

REACT: Rolling Denoising and Dual Decoupling for Reactive Robot Control with VLA Models

Houlong Xiong^{1,2*} Zhenqi Qiu^{2,3*} Zechen Wang^{2*} Suohang Zhang⁴
 Yiyu Ren² Wanting Xu³ Hongfei Niu² Chengyang He⁵
 Ge Sun⁵ Ran Cheng² Qian Zhu^{2†}

¹ Shanghai Jiao Tong University ² PrimeBot ³ ShanghaiTech University
⁴ Zhejiang University ⁵ National University of Singapore
 * Equal contribution † Corresponding author

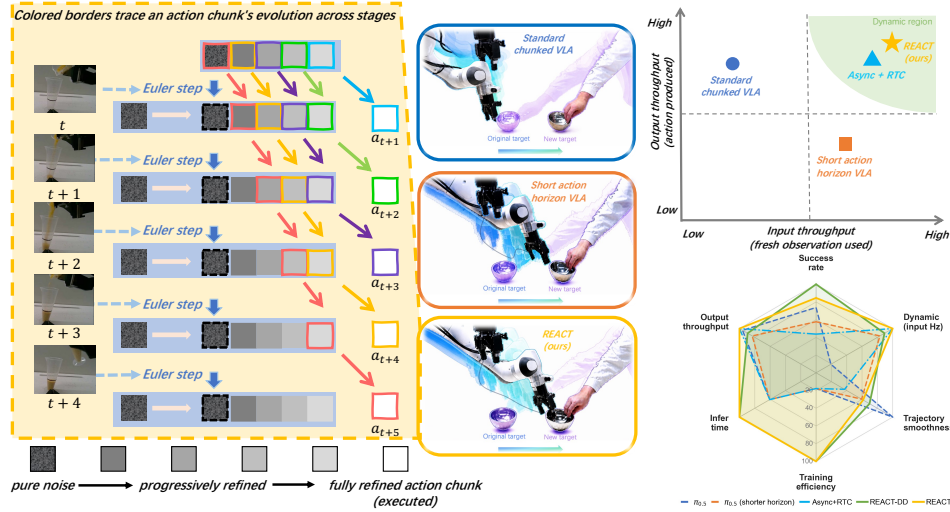


Figure 1: **Overview of REACT.** REACT reformulates flow-based action generation into a receding-horizon rolling denoising process. At each control step, the active horizon is refined conditioned on the latest observation embedding; the oldest refined chunk is executed, while the remaining buffer is shifted forward and replenished with fresh noise. The colored borders track the same action chunk as it is progressively refined across consecutive observations before execution. The right panels summarize throughput and overall performance, showing that REACT reaches the high-input, high-output regime while improving success, reactivity, smoothness, and efficiency.

Abstract: Flow-based vision-language-action (VLA) models generate action chunks for temporally coherent robot motion, but chunked control creates a fundamental closed-loop trade-off: long chunks provide smooth execution, whereas frequent replanning improves reactivity at the cost of action discontinuities. We introduce REACT, a rolling-denoising framework that makes flow-based VLAs more reactive while preserving long-horizon context. Instead of regenerating entire action chunks from scratch, REACT maintains a persistent action buffer with staggered flow timesteps. At each control step, the full horizon is denoised using the latest observation, the cleanest action block is executed, partially refined future blocks are shifted forward, and fresh noise is appended to the tail. As a result, each executed action block is refined across multiple recent observations before deployment. To support real-time control, we further introduce dual decoupling, which separates sensing, VLM encoding, DiT denoising, and action execution, enabling high-frequency observation updates and action streaming under practical compute constraints. Across the RoboTwin 2.0 simulation benchmark and real-world tasks spanning bimanual manipulation and dynamic control on multiple robot platforms, REACT improves task success and reduces reaction latency while producing smoother trajectories than frequent-replanning and asynchronous baselines.

Keywords: Reactive Robot Control, Vision-Language-Action Models, Flow Matching

Project page: react-vla.github.io

1 Introduction

Robots in dynamic environments must react quickly to fresh observations while executing smooth, physically stable actions. Vision-language-action (VLA) models have become a promising framework for general-purpose robot control by unifying perception, language grounding, and action generation. Many modern VLAs [1, 2, 3, 4, 5, 6, 7] generate continuous action chunks with diffusion or flow-matching objectives. Action chunking improves temporal consistency, reduces compounding errors, and preserves multimodal action coherence. However, it also introduces a closed-loop control limitation: once a chunk is being generated or executed, new observations cannot immediately influence the future actions that remain inside the chunk.

Existing methods improve closed-loop reactivity mainly by reducing or hiding inference latency. Asynchronous inference overlaps prediction with execution, but the next chunk may still be generated from stale observations and can introduce discontinuities when the controller switches between chunks. RTC-style methods mitigate this issue by freezing committed actions and inpainting future actions [8, 9], while other methods accelerate VLA inference through token optimization, compression, system-level optimization, or faster flow inference [10, 11, 12, 13, 14]. We argue that latency alone does not fully characterize dynamic robot control. A closed-loop VLA must simultaneously incorporate fresh observations at high frequency and produce executable actions continuously. We capture these two requirements with *input throughput* and *output throughput*.

This exposes a coherence-freshness trade-off: long chunks preserve smooth, mode-consistent motion but reduce observation freshness, whereas frequent replanning improves reactivity but weakens temporal coherence. To address this trade-off, we propose **REACT**, a rolling denoising framework for flow-based VLAs. REACT maintains a persistent action buffer with staggered flow times instead of regenerating each chunk from scratch. At each control step, it denoises the active buffer using the latest observation, executes the cleanest front block, shifts partially refined blocks forward, and appends fresh noise. Thus, each executed block is refined across multiple recent observations while the policy retains a long action horizon.

For deployment, we introduce *dual-decoupling*, which separates sensing, VLM encoding, DiT denoising, and action execution without changing model weights, enabling high-frequency observation updates and continuous action streaming. We also introduce matched staircase training to align flow supervision with rolling inference. Across RoboTwin 2.0 simulation, real-world bimanual manipulation on ARX X5 and Franka R3, and targeted Pour Rice and Reaction Game evaluations, REACT improves the success-reactivity trade-off and trajectory smoothness compared with frequent-replanning and asynchronous baselines. Our contributions are:

- We identify the coherence-freshness trade-off in chunked flow-based VLA inference and introduce input/output throughput as system-level measures of closed-loop responsiveness.
- We propose **REACT**, a rolling denoising mechanism that preserves full-horizon action consistency while refining future actions across consecutive observations.
- We develop dual-decoupled inference and matched staircase training, enabling high-throughput real-time control with improved success, smoothness, and reactivity.

2 Related Work

Chunked VLA policies and dynamic reactivity. Action chunking is a dominant imitation-learning formulation for visuomotor control, producing action sequences rather than single commands [15, 5, 6]. It relies on expressive models such as action-chunking transformers, diffusion policies, and

flow-matching policies [5, 16, 6, 17, 4, 18, 19, 20]. Recent work scales them with pretrained vision-language backbones into vision-language-action (VLA) models for generalist robot control [1, 2, 3, 4, 7, 21, 22, 23, 24, 25, 26], enabling task and embodiment generalization through larger robot datasets and visual-language pretraining [21, 27, 28, 29, 30, 31, 32]. Yet chunked VLA policies create a coherence–reactivity conflict: longer chunks improve temporal consistency, but observations refresh only at boundaries, so scene changes can leave robots executing stale-context actions.

Efficient inference acceleration and asynchronous inference. Prior work improves VLA responsiveness by reducing inference latency or hiding it with asynchronous execution [33, 34]. Latency reduction uses compact VLA architectures [35, 36, 37, 38, 39, 32], token caching or pruning [10, 11, 40], compression [41, 12, 42], and faster generation via speculative decoding, flow acceleration, system optimization, or quantization [43, 44, 45, 14, 13, 46, 47]. Asynchronous execution predicts the next action chunk during current-chunk execution to remove inter-chunk pauses [48, 49, 50, 51, 52], but direct chunk switching can cause observation-state mismatch, discontinuity, performance drops, and trajectory jitter [53, 50]. RTC inpaints future actions conditioned on committed prefixes, while training-time RTC, REMAC, and VLASH condition prediction on executed or predicted prefixes [8, 9, 54, 55]. These methods reduce or mask latency, but input throughput remains tied to action-execution cadence and may hurt performance or trajectory continuity.

Rolling denoising for generative and robotic policies. Rolling denoising extends diffusion sampling from whole-sample refinement to sequential generation, emitting clean variables while appending noisy future ones. Vision and motion methods implement this idea through causal factorization, streaming diffusion, or rolling buffers for frame- or token-level decoding [56, 57, 58, 59, 60]. Robotic diffusion policies roll action buffers, relay noise, or model observation-execution delay to avoid full sequence regeneration [61, 62, 53]. However, they target non-VLA policies and do not address VLA inference, where VLM encoding and DiT denoising are coupled bottlenecks. REACT brings rolling denoising to VLA stacks: executed actions exit, noisy actions enter, and active chunks are refined with updated visual-language embeddings. Dual Decoupling separates input processing from action generation, recovering async-like throughput while preserving trajectory continuity.

