. Scoring each candidate under its own generation input (A^+ under the future observation, A^- under the stale one) instead of the shared deployment input costs -0.7 on average and -1.3 at d=4, and the gap widens monotonically with delay (-0.3, -0.5, -0.7, -0.9, -1.3 over d=$0, \ldots, 4$), consistent with deployment-input alignment mattering more as the two generation contexts diverge.

Table 5: **Design ablations** on Kinetix (success rate $\%$).

Variant	avg(d=0-4)	d=4
VLASH base	86.8	81.9
DEFLECT w/o SFT anchor	15.1	10.6
DEFLECT, scoring under each generation input	88.4	84.8
DEFLECT (full)	**89.2**	**86.1**

G Effect of Temporal Preference-Label Direction

Experimental question. We examine whether DEFLECT's training gains depend on the temporal ordering of preference labels. We compare the original DEFLECT ordering $A^+ \succ A^-$ with randomized and fully reversed orderings, as well as Continued SFT, which retains the supervised anchor without the preference term. The original, randomized, and reversed variants test the role of label direction; Continued SFT provides the comparison against supervised continuation alone.

Controlled training setup. The experiment uses the implementation of DEFLECT described in this paper. Within each suite, we hold the initialization, frozen reference model, supervised-anchor construction, training budget, precision, and hyperparameters fixed. The three preference variants use the same candidate-pair dataset and change only the label ordering: the original ordering prefers A^+ to A^-, random labels randomize this ordering, and reversed labels prefer A^- to A^+ for every pair. Swapping labels affects only the preference term and leaves the SFT anchor's supervised targets unchanged. Continued SFT sets $\lambda_{\mathrm{DPO}} = 0$.

Evaluation. We evaluate GR00T N1.7 on LIBERO-Object, LIBERO-Goal, LIBERO-Spatial, and LIBERO-10 at inference delays $d \in \{1, 2, 3, 4\}$. Each delay column in Table 6 is the equally weighted mean success rate over the four suites, and Avg. additionally averages equally over the four delays. Differences are computed before rounding.

Results. The original temporal labels achieve 85.76% average success, exceeding Continued SFT by 7.05 percentage points, randomized labels by 10.63 points, and reversed labels by 23.64 points. The margin over randomized labels increases from 1.10 points at $d = 1$ to 19.90 points at $d = 4$; the margin over Continued SFT increases from 1.95 to 10.85 points over the same delays. Averaged over the four suites and delays, the ordering is DEFLECT $>$ Continued SFT $>$ random labels $>$ reversed labels.

Interpretation. The label-direction ablation demonstrates that temporal ordering is a consequential component of DEFLECT: preserving the original ordering improves performance over both supervised continuation and randomized or reversed preferences, with larger gains at longer delays.

H Real-Robot Per-Sub-Goal Breakdown

Hardware setup. We use a bimanual AgileX dual-arm platform with 6 DoF per arm, three Intel RealSense D435i cameras, one head-mounted overhead and one wrist-mounted on each arm, 30 Hz control, and a joint-position action space of 7 dimensions per arm and 14 dimensions bimanually, namely 6 joint targets plus 1 gripper open/close per arm. The reference policy is a $\pi_{0.5}$ checkpoint we

Table 6: **Effect of temporal preference-label direction on GR00T N1.7.** Success rates (%) are averaged equally over the four LIBERO suites at each inference delay. Avg. additionally averages equally over $d = 1, 2, 3, 4$. The three preference variants differ only in label ordering, and Continued SFT disables the preference term. Original-label DEFLECT and Continued SFT results are reused from Table 1(a); randomized and reversed labels are additional controls using the same training and evaluation protocol.

Method	$d = 1$	$d = 2$	$d = 3$	$d = 4$	Avg.
DEFLECT (original labels)	**93.60**	**91.05**	**85.30**	**73.10**	**85.76**
Continued SFT	91.65	85.40	75.55	62.25	78.71
Random labels	92.50	83.75	71.10	53.20	75.14
Reversed labels	84.95	71.95	55.10	36.50	62.13

trained in-house on this platform following the VLASH training procedure on our own teleoperation demonstrations; it is not the released LIBERO checkpoint used in the simulation experiments. Policy inference runs on a single NVIDIA RTX 5090. No additional delay is injected in the real-robot experiments. On the RTX 5090, model inference takes 57.1 ms, which at 30 Hz rounds up to 2 control steps (Table 2). All three methods use the same model architecture, number of denoising steps, and asynchronous controller. We compare them under this deployment setup rather than sweeping over injected delays. One implementation detail is worth recording for anyone reproducing these timings: openpi pads absent camera slots with black images and runs the vision encoder over all three regardless, so reducing the camera count does not reduce latency in this stack (61.2 ms with one camera versus 61.1 ms with three, the difference being only the two skipped resizes).

Demonstration data. All demonstrations are collected by leader–follower teleoperation on the same platform at 30 Hz, recording the three camera streams at 480×640 together with the 14-dimensional joint-position state and action. Table 7 lists the per-task budget. No demonstrations are shared across tasks, and no simulation data is used.

Table 7: **Real-robot demonstration data.** Each task is trained from its own demonstrations only.