3 Methodology

3.1 Rolling Denoising for Flow-Based VLAs

We consider a flow-based VLA policy [7] that maps an observation $\mathbf{o}_i = (\{I_c\}_{c=1}^C, l, s_i)$ to an H -step action chunk $\mathbf{A}_i = [\mathbf{a}_i, \dots, \mathbf{a}_{i+H-1}] \in \mathbb{R}^{H \times D}$ at control iteration i . For a clean action chunk $\mathbf{a}$ and Gaussian noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, conditional flow matching [18] uses the interpolation $\mathbf{x}_\tau = \tau\epsilon + (1 - \tau)\mathbf{a}$, where $\tau=1$ is pure noise and $\tau=0$ is clean action, and learns a velocity field $v_\theta(\mathbf{x}_\tau, \tau, \mathbf{o})$ toward $\mathbf{u}_\tau = \epsilon - \mathbf{a}$. Standard inference initializes a noisy chunk and applies N Euler denoising steps conditioned on the same observation. This keeps all H actions in a shared temporal context, but it ties the entire denoising process to one visual snapshot. REACT keeps the same H -step horizon while rolling the denoising process across consecutive control iterations. Instead of spending the full denoising budget on one fixed observation, each REACT update performs one Euler step on the rolling buffer under the latest observation, emits the executable front block, shifts the remaining blocks forward, and appends fresh tail noise.

At each iteration, we maintain a rolling action buffer $\mathbf{x}^{(i)} = [B_0^{(i)}, B_1^{(i)}, \dots, B_{K-1}^{(i)}]$ with K blocks of length S , so $H = KS$. Block B_0 is closest to execution and B_{K-1} represents the far future. Instead of assigning one flow time to the entire horizon, we assign a per-position staircase time:

$$\tau_j = \frac{\lfloor j/S \rfloor + 1}{K}, \quad j = 0, 1, \dots, H-1, \quad (1)$$

which yields $\tau = [\underbrace{\frac{1}{K}, \dots, \frac{1}{K}}_S, \underbrace{\frac{2}{K}, \dots, \frac{2}{K}}_S, \dots, \underbrace{1, \dots, 1}_S]$. Given the latest $\mathbf{o}_i$, one DiT pass predicts $\mathbf{v}^{(i)} = v_\theta(\mathbf{x}^{(i)}, \tau, \mathbf{o}_i)$, and one Euler update advances every block by one denoising stage:

$$\tilde{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} - \frac{1}{K} \mathbf{v}^{(i)}. \quad (2)$$

Thus, block B_0 reaches $\tau=0$ and becomes executable, while each later block moves to the stage previously occupied by its predecessor.

After the Euler step, REACT executes the updated front block, shifts the remaining blocks left, and appends fresh Gaussian noise to the tail:

$$\mathbf{x}^{(i+1)} = [\tilde{B}_1^{(i)}, \tilde{B}_2^{(i)}, \dots, \tilde{B}_{K-1}^{(i)}, \epsilon_{\text{new}}], \quad (3)$$

where $\tilde{B}_k^{(i)}$ is the k -th block after the update and $\epsilon_{\text{new}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ reinitializes the far future. The buffer therefore preserves the same staircase schedule at every iteration, while the contents of each block progressively move toward execution under successive observations. Steady-state emission begins after $K-1$ warm-up iterations; see Appendix C.4 for the warm-up and inference configuration.

This rolling denoising structure improves observation freshness while preserving the long-horizon mode consistency of action chunking. At each block-rate update, REACT refreshes the observation but still denoises the full H -step rolling buffer rather than an isolated short-horizon block. Thus, adjacent blocks are refined under a shared long-horizon action context, keeping them coupled in the same action mode and smoothing transitions across block boundaries. Meanwhile, each newly appended block is refined across K rolling updates as it moves from the noisy tail to the executable front, so the final executed block is shaped by multiple recent observations instead of a single visual snapshot. This many-observations-to-one-block refinement progressively disambiguates the multimodal action distribution, encourages the block to converge toward a consistent action-space mode, and reduces the instability caused by independently regenerated short-horizon chunks. This also distinguishes REACT from RTC-style inpainting [8, 9]: RTC still generates each new chunk from a single observation and stitches it to a committed prefix, which makes the boundary continuous but does not prevent the new suffix from selecting a different action mode. REACT instead preserves one rolling buffer and refines it progressively under consecutive observations, reducing such mode switches before execution.

3.2 Dual Decoupling for High-Throughput Deployment

Rolling denoising gives REACT the control-quality benefits of rolling multi-observation refinement, but a serialized implementation would behave like a synchronous policy: each update would wait for VLM encoding, DiT denoising, and block execution before the next observation could be used. We therefore use **Dual Decoupling**, an inference-time scheduler that leaves the VLA weights unchanged while separating sensing and VLM prefix computation from DiT denoising and action execution. For block size S , the dual-decoupled scheduler selects one observation every S camera periods, yielding a selected-observation rate of f_{cam}/S . The sensing loop runs at this camera-derived cadence without waiting for block execution, a pool of M VLM workers encodes selected frames asynchronously into a latest-ready cache, and each DiT update consumes the freshest completed embedding together with the rolling buffer. Here, M denotes the number of VLM worker instances rather than the number of GPUs: workers can be colocated on one GPU when memory permits or distributed across multiple GPUs when per-device throughput is limited. When the VLM pool is provisioned so that $M/T_{\text{VLM}} \geq f_{\text{cam}}/S$, the visible observation rate and executable-action rate are bounded by

$$\Phi_{\text{in}} = \min\left(\frac{f_{\text{cam}}}{S}, \frac{1}{T_{\text{DiT}}}\right), \quad \Phi_{\text{out}} = \min\left(f_{\text{cam}}, \frac{S}{T_{\text{DiT}}}\right). \quad (4)$$

The purpose of this decoupling is to recover the input and output throughput that REACT would otherwise lose to serialized synchronization. For example, when the VLM workers and DiT can sustain the selected observation stream, the system processes observations at f_{cam}/S and emits S

executable actions per update, yielding action throughput at the camera cadence. The block size S remains a deployment parameter that sets the selected observation interval, but the main role of Dual Decoupling is not flexible rate control; it is to lift full REACT into the asynchronous-throughput regime while preserving the rolling-buffer refinement that improves success and smoothness. Dual Decoupling trades a small amount of conditioning staleness for this throughput: each DiT update may consume an embedding captured slightly before the update. We log the embedding age $a = t_{\text{consume}} - t_{\text{capture}}$ for every DiT update; with $S=10$ and $M=2$ workers, its mean is 233 ms, its 95th percentile is 383 ms, and its maximum is 416 ms, compared with a selected-observation period of 333 ms, so the conditioning is never more than one selected observation behind. Out-of-order completions were not observed (0.0% of updates) and would be harmless in any case, because the latest-ready cache is monotone in capture timestamp: a late worker never regresses the conditioning. Detailed throughput derivations and the Dual Decoupling timing diagram are provided in Appendix B.

3.3 Staircase Training for Rolling Denoising

Staircase training procedure. REACT trains the action expert on the same per-position flow times used by rolling inference [63]. Given a demo chunk $(\mathbf{a}, \mathbf{o})$, we split the horizon into K blocks, assign τ_j by Eq. (1), sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and form $x_{\tau_j}^{(j)} = \tau_j \epsilon^{(j)} + (1 - \tau_j) a^{(j)}$ for each action position. One forward pass receives the full noisy horizon with per-position time embeddings and predicts all velocities jointly. The loss supervises the K deterministic denoising stages in one example:

$$\mathcal{L}_{\text{REACT}} = \frac{1}{K} \sum_{k=0}^{K-1} \frac{1}{SD} \sum_{j \in B_k} \|v_{\theta}^{(j)} - u_{\tau_j}^{(j)}\|^2, \quad (5)$$

where B_k denotes action positions in block k . This objective covers the time-position pairing used by rolling inference: block k is always queried at flow time $(k+1)/K$ before shifting one position toward execution, so the training loss supervises the same blockwise noise levels used by the deployed scheduler. Appendix C.5 provides the detailed consistency argument.

Deterministic multi-timestep training. Standard flow matching samples a single timestep $\tau \sim p(\tau)$ per training sample, requiring many epochs to cover the full probability path from $\tau=1$ to $\tau=0$. Recent flow-based VLAs similarly densify supervision by evaluating multiple random flow samples per action chunk [64]. REACT can be viewed as a structured variant of this idea: instead of drawing independent timesteps, it supervises the deterministic set $\{1/K, 2/K, \dots, 1\}$ tied to the rolling-buffer positions. This gives each demonstration chunk coverage over all denoising stages required by the deployed scheduler while preserving the train-inference alignment analyzed in Appendix C.5. Empirically, this leads to faster convergence and lower open-loop prediction error (§4).

4 Experiments

Our experiments are designed to answer three questions: (1) What limits the dynamic performance of existing VLA inference schemes? (2) How does REACT address these problems? (3) Beyond addressing these problems, what other benefits does REACT provide?

4.1 Experimental Setup

Policy variants and baselines. All methods build on $\pi_{0.5}$ [7], a flow-based VLA, to isolate the effect of the closed-loop inference schedule. We compare against $\pi_{0.5}$ variants that expose the coherence-reactivity trade-off in chunked control: $\pi_{0.5}$ **H50E50** trains with horizon $H=50$ and executes the full 50-action chunk before re-planning; $\pi_{0.5}$ **H50E10** keeps the same training horizon but executes only the first 10 actions; $\pi_{0.5}$ **H10E10** trains and executes with a shorter $H=10$ horizon; and **Async + Training RTC** uses asynchronous inference with a 10-action inference interval and training-time Real-Time Control [9], conditioning predictions on executed actions while keeping the original training recipe. We denote Dual Decoupling as DD. Our variants are **REACT w/o DD**, which isolates rolling denoising, and full **REACT**, which adds DD; both use block size $S=10$ and

$K=5$ blocks. We choose this setting based on a block-granularity ablation over S and K reported in Appendix E. Model architecture and action-representation details are provided in Appendix C.2.

Benchmarks, platforms, and evaluation protocol. We evaluate in RoboTwin 2.0 simulation [65] and on two real bimanual platforms, **ARX Robotics X5** (ARX X5) and **Franka Research 3** (Franka R3). For simulation, we use seven representative tasks under Clean and Randomized conditions (Table 1), train each policy with 50 expert demonstrations per task, and evaluate 100 rollouts per task-condition pair. For real-world evaluation, we use six platform-task pairs (Figure 2) spanning size-based reasoning, bimanual coordination, and deformable or articulated object manipulation, with 30 trials per task under matched randomized object placements across methods. We report **Success Rate (%)** as the primary task-completion metric and **Trajectory Smoothness** as a motion-quality metric, measured by mean joint-trajectory jerk over successful episodes only. Specialized tasks additionally use task-specific metrics introduced in §4.3. Full task lists, benchmark protocol, hardware specifications, and metric definitions are provided in Appendices C.1, C.6, and C.7.

4.2 General Task Results



Figure 2: **Real-world general manipulation tasks.** ARX X5 (top) and Franka R3 (bottom) perform two shared tasks, Bowl Stacking by Size and Bottle Cap Unscrewing, plus robot-specific hanging tasks: Cable Hanging and Keyring Hanging, respectively.

We first evaluate manipulation performance in simulation and on real robots, following the protocol in §4.1. This broad evaluation compares existing VLA inference schemes and REACT across task success and trajectory smoothness, providing the main evidence for the first two questions.

Table 1, Table 2 and Figure 3 show that dynamic performance of existing chunked VLA inference is constrained by a coherence–freshness conflict, which cannot be resolved by simply increasing replanning frequency. H50E50 executes the full 50-action chunk, preserving the strongest temporal coherence and achieving the lowest simulation jerk among policies (658.9), closest to the demonstrations themselves (570.1), but this long open-loop execution makes the policy less responsive to dynamic scene changes. H50E10 and H10E10 are two naive strategies that improve the observation-refresh rate, but both reduce success and increase jerk. H50E10 drops from 37.43%/11.43% to 33.71%/8.71% on Clean/Randomized simulation, decreases real-world success from 58.7% to 35.9%, and increases simulation jerk to 1463.7. H10E10 degrades further because it also lacks long-horizon context during training, reaching only 10.86%/2.43% in simulation and 21.0% in real-world tasks. These results indicate that short-horizon replanning improves nominal observation freshness, but weakens long-horizon action coherence, amplifies action-mode drift across replanning boundaries, and can

Table 1: **Simulation results on RoboTwin 2.0 benchmark.** Success rate (%) under clean and randomized settings. Best in **bold**, second best underlined.

Task	$\pi_{0.5}$ H50E50		$\pi_{0.5}$ H50E10		$\pi_{0.5}$ H10E10		Async+RTC		REACT w/o DD		REACT	
	Clean	Rand	Clean	Rand	Clean	Rand	Clean	Rand	Clean	Rand	Clean	Rand
beat_block_hammer	17	1	8	0	0	0	13	4	<u>15</u>	<u>2</u>	14	4
click_bell	61	6	<u>71</u>	18	35	0	64	8	89	32	61	<u>22</u>
move_playingcard_away	59	26	71	13	7	1	64	11	71	<u>25</u>	<u>66</u>	22
pick_diverse_bottles	20	3	9	0	0	1	10	<u>6</u>	<u>27</u>	9	33	<u>6</u>
press_stapler	58	23	22	3	9	4	37	13	67	42	<u>63</u>	<u>39</u>
rotate_qrcode	30	3	21	3	4	1	12	<u>8</u>	52	6	<u>46</u>	11
turn_switch	17	18	34	24	21	10	<u>33</u>	24	27	<u>30</u>	31	35
Average	37.43	11.43	33.71	8.71	10.86	2.43	33.29	10.57	49.71	20.86	<u>44.86</u>	<u>19.86</u>

Table 2: **Real-world results on general manipulation tasks.** Success rate (%) across ARX X5 and Franka platforms. Best in **bold**, second best underlined.

Robot	Task	$\pi_{0.5}$ H50E50	$\pi_{0.5}$ H50E10	$\pi_{0.5}$ H10E10	Async+RTC	REACT w/o DD	REACT
ARX X5	Bowl Stacking by Size	73	66	70	36	86	76
	Bottle Cap Unscrewing	36	0	0	0	56	<u>47</u>
	Cable Hanging	56	<u>66</u>	56	3	63	80
	<i>ARX X5 Avg.</i>	55.0	44.0	42.0	13.0	68.3	<u>67.7</u>
Franka	Bowl Stacking by Size	60	50	0	47	66	63
	Bottle Cap Unscrewing	57	0	0	3	<u>63</u>	70
	Keyring Hanging	70	33	0	17	<u>60</u>	53
	<i>Franka Avg.</i>	<u>62.3</u>	27.8	0.0	22.3	63.0	62.0
Overall Avg.		58.7	35.9	21.0	17.7	65.7	<u>64.8</u>

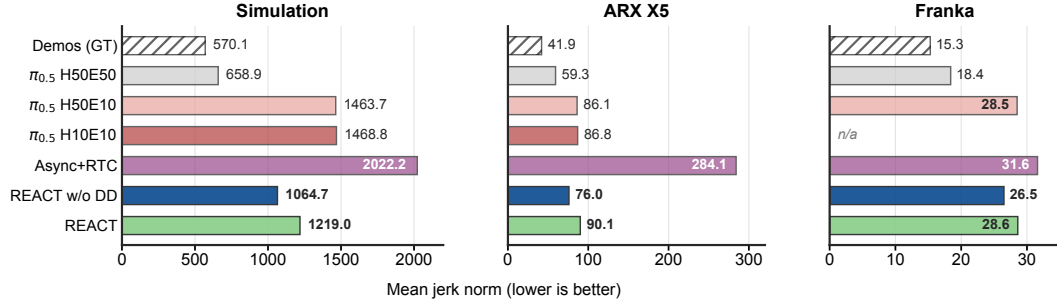


Figure 3: **Trajectory smoothness comparison.** Mean jerk norm averaged over successful episodes in simulation, ARX X5, and Franka settings. Demos (GT) applies the same metric to the imitation-learning demonstrations as a reference. Full per-task values are reported in Appendix D.

cause repeated retries and OOD trapping when the policy fails to make state-changing progress. Appendix A provides a detailed failure analysis.

REACT w/o DD isolates the effect of rolling denoising: it preserves the full $H=50$ rolling buffer and refines future action blocks under multiple observations selected at block intervals, rather than independently sampling short chunks. This preserves long-horizon action consistency and further narrows the action distribution, reducing multimodality. Compared with the three synchronous baselines, REACT w/o DD achieves the highest average simulation success, with 49.71% on Clean and 20.86% on Randomized settings; it also reaches the highest real-world overall success at 65.7%. Although REACT w/o DD is not as smooth as the fully open-loop H50E50 baseline, its simulation jerk is 1064.7, clearly lower than the more reactive synchronous baselines H50E10 (1463.7) and H10E10 (1468.8). This result shows that rolling denoising improves observation freshness while preserving the action consistency and smoothness of long-horizon chunked control.

Full REACT adds Dual Decoupling to raise the throughput of rolling denoising to the same level as asynchronous execution, as shown in Table 5, making it more suitable for real-time deployment. Results show that full REACT substantially outperforms Async+RTC: it improves Clean/Randomized simulation success from 33.29%/10.57% to 44.86%/19.86%, and real-world success from 17.7% to 64.8%. Meanwhile, compared with async execution using RTC, it reduces jerk from 2022.2 to 1219.0 in simulation, from 284.1 to 90.1 on ARX X5, and from 31.6 to 28.6 on Franka. This shows that REACT with Dual Decoupling achieves the same throughput as async execution without causing the large success degradation and trajectory jitter observed in async execution. These differences exceed evaluation-sampling noise: a two-proportion z -test gives $z=4.44$, 4.84 , and 9.10 for clean simulation, randomized simulation, and real-world evaluation, respectively (all $p < 10^{-5}$). Adding Dual Decoupling does not discard the cross-block modal consistency and smoothness brought by rolling denoising. At the same time, the small gap between REACT and REACT w/o DD is an expected deployment trade-off: the latest-ready embedding cache can introduce residual observation-state mismatch, which rolling denoising greatly mitigates but cannot fully eliminate. On these general tasks, where reactivity matters less, this staleness can leave full REACT slightly below REACT w/o

DD; on the specialized dynamic tasks in §4.3, the throughput benefit dominates and the ordering reverses.

4.3 Specialized Task Results

The general results above suggest that dynamic manipulation depends on both observation freshness and temporally coherent execution. We therefore further evaluate two specialized tasks that isolate these requirements: closed-loop control under continuously changing physical states and high-frequency reaction to sudden visual events.

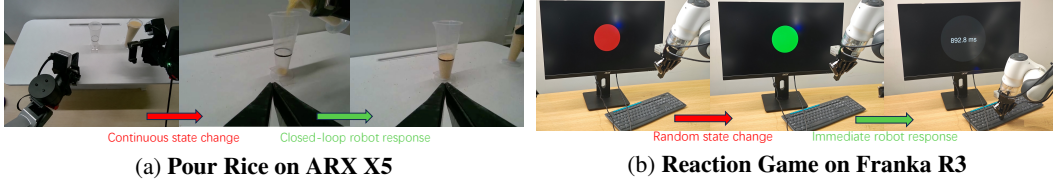


Figure 4: **Specialized dynamic closed-loop control tasks.** (a) **Pour Rice** requires the robot to pour rice into a vessel and stop at the target state, testing whether the policy can react to continuously changing flow while maintaining smooth motion. (b) **Reaction Game** requires the robot to click the mouse when the on-screen signal turns green, directly measuring visual-to-motor reaction latency.