Task	Demonstrations	Total duration	Mean episode
Conveyor-I	199	75.0 min	22.6 s
Conveyor-II	149	55.7 min	22.4 s
Whack-a-Mole	192	122.7 min	38.3 s

Training setup. For each task we first train the $\pi_{0.5}$ reference policy on that task's demonstrations, then apply DEFLECT to it; the $\pi_{0.5}$ and VLASH rows of Tables 8–9 are the corresponding checkpoints of the same pipeline, so all three methods share one dataset, one backbone, and one control stack. The policy predicts chunks of H=50 actions and the flow-matching ODE is integrated with 10 denoising steps, both at training and at deployment. The DEFLECT stage runs for 2,000 gradient steps at batch size 8 and learning rate 1×10^{-5}, with β=1.0, λ_{SFT}=1.0, and λ_{DPO}=0.02, updating only the action expert while the VLM backbone stays frozen. The learning rate uses a 1,000-step linear warmup followed by cosine decay to 1% of its peak value. The contrast horizon is drawn from $\delta \in \{1, \ldots, 4\}$, the same range used in simulation, and pairs are generated offline from the recorded demonstrations without any additional robot time.

Conveyor-I (simple, two sub-goals). The simpler variant scores two sequential sub-goals: S1 (left arm places cup on conveyor) and S2 (right arm retrieves cup from conveyor). Full-task success is the conjunction.

Conveyor-II (full, three sub-goals). The harder variant inserts a middle sub-goal (S2: select the red rather than green cube and place into cup), so the scoring becomes S1 (cup on conveyor), S2 (correct cube into cup), S3 (cup off conveyor). Sub-goal outcomes are recorded separately over all 30 trials. Later sub-goals are still evaluated when an earlier sub-goal fails. Full-task success requires all three sub-goals to succeed in the same trial.

Table 8: **Conveyor-I per-sub-goal success** (N=30 trials per method). All methods complete S1 (cup placement) without failure, so the only failure mode on this task is S2 (cup retrieval). DEFLECT improves S2 success by $+10$ pp over VLASH (96.7 vs. 86.7) and by $+20$ pp over $\pi_{0.5}$ (96.7 vs. 76.7).

Method	S1: cup on conveyor	S2: cup off conveyor	Full task ($\wedge$ S1, S2)
$\pi_{0.5}$	$30/30$ (100.0)	$23/30$ (76.7)	$23/30$ (76.7)
VLASH	$30/30$ (100.0)	$26/30$ (86.7)	$26/30$ (86.7)
DEFLECT (ours)	$\mathbf{30/30}$ (**100.0**)	$\mathbf{29/30}$ (**96.7**)	$\mathbf{29/30}$ (**96.7**)

Table 9: **Conveyor-II per-sub-goal success** (N=30 trials per method, scored on three sequential sub-goals: **S1** cup on conveyor, **S2** correct cube into cup, **S3** cup off conveyor; full task is the conjunction). The $\pi_{0.5}$ failures are concentrated on S3 (cup retrieval), with a smaller drop on S2 (cube selection). VLASH closes both gaps but still misses $5/30$ on S3 and $1/30$ on S2.

Method	S1	S2	S3	Full task
$\pi_{0.5}$	$30/30$ (100.0)	$22/30$ (73.3)	$15/30$ (50.0)	$14/30$ (46.7)
VLASH	$30/30$ (100.0)	$29/30$ (96.7)	$25/30$ (83.3)	$25/30$ (83.3)
DEFLECT (ours)	$\mathbf{30/30}$ (**100.0**)	$\mathbf{30/30}$ (**100.0**)	$\mathbf{27/30}$ (**90.0**)	$\mathbf{27/30}$ (**90.0**)

I TRAINING-TIME COST OF DEFLECT

DEFLECT leaves inference unchanged. The deployed policy retains the flow-matching ODE, deployment input, and step budget of its SFT initialization, while the additional cost is confined to offline training (Table 10). Counts are per gradient step and per network call, with the batch composition stated here rather than implied. On Kinetix, preference pairs are sampled on the fly, so each gradient step pays 11 frozen-reference forwards (2 chunk generations at 5 denoising steps each, plus one reference-velocity evaluation for the margin in Eq. (5)). On LIBERO, preference pairs are generated by the frozen reference policy from recorded expert trajectories and cached before post-training. The two chunks in each preference pair share the deployment context, flow time, and Gaussian noise, but each is evaluated at its own interpolated input $A_\tau^\pm$ of Eq. (4). A batch of four preference pairs therefore requires velocity predictions for eight candidate inputs. These eight inputs are concatenated into one network call, so each gradient step makes two calls that carry gradients, one over the eight candidate inputs and one for the SFT anchor term, plus one gradient-free reference call over the same eight inputs at the same (τ, ϵ). For GR00T N1.7 the cache holds 12,000 pairs per suite over four suites, and the released checkpoint integrates the flow ODE in 4 steps. Generating both chunks for all 48,000 cached pairs with four flow steps requires $48{,}000 \times 2 \times 4 = 384{,}000$ single-candidate forward evaluations. Batched generation can process multiple such evaluations in one network call. All reference passes carry no gradient.

Table 10: **Training and inference cost** relative to continued SFT on the same backbone (fwd = forward pass). Per-step counts refer to network calls with the batch composition stated in the text; the one-time cost is reported in single-candidate forward evaluations. The LIBERO row is stated for GR00T N1.7, and its one-time column covers all four suites.

	SFT	DEFLECT	one-time
Kinetix, per gradient step	1 fwd+bwd	1 fwd+bwd $+$ 11 ref fwd	—
LIBERO, per gradient step	1 fwd+bwd	2 fwd+bwd $+$ 1 ref fwd	$\approx 3.8 \times 10^5$ ref fwd
Inference	$1\times$	$1\times$	—