Table 3: **Specialized task results.** Pour Rice reports success rate (%) (higher is better). Reaction Game reports mean $\pm$ standard deviation latency in milliseconds (lower is better).

Task	Metric	$\pi_{0.5}$ H50E50	$\pi_{0.5}$ H50E10	$\pi_{0.5}$ H10E10	Async+RTC	REACT w/o DD	REACT	Human Avg.
Pour Rice (ARX X5)	Success (%) $\uparrow$	3	33	0	0	57	63	—
Reaction Game (Franka R3)	Latency (ms) $\downarrow$	1505 $\pm$ 502	768 $\pm$ 113	772 $\pm$ 119	747 $\pm$ 123	754 $\pm$ 127	734 $\pm$ 94	705 $\pm$ 73

Pour Rice and **Reaction Game** expose complementary dynamic-control demands: continuous physical-state changes and sudden visual events. In Pour Rice, H50E50 reacts too late to the rising rice level and over-pours, revealing the cost of stale observations in a long open-loop segment. H50E10 and H10E10 refresh observations more often but suffer short-horizon failures such as OOD pauses and repeated re-grasp attempts, while Async+RTC is more reactive but often spills due to jerky stitched trajectories. Full REACT achieves the highest Pour Rice success rate (63%) by combining fast closed-loop correction with smoother action evolution, avoiding both stale over-pouring and short-horizon mode oscillations. Reaction Game confirms the same advantage under direct latency stress: H50E50 remains dominated by observation staleness, reaching 1505 ms, whereas full REACT reaches the lowest robot latency at 734 ms, ahead of Async+RTC (747 ms) and only 29 ms above human teleoperation (705 ms). Together, these tasks show that REACT reacts quickly while preserving the smooth, temporally coherent actions needed for physical execution.

4.4 Training Efficiency Analysis

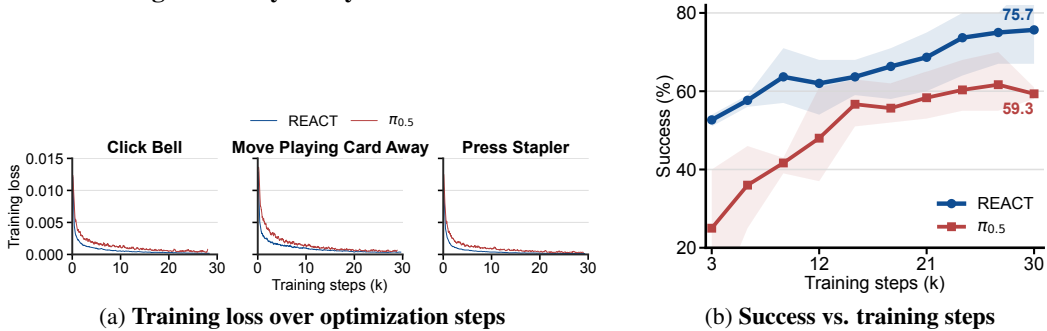


Figure 5: **Training efficiency test.** (a) Training loss over optimization steps. (b) Average closed-loop success rate as a function of training steps; shaded bands show the task-wise range.

Beyond the dynamic-control improvements above, we evaluate whether the training recipe in §3.3 also improves training efficiency. The recipe supervises K deterministic denoising stages in each

forward pass, so it learns the deployed action-flow schedule with fewer optimization steps than standard single-timestep flow training. We test this on three representative RoboTwin tasks, Click Bell, Move Playing Card Away, and Press Stapler, using two main measurements: training loss over optimization steps and closed-loop task success as a function of training steps. Appendix F additionally reports open-loop action prediction on the `demo_randomized` OOD dataset.

Figure 5(a) shows that REACT lowers training loss faster than the $\pi_{0.5}$ baseline on all three tasks. This trend supports the optimization-efficiency interpretation of the staircase objective: by supervising the same deterministic denoising stages used at deployment, the model receives denser guidance for the action-flow schedule it executes. Figure 5(b) shows that this faster optimization also translates to closed-loop success: REACT reaches 52.7% average success after only 3k steps versus 25.0% for the $\pi_{0.5}$ baseline and matches the baseline’s 30k-step performance using only 15k steps. Thus, beyond improving inference-time dynamic control, REACT also makes training more efficient in terms of optimization steps.

5 Conclusion

This paper presented REACT, a rolling-denoising framework for making flow-based VLA policies more reactive while preserving the temporal coherence of long-horizon action chunks. Rather than treating closed-loop control as a choice between stale long chunks and unstable frequent replanning, REACT maintains a persistent action buffer in which future action blocks are progressively refined under successive observations before execution. Together with Dual Decoupling and matched staircase training, this design improves the success–smoothness–reactivity trade-off on RoboTwin 2.0 and in real-world tasks, including bimanual manipulation and dynamic control. These results suggest that real-time VLA control should be evaluated not only by model inference latency, but also by how effectively fresh observations are incorporated into coherent executable actions.

6 Limitations

A primary limitation is that we validate REACT only on the flow-based VLA backbone $\pi_{0.5}$ [7], leaving adaptation to other VLA models untested. This limits empirical evidence for cross-model compatibility, although REACT only assumes a dual-system interface for perception-language-conditioned action generation and is not tied to model scale or a specific backbone. It could therefore apply to dual-system VLAs such as GR00T N1 [66] and X-VLA [67]; we plan to validate REACT on more dual-system VLAs and to extend it to dual-system video-action models such as mimic-video [68] and LingBot-VA [69]. A second limitation is sensitivity to the rolling schedule (K, S) : each policy is trained for one fixed schedule, and we neither evaluate nor claim zero-shot switching between schedules. In practice, S can be chosen from the required reaction timescale and compute budget, and K set to the largest integer with $KS \leq 50$, the pretrained horizon of $\pi_{0.5}$; the same $(K, S) = (5, 10)$ was used in all main experiments without per-task tuning (Appendix E). Future work could train a schedule-conditioned policy by sampling valid block sizes or jointly supervising multiple (K, S) configurations on the same VLM context, similar to the multiple-flow-sample design of MolmoAct2 [64]; a state-conditioned controller could then select among supported schedules according to the complexity and uncertainty of each task phase. Finally, we did not train independent policy seeds, so our statistical comparisons reflect evaluation-rollout sampling noise rather than training-seed robustness.

References

- [1] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

- [2] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.
- [3] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.
- [4] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0 : A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.
- [5] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.
- [6] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.
- [7] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.
- [8] K. Black, M. Galliker, and S. Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.
- [9] K. Black, A. Z. Ren, M. Equi, and S. Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.
- [10] S. Xu, Y. Wang, C. Xia, D. Zhu, T. Huang, and C. Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *Advances in Neural Information Processing Systems*, 38:164448–164473, 2026.
- [11] X. Pei, Y. Chen, S. Xu, Y. Wang, Y. Shi, and C. Xu. Action-aware dynamic pruning for efficient vision-language-action manipulation. *arXiv preprint arXiv:2509.22093*, 2025.
- [12] Y. Yang, Y. Wang, Z. Wen, L. Zhongwei, C. Zou, Z. Zhang, C. Wen, and L. Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40891–40914, 2026.
- [13] Y. Ma, Y. Zhou, Y. Yang, T. Wang, and H. Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.
- [14] Y. Lu, Z. Liu, X. Fan, Z. Yang, J. Hou, J. Li, K. Ding, and H. Zhao. Faster: Rethinking real-time flow vlas. *arXiv preprint arXiv:2603.19199*, 2026.
- [15] L. Lai, A. Z. Huang, and S. J. Gershman. Action chunking as conditional policy compression. *Cognition*, 264:106201, 2025.
- [16] A. George and A. B. Farimani. One act play: Single demonstration behavior cloning with action chunking transformers. *arXiv preprint arXiv:2309.10175*, 2023.
- [17] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.
- [18] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

- [19] C. He, X. Liu, G. M. S. Camps, J. Bruno, G. A. Sartoretti, and M. Schwager. Demystifying robot diffusion policies: Action memorization and a simple lookup table alternative. In *The Fourteenth International Conference on Learning Representations*, 2026.
- [20] C. He, G. S. Camps, X. Liu, M. Schwager, and G. Sartoretti. Latent theory of mind: A decentralized diffusion architecture for cooperative manipulation. *arXiv preprint arXiv:2505.09144*, 2025.
- [21] A. O’Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.
- [22] A.-C. Cheng, Y. Ji, Z. Yang, Z. Gongye, X. Zou, J. Kautz, E. Bıyık, H. Yin, S. Liu, and X. Wang. Navila: Legged robot vision-language-action model for navigation. *arXiv preprint arXiv:2412.04453*, 2024.
- [23] H. Zhen, X. Qiu, P. Chen, J. Yang, X. Yan, Y. Du, Y. Hong, and C. Gan. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.
- [24] R. Zheng, Y. Liang, S. Huang, J. Gao, H. Daumé III, A. Kolobov, F. Huang, and J. Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. In *International Conference on Learning Representations*, volume 2025, pages 54277–54296, 2025.
- [25] S. Liu, L. Wu, B. Li, H. Tan, H. Chen, Z. Wang, K. Xu, H. Su, and J. Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, volume 2025, pages 29982–30009, 2025.
- [26] C. He, G. Sun, Y. Bai, J. Lu, J. Zhao, and G. Sartoretti. Falcon: Actively decoupled visuomotor policies for loco-manipulation with foundation-model-based coordination. *arXiv preprint arXiv:2512.04381*, 2025.
- [27] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.
- [28] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.
- [29] H.-S. Fang, H. Fang, Z. Tang, J. Liu, C. Wang, J. Wang, H. Zhu, and C. Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595*, 2023.
- [30] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.
- [31] G. R. Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, M. Bauza, M. Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.
- [32] M. Reuss, H. Zhou, M. Rühle, Ö. E. Yağmurlu, F. Otto, and R. Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-action flow policies. *arXiv preprint arXiv:2509.04996*, 2025.
- [33] Z. Yu, B. Wang, P. Zeng, H. Zhang, J. Zhang, Z. Wang, L. Gao, J. Song, N. Sebe, and H. T. Shen. A survey on efficient vision-language-action models. *arXiv preprint arXiv:2510.24795*, 2025.

- [34] W. Guan, Q. Hu, A. Li, and J. Cheng. Efficient vision-language-action models for embodied manipulation: A systematic survey. *arXiv preprint arXiv:2510.17111*, 2025.
- [35] P. Budzianowski, W. Maa, M. Freed, J. Mo, W. Hsiao, A. Xie, T. Młoduchowski, V. Tipnis, and B. Bolte. Edgevla: Efficient vision-language-action models. *arXiv preprint arXiv:2507.14049*, 2025.
- [36] J. Wen, Y. Zhu, J. Li, M. Zhu, Z. Tang, K. Wu, Z. Xu, N. Liu, R. Cheng, C. Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.
- [37] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.
- [38] J. Chen, J. Wang, L. Chen, C. Cai, and J. Lu. Nanovla: Routing decoupled vision-language understanding for nano-sized generalist robotic policies. *arXiv preprint arXiv:2510.25122*, 2025.
- [39] J. Williams, K. D. Gupta, R. George, and M. Sarkar. Lite vla: Efficient vision-language-action control on cpu-bound edge robots. *arXiv preprint arXiv:2511.05642*, 2025.
- [40] Z. Liu, Y. Chen, H. Cai, T. Lin, S. Yang, Z. Liu, and B. Zhao. Vla-pruner: Temporal-aware dual-level visual token pruning for efficient vision-language-action inference. *arXiv preprint arXiv:2511.16449*, 2025.
- [41] R. Zhang, M. Dong, Y. Zhang, L. Heng, X. Chi, G. Dai, L. Du, D. Wang, Y. Du, and S. Zhang. Mole-vla: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 18764–18772, 2026.
- [42] Y. Chen and X. Li. Rlrc: Reinforcement learning-based recovery for compressed vision-language-action models. *arXiv preprint arXiv:2506.17639*, 2025.
- [43] S. Wang, R. Yu, Z. Yuan, C. Yu, F. Gao, Y. Wang, and D. F. Wong. Spec-vla: speculative decoding for vision-language-action models with relaxed acceptance. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26916–26928, 2025.
- [44] J. Jia, G. Li, X. Chen, T. An, Y. Hu, J. Li, X. Guo, and J. Yang. Action-to-action flow matching. *arXiv preprint arXiv:2602.07322*, 2026.
- [45] K. Pertsch, K. Stachowicz, B. Ichter, D. Driess, S. Nair, Q. Vuong, O. Mees, C. Finn, and S. Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.
- [46] Y. Dai, H. Gu, T. Wang, Q. Cheng, Y. Zheng, Z. Qiu, L. Gong, W. Lou, and X. Zhou. Actionflow: A pipelined action acceleration for vision language models on edge. *arXiv preprint arXiv:2512.20276*, 2025.
- [47] H. Wang, C. Xiong, R. Wang, and X. Chen. Bitvla: 1-bit vision-language-action models for robotics manipulation. *arXiv preprint arXiv:2506.07530*, 2025.
- [48] Y. Zhao, L. Zhao, B. Cheng, G. Yao, X. Wen, and H. Gao. Vla-rail: A real-time asynchronous inference linker for vla models and robots. *arXiv preprint arXiv:2512.24673*, 2025.
- [49] Y. Jiang, S. Cheng, Y. Ding, F. Gao, and B. Qi. Asyncvla: Asynchronous flow matching for vision-language-action models. *arXiv preprint arXiv:2511.14148*, 2025.

- [50] H. Xie, B. Wen, J. Zheng, Z. Chen, F. Hong, H. Diao, and Z. Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.
- [51] R. Cai, J. Guo, X. He, P. Jin, J. Li, B. Lin, F. Liu, W. Liu, F. Ma, K. Ma, et al. Xiaomi-robotics-0: An open-sourced vision-language-action model with real-time execution. *arXiv preprint arXiv:2602.12684*, 2026.
- [52] Y. Liu, H. Yu, J. Zhao, B. Li, D. Zhang, M. Li, W. Wu, Y. Hu, J. Xie, J. Guo, et al. Learning native continuation for action chunking flow policies. *arXiv preprint arXiv:2602.12978*, 2026.
- [53] A. Liao, D.-K. Kim, M. O. Smith, A.-a. Agha-mohammadi, and S. Omidshafiei. Delay-aware diffusion policy: Bridging the observation-execution gap in dynamic tasks. *arXiv preprint arXiv:2512.07697*, 2025.
- [54] H. Wang, G. Zhang, Y. Yan, Y. Shang, R. R. Kompella, and G. Liu. Real-time robot execution with masked action chunking. *arXiv preprint arXiv:2601.20130*, 2026.
- [55] J. Tang, Y. Sun, Y. Zhao, S. Yang, Y. Lin, Z. Zhang, J. Hou, Y. Lu, Z. Liu, and S. Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.
- [56] D. Ruhe, J. Heek, T. Salimans, and E. Hoogeboom. Rolling diffusion models. *arXiv preprint arXiv:2402.09470*, 2024.
- [57] J. Kim, J. Kang, J. Choi, and B. Han. Fifo-diffusion: Generating infinite videos from text without training. *Advances in Neural Information Processing Systems*, 37:89834–89868, 2024.
- [58] A. Kodaira, T. Hou, J. Hou, M. Georgopoulos, F. Juefei-Xu, M. Tomizuka, and Y. Zhao. Streamdit: Real-time streaming text-to-video generation. *arXiv preprint arXiv:2507.03745*, 2025.
- [59] C. Deng, D. Zhu, K. Li, S. Guang, and H. Fan. Causal diffusion transformers for generative modeling. *arXiv preprint arXiv:2412.12095*, 2024.
- [60] Z. Zhang, R. Liu, R. Hanocka, and K. Aberman. Tedi: Temporally-entangled diffusion for long-term motion synthesis. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024.
- [61] S. H. Høeg, Y. Du, and O. Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.
- [62] Z. Chen, X. Yuan, T. Mu, and H. Su. Responsive noise-relaying diffusion policy: Responsive and efficient visuomotor control. *arXiv preprint arXiv:2502.12724*, 2025.
- [63] B. Chen, D. Martí Monsó, Y. Du, M. Simchowitz, R. Tedrake, and V. Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.
- [64] H. Fang, J. Duan, D. Clay, S. Wang, S. Liu, W. Huang, X. Fan, W.-C. Tsai, S. Chen, Y. R. Wang, et al. Molmoact2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026.
- [65] T. Chen, Z. Chen, B. Chen, Z. Cai, Y. Liu, Z. Li, Q. Liang, X. Lin, Y. Ge, Z. Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.
- [66] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

- [67] J. Zheng, J. Li, Z. Wang, D. Liu, X. Kang, Y. Feng, Y. Zheng, J. Zou, Y. Chen, J. Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.
- [68] J. Pai, L. Achenbach, V. Montesinos, B. Forrai, O. Mees, and E. Nava. mimic-video: Video-action models for generalizable robot control beyond vlas. *arXiv preprint arXiv:2512.15692*, 2025.
- [69] L. Li, Q. Zhang, Y. Luo, S. Yang, R. Wang, F. Han, M. Yu, Z. Gao, N. Xue, X. Zhu, et al. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.
- [70] S. Lian, B. Yu, X. Lin, Z. Shen, L. T. Yang, Y. Jin, H. Liu, C. Wu, H. Yuan, C. Huang, et al. Intentvla: Short-horizon intent modeling for aliased robot manipulation. *arXiv preprint arXiv:2605.14712*, 2026.
- [71] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5745–5753, 2019.

A Failure Case Analysis

We expand on the first analysis in §4.2 to explain why higher nominal dynamics do not necessarily produce better closed-loop manipulation. Across the general manipulation tasks and the two specialized dynamic tasks, the failed baselines show three recurring mechanisms: long open-loop execution remains successful and smooth in standard settings but becomes stale under dynamic evaluation, shortened horizons improve observation freshness while weakening action-distribution support, and asynchronous execution improves output streaming while introducing trajectory stitching artifacts. These mechanisms are visible not only in qualitative rollouts, but also in the paired success, latency, and jerk measurements in Tables 1–3 and Table 8.

Long open-loop chunks remain successful and smooth, but stale. The full-chunk $\pi_{0.5}$ H50E50 baseline illustrates why long action chunks can look strong on standard benchmarks while failing under dynamic evaluation. Because it executes a complete $H=50$ segment before replanning, it preserves strong temporal coherence, achieves the lowest average simulation jerk (658.9), and remains competitive on many standard simulated and real-robot tasks. However, this stability comes from committing to a plan generated from an earlier observation for a long interval. Qualitative instances of this staleness are shown in Figures 6 and 7: in Pour Rice, H50E50 continues pouring even after the rice level exceeds the target mark; in Bowl Stacking by Size, when the bowl position is suddenly changed, it still follows the trajectory planned to grasp the original bowl before correcting. The two specialized dynamic tasks quantify the same issue: in Pour Rice, H50E50 often keeps executing a pouring motion after the rice level has changed, yielding only 3% success; in Reaction Game, its slow observation refresh dominates the response time, producing a 1505 ms latency that trails the human teleoperation average by 800 ms. Thus, long open-loop chunks can preserve action-mode consistency and low jerk in standard settings, but they become stale once success depends on reacting to newly observed state changes within the chunk horizon.

Shortening the execution horizon does not work. H50E10 attempts to improve reactivity in the most direct way: it keeps the $H=50$ training horizon but executes only the first 10 actions before replanning. This shorter execution horizon fails for two reasons. First, many contact-rich manipulation skills require an action prefix to carry the object into a new physical state before the next observation becomes informative. When only a short prefix is executed, the robot often reaches the vicinity of contact but does not complete the state-changing part of the skill, such as closing a stable grasp, initiating a twist, or releasing after rotation. The next replanning step therefore sees an observation that is still close to the previous local state, causing the policy to re-enter the same approach or grasping intent rather than progress to the next stage. This produces the stop-and-replan behavior observed near contact: the arm approaches the object, pauses or retries, and becomes trapped

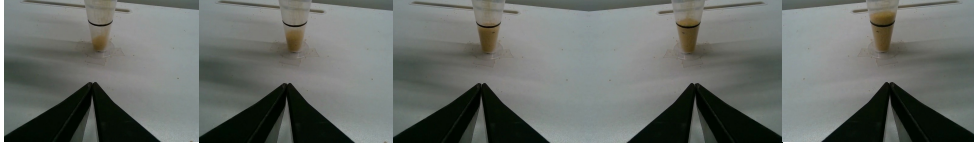


Figure 6: **Over-pouring under long open-loop execution in Pour Rice.** Consecutive frames from the H50E50 baseline show the policy continuing the pouring motion even after the rice level exceeds the target mark, illustrating how stale open-loop execution can overshoot the target state before the next observation update.

in a narrow behavioral region. Once such a short-prefix rollout enters an out-of-distribution contact configuration, frequent replanning further increases the chance of remaining in that region, because each new prediction is conditioned on a poorly supported local observation and again executes only a short recovery-like prefix. The task-level results reflect this failure mode. Compared with H50E50, H50E10 drops from 37.43%/11.43% to 33.71%/8.71% on RoboTwin Clean/Randomized settings and from 58.7% to 35.9% on real robots. The degradation is especially sharp in Bottle Cap Unscrewing, where H50E10 obtains 0% success on both ARX X5 and Franka. This task requires a continuous grasp–twist–release sequence, but the shortened execution horizon repeatedly pulls the policy back into cap-grasping behavior before it can sustain the rotational phase. Figure 8 illustrates this failure: the robot repeatedly returns to the cap-contact region instead of carrying the interaction through the full twisting motion. Second, shortening the execution horizon increases the number of replanning boundaries, which amplifies multimodal action ambiguity. Recent work on aliased robot manipulation shows that similar frame-conditioned observations can admit multiple plausible short-horizon intents, so independently replanned chunks may sample different modes across adjacent steps [70]. In H50E10, frequent replanning exposes the policy to this ambiguity at every 10-action boundary: adjacent chunks can be drawn toward different plausible short-horizon intents for nearly the same visual state, yielding inconsistent inter-chunk transitions even when each individual chunk is locally reasonable. The middle panel of Figure 7 shows the corresponding trajectory-level symptom: after the bowl is moved, the shortened-chunk baseline updates sooner but follows a fragmented stop-and-replan path. This loss of action-mode consistency reduces success and makes the trajectory less smooth, with average simulation jerk increasing from 658.9 for H50E50 to 1463.7 for H50E10. Thus, simply shortening the execution horizon improves nominal observation freshness, but it does not provide the temporally extended action support needed for stable contact-rich manipulation.

Shortening the training horizon is worse. H10E10 removes the mismatch between training and execution horizons, but it also removes the long-horizon action context that chunked VLAs use to coordinate multi-stage behavior and stabilize a consistent short-horizon intent. Instead of executing prefixes of a long plan, the policy is trained from the beginning to predict only short local segments, so it never learns the long-horizon context needed to move through the same contact-rich stages that H50E10 already struggles to complete at execution time. This makes H10E10 more likely to fall into the two failure modes above: repeated local retries near contact and inconsistent transitions across independently predicted short segments. The resulting policy is therefore more brittle than H50E10: success collapses to 10.86%/2.43% on RoboTwin Clean/Randomized settings and to 21.0% on real robots. On Franka, H10E10 reaches 0% on all three general tasks, and its smoothness cannot be reported for those tasks because there are no successful episodes. Overall, shortening the training horizon does not resolve the weaknesses of H50E10; it removes the long-horizon training signal before deployment and leads to even lower success.

Async improves throughput but not consistency. Async+RTC addresses the dynamic-control bottleneck from a systems perspective by hiding inference time behind action execution. With the same $H_e=10$ execution interval used by the short-horizon baselines, asynchronous execution raises the nominal throughput to $\Phi_{in}=3.0$ Hz and $\Phi_{out}=30.0$ Hz in Table 5, compared with 2.5 Hz

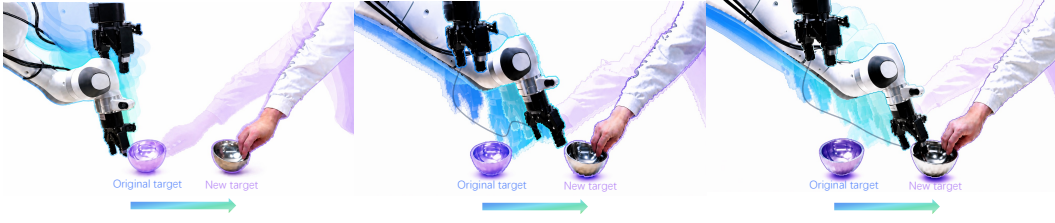


Figure 7: **Trajectory response in Bowl Stacking by Size when the bowl is suddenly moved during grasping.** Left: the original $\pi_{0.5}$ full-chunk policy reacts slowly because it continues executing a stale action chunk. Middle: the shortened action-chunk baseline updates more frequently but produces discontinuous stop-and-replan motion. Right: REACT w/o DD adapts to the moved bowl while maintaining a smoother approach trajectory.

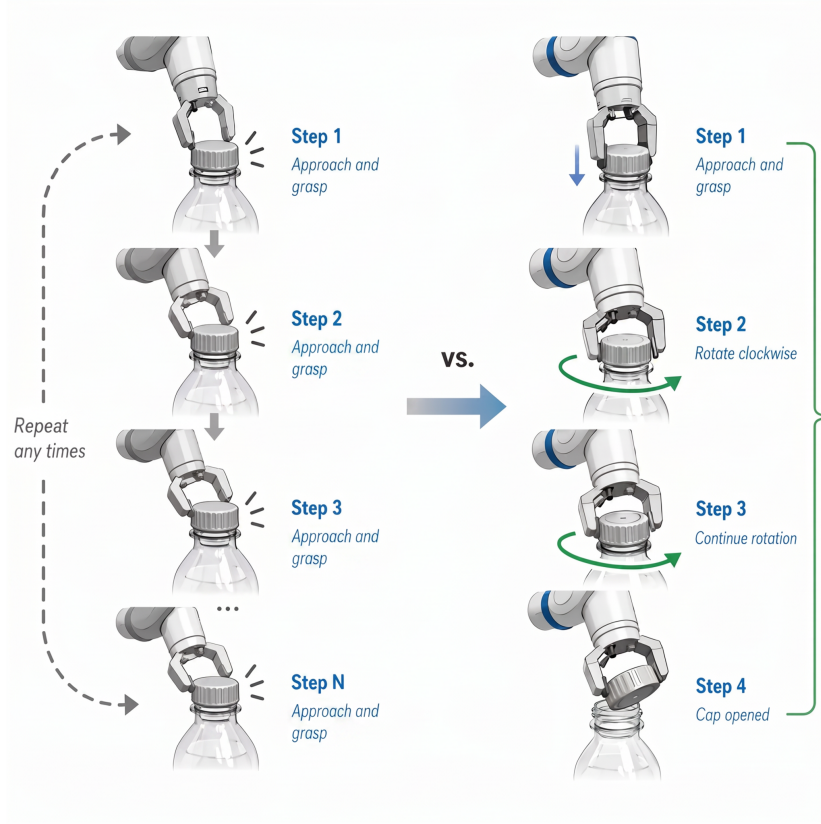


Figure 8: **Repeated cap-contact retries under shortened execution in Bottle Cap Unscrewing.** Representative frames from the H50E10 baseline show the policy repeatedly returning to the cap-contact region instead of sustaining the continuous grasp–twist–release sequence required to open the bottle.

and 25.0 Hz for the corresponding synchronous setting. However, this throughput gain does not remove the control problem created by short executable chunks. Async+RTC still predicts a future chunk from an observation that may no longer match the robot and scene state when that chunk is executed, and the policy must stitch the predicted chunk onto the trajectory already being carried out. RTC reduces this mismatch by accounting for committed actions, but it cannot fully eliminate the inconsistency between the observation used for prediction and the state reached after intervening execution. As a result, Async+RTC often produces abrupt corrections at chunk boundaries and high-jerk motion after contact. The smoothness results support this interpretation: Async+RTC has the highest average simulation jerk among the main methods (2022.2), well above full REACT (1219.0), and on ARX X5 it raises average jerk to 284.1 versus 90.1 for REACT. The same inconsistency hurts task completion, with Async+RTC averaging only 17.7% across real-world general tasks, far below REACT w/o DD (65.7%) and full REACT (64.8%). Per-task ARX results show the characteristic pattern: Async+RTC reaches high jerk on Bowl Stacking by Size (189.3 vs. 80.3 for REACT) and Cable Hanging (378.8 vs. 102.7), while its success falls to 36% and 3% on those tasks; in Pour Rice, it also reaches 0% because stitched, jerky tilting can spill rice or destabilize the flow before the target level is reached.

Implication for REACT. These failure modes show that dynamic VLA control cannot be improved by optimizing observation freshness or throughput alone; the controller must also preserve a coherent long-horizon action context. Short action chunks refresh observations more frequently, but they weaken the long-horizon support needed to carry contact-rich skills through state-changing phases, introduce more replanning boundaries, and amplify multimodal short-horizon ambiguity. Async+RTC improves the systems side of the problem by hiding inference time behind execution and increasing $\Phi_{\text{in}}/\Phi_{\text{out}}$, but it still dispatches short executable chunks and must stitch future predictions onto an evolving robot and scene state. REACT addresses both weaknesses by keeping a rolling $H=50$ action buffer while refining executable blocks across fresh observations. This design lets the policy react to new scene information without repeatedly restarting from independent short-horizon prefixes, preserving the action-distribution support that stabilizes contact-rich manipulation. The empirical pattern matches this mechanism: REACT w/o DD improves real-world success over H50E10, H10E10, and Async+RTC (65.7% vs. 35.9%, 21.0%, and 17.7%), while full REACT recovers camera-rate action streaming and remains far smoother than Async+RTC (simulation jerk 1219.0 vs. 2022.2; ARX jerk 90.1 vs. 284.1). Thus, full REACT is not merely a faster replanning schedule; it couples fresh observations with a persistent long-horizon action buffer, which is the combination needed for responsive and coherent robot manipulation.

B Throughput Comparison

We provide a systematic analysis of input and output throughput across different VLA inference schemes. For a detailed introduction to the synchronous and asynchronous inference mechanisms in action-chunking policies, see Section 3 of FASTER [14]. Let H denote the *action horizon* (total actions predicted per inference), H_e denote the *execution horizon* (actions actually executed before the next inference), T_{VLM} the VLM forward pass latency, T_{DiT} the per-step DiT latency, N the number of denoising steps, $T_{\text{infer}} = T_{\text{VLM}} + N \cdot T_{\text{DiT}}$ the total inference time, f_{ctrl} the robot control frequency used by serialized baselines, and f_{cam} the camera/action-dispatch cadence used by dual-decoupled REACT. For REACT variants with block size $S = H/K$, the observation pipeline selects one camera frame for each rolling block update, giving a selected-observation rate of f_{cam}/S .

B.1 Qualitative Analysis

Synchronous inference. In synchronous mode, inference and execution are strictly sequential: the policy first completes a full inference (VLM encoding + N denoising steps), then executes H_e actions

open-loop before acquiring the next observation. The cycle time and throughput are:

$$T_{\text{cycle}}^{\text{sync}} = T_{\text{VLM}} + N \cdot T_{\text{DiT}} + \frac{H_e}{f_{\text{ctrl}}}, \quad (6)$$

$$\Phi_{\text{in}}^{\text{sync}} = \frac{1}{T_{\text{cycle}}^{\text{sync}}}, \quad \Phi_{\text{out}}^{\text{sync}} = \frac{H_e}{T_{\text{cycle}}^{\text{sync}}}. \quad (7)$$

Asynchronous inference. Asynchronous execution overlaps the next inference with action execution. The policy begins computing the next chunk while still executing the current one. However, each cycle still processes only one observation:

$$T_{\text{cycle}}^{\text{async}} = \max \left(T_{\text{VLM}} + N \cdot T_{\text{DiT}}, \frac{H_e}{f_{\text{ctrl}}} \right), \quad (8)$$

$$\Phi_{\text{in}}^{\text{async}} = \frac{1}{T_{\text{cycle}}^{\text{async}}}, \quad \Phi_{\text{out}}^{\text{async}} = \frac{H_e}{T_{\text{cycle}}^{\text{async}}}. \quad (9)$$

When $T_{\text{infer}} < H_e/f_{\text{ctrl}}$, inference is fully hidden and $T_{\text{cycle}}^{\text{async}} = H_e/f_{\text{ctrl}}$.

Synchronous + acceleration (e.g., quantization, distillation, fewer denoising steps). Acceleration methods reduce T_{infer} by lowering T_{VLM} , T_{DiT} , or N . Let $\alpha \in (0, 1]$ be the speedup factor such that $T'_{\text{infer}} = \alpha \cdot T_{\text{infer}}$. In synchronous mode, where inference and execution are sequential, acceleration directly reduces cycle time:

$$T_{\text{cycle}}^{\text{accel}} = \alpha \cdot T_{\text{infer}} + \frac{H_e}{f_{\text{ctrl}}}, \quad (10)$$

$$\Phi_{\text{in}}^{\text{accel}} = \frac{1}{T_{\text{cycle}}^{\text{accel}}}, \quad \Phi_{\text{out}}^{\text{accel}} = \frac{H_e}{T_{\text{cycle}}^{\text{accel}}}. \quad (11)$$

Note that in asynchronous mode, inference time is already hidden during execution (when $T_{\text{infer}} < H_e/f_{\text{ctrl}}$), so acceleration provides no additional throughput benefit. The primary value of acceleration is therefore in synchronous settings or when $T_{\text{infer}} > H_e/f_{\text{ctrl}}$.

REACT w/o DD. REACT w/o DD restructures inference so that each denoising step consumes a fresh observation and outputs $S = H/K$ actions. Without Dual Decoupling, VLM encoding, DiT denoising, and action execution are sequential:

$$T_{\text{cycle}}^{\text{REACT-w/o-DD}} = T_{\text{VLM}} + T_{\text{DiT}} + \frac{S}{f_{\text{ctrl}}}, \quad (12)$$

$$\Phi_{\text{in}}^{\text{REACT-w/o-DD}} = \frac{1}{T_{\text{cycle}}^{\text{REACT-w/o-DD}}}, \quad \Phi_{\text{out}}^{\text{REACT-w/o-DD}} = \frac{S}{T_{\text{cycle}}^{\text{REACT-w/o-DD}}}. \quad (13)$$

The bottleneck is typically the VLM encoding time T_{VLM} , which dominates the cycle.

Full REACT with Dual Decoupling. Dual Decoupling separates *sensing*, *VLM encoding*, and *DiT denoising* into asynchronous pipelines that run in parallel:

- **Sensing–Action decoupling:** The sensing pipeline selects one available camera frame every S camera periods, i.e., at rate f_{cam}/S , without waiting for block execution or VLM encoding to complete.
- **VLM–DiT decoupling:** A pool of M parallel VLM workers processes selected frames asynchronously; the DiT consumes the most recently completed embedding without waiting for any particular VLM worker.

Let R_{VLM} denote the effective aggregate service rate of the VLM worker pool. The scheduler requires $R_{\text{VLM}} \geq f_{\text{cam}}/S$. If M workers run on independent devices and each worker has latency T_{VLM} , then $R_{\text{VLM}} = M/T_{\text{VLM}}$. When workers are colocated on one GPU, R_{VLM} denotes the effective pool throughput under that colocated deployment. In our real-robot REACT deployment, we use $M=2$ VLM workers colocated on a single NVIDIA RTX 4090 GPU together with the action-expert scheduler. This evaluated layout is not a requirement of the method: REACT also

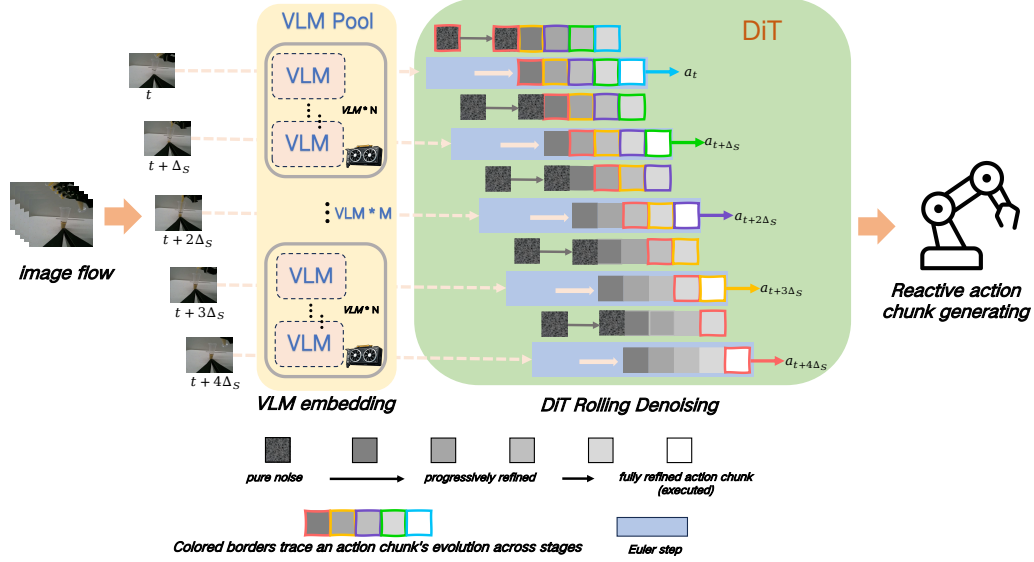


Figure 9: **Dual Decoupling timing diagram.** Sensing, VLM encoding, DiT denoising, and action execution run as separate streams. The scheduler selects observations every $\Delta_S = S/f_{\text{cam}}$ from the camera stream, and parallel VLM workers encode them into a latest-ready embedding cache; each update consumes the freshest completed embedding and the rolling action buffer, then commits the next S executable actions. The architecture removes both the execution wait and the VLM wait from the DiT critical path: camera-rate input occurs in the $S = 1$ setting, while output action throughput reaches the camera cadence whenever the VLM pool and DiT keep up.

supports multi-GPU deployment, where multiple VLM workers can be distributed across devices when additional VLM service rate or memory headroom is needed. In the main setting, $S=10$ and $f_{\text{cam}}=30$ Hz, so the selected-observation rate is only 3 Hz, well below the single-worker reference throughput of approximately $1/T_{\text{VLM}} = 27$ Hz. Thus, the main REACT setting does not rely on linear multi-worker speedup; additional workers mainly provide asynchronous buffering and support smaller-block settings. Under this provisioning condition, the input throughput is bounded by the selected observation frequency and the DiT update rate:

$$\Phi_{\text{in}}^{\text{REACT}} = \min \left(\frac{f_{\text{cam}}}{S}, \frac{1}{T_{\text{DiT}}} \right), \quad (14)$$

$$\Phi_{\text{out}}^{\text{REACT}} = \min \left(f_{\text{cam}}, \frac{S}{T_{\text{DiT}}} \right). \quad (15)$$

When the DiT is also faster than the selected observation schedule, this simplifies to $\Phi_{\text{in}}^{\text{REACT}} = f_{\text{cam}}/S$ and $\Phi_{\text{out}}^{\text{REACT}} = f_{\text{cam}}$. For example, with $f_{\text{cam}} = 30$ Hz and $S=5$, the selected-observation rate is 6 Hz, which is still below the single-worker reference throughput of approximately 27 Hz; with $T_{\text{DiT}} = 3$ ms, the DiT ceiling is about 333 Hz. The system therefore processes selected observations at 6 Hz while streaming executable actions at 30 Hz when the worker pool sustains the selected stream. Thus, under the same provisioning condition, output throughput reaches the camera cadence for any S whenever the VLM pool and DiT sustain the selected stream.

B.2 Summary of Throughput Formulas

Table 4 summarizes the throughput expressions for each inference scheme.

B.3 Comparison with Typical Values

Measurement setup. We base our analysis on $\pi_{0.5}$ [7], a state-of-the-art VLA model with a PaliGemma/SigLIP VLM backbone and a flow-matching action expert. We use empirical reference

Table 4: Throughput formulas for different inference schemes. $T_{\text{infer}} = T_{\text{VLM}} + N \cdot T_{\text{DiT}}$.

Method	Φ_{in} (obs/sec)	Φ_{out} (act/sec)
Synchronous	$\frac{1}{T_{\text{infer}} + H_e/f_{\text{ctrl}}}$	$\frac{H_e}{T_{\text{infer}} + H_e/f_{\text{ctrl}}}$
Sync + Accel (α)	$\frac{1}{\alpha T_{\text{infer}} + H_e/f_{\text{ctrl}}}$	$\frac{H_e}{\alpha T_{\text{infer}} + H_e/f_{\text{ctrl}}}$
Asynchronous	$\frac{1}{\max(T_{\text{infer}}, H_e/f_{\text{ctrl}})}$	$\frac{H_e}{\max(T_{\text{infer}}, H_e/f_{\text{ctrl}})}$
REACT w/o DD	$\frac{1}{T_{\text{VLM}} + T_{\text{DiT}} + S/f_{\text{ctrl}}}$	$\frac{S}{T_{\text{VLM}} + T_{\text{DiT}} + S/f_{\text{ctrl}}}$
REACT	$\min\left(\frac{f_{\text{cam}}}{S}, \frac{1}{T_{\text{DiT}}}\right)$, if $R_{\text{VLM}} \geq f_{\text{cam}}/S$	$\min\left(f_{\text{cam}}, \frac{S}{T_{\text{DiT}}}\right)$

Table 5: Throughput comparison based on $\pi_{0.5}$ ($H=50$, $f_{\text{cam}}=30$ Hz, $f_{\text{ctrl}}=30$ Hz, $T_{\text{VLM}}=37$ ms, $T_{\text{DiT}}=3$ ms, $N=10$).

Method	H_e or S	T_{cycle} (ms)	Φ_{in} (Hz)	Φ_{out} (Hz)
Synchronous	$H_e=50$	1734	0.58	28.8
Synchronous	$H_e=25$	900	1.11	27.8
Synchronous	$H_e=10$	400	2.50	25.0
Synchronous	$H_e=5$	234	4.28	21.4
Sync + Accel ($\alpha=0.5$)	$H_e=50$	1700	0.59	29.4
Sync + Accel ($\alpha=0.5$)	$H_e=25$	867	1.15	28.8
Sync + Accel ($\alpha=0.5$)	$H_e=10$	367	2.73	27.3
Sync + Accel ($\alpha=0.5$)	$H_e=5$	200	5.00	25.0
Asynchronous	$H_e=50$	1667	0.60	30.0
Asynchronous	$H_e=25$	833	1.20	30.0
Asynchronous	$H_e=10$	333	3.00	30.0
Asynchronous	$H_e=5$	167	6.00	30.0
REACT w/o DD ($K=10$)	$S=5$	207	4.84	24.2
REACT w/o DD ($K=5$)	$S=10$	373	2.68	26.8
REACT (main)	$S=10$	—	3.0	30.0
REACT (faster input)	$S=5$	—	6.0	30.0
REACT (upper bound)	$S=1$	—	30.0	30.0

Throughputs are computed from unrounded cycle times; T_{cycle} is rounded to the nearest millisecond for display. For full REACT, T_{cycle} is not listed because the serialized inference–execution cycle is removed; after the one-time warm-up, the steady-state update period is S/f_{cam} .

latencies from a single NVIDIA RTX 4090 GPU after applying `torch.compile` optimization. The VLM forward pass (including vision encoders and language model) takes approximately $T_{\text{VLM}} = 37$ ms; each DiT denoising step takes approximately $T_{\text{DiT}} = 3$ ms. With $N = 10$ denoising steps, the total inference time is $T_{\text{infer}} = 37 + 10 \times 3 = 67$ ms. The camera operates at $f_{\text{cam}} = 30$ Hz, the robot control frequency is $f_{\text{ctrl}} = 30$ Hz, and the action horizon is $H = 50$.

Key observations.

- **Standard chunked VLA inference has very low input throughput.** With the full $H_e=50$ execution horizon, synchronous $\pi_{0.5}$ still dispatches actions at 28.8 Hz, close to the 30 Hz control cadence. However, it incorporates only 0.58 fresh observations per second, so the policy executes a long open-loop segment before visual feedback can affect the next chunk.
- **Model acceleration alone does not remove the closed-loop bottleneck for long chunks.** A $2\times$ inference speedup improves the $H_e=50$ synchronous input rate only from 0.58 Hz to 0.59 Hz, because the cycle time is dominated by executing the long action chunk rather than by the 67 ms model call. Thus faster inference is complementary, but it does not by itself make long-horizon chunked control dynamically responsive.

- **Shortening the action chunk improves input throughput but introduces the control-quality problems analyzed in the experiments.** Reducing H_e lets the policy replan more often: for example, $H_e=5$ raises Φ_{in} to 4.28 Hz. The same serialized schedule also lowers Φ_{out} to 21.4 Hz and creates more chunk boundaries, matching the reduced success and higher jerk observed for short-execution baselines in §4.
- **Asynchronous execution improves both input and output throughput, but it does not eliminate observation–state mismatch.** For $H_e=10$, async raises throughput from 2.50 Hz/25.0 Hz in the synchronous setting to 3.00 Hz/30.0 Hz by hiding inference behind execution. However, each future chunk is still predicted from an observation that may no longer match the robot and scene state at execution time, which is the mismatch analyzed for Async+RTC in the experimental results.

Together, these observations motivate REACT: rolling denoising preserves a long-horizon buffer with multi-observation refinement, while Dual Decoupling recovers the async-like selected-observation and action-dispatch rates shown in Table 5.

C Implementation Details

C.1 Hardware Platforms

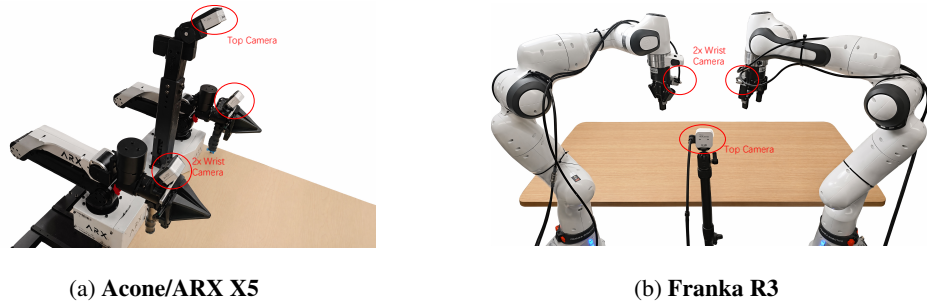


Figure 10: **Hardware platforms.** Representative views of the Acone/ARX X5 bimanual platform and the Franka Research 3 dual-arm platform used in the real-robot evaluation.

Acone/ARX X5. The Acone/ARX X5 platform is a bimanual manipulation system following the ALOHA-style dual-arm setup [5]. It consists of two 6-DoF robotic arms, each equipped with a 1-DoF parallel gripper. The workspace is monitored at 30 Hz by three RGB cameras, including a front-facing scene camera and two wrist-mounted cameras for local manipulation feedback. The robot is controlled at 30 Hz.

Franka Research 3. We deploy a dual-arm setup comprising two Franka Research 3 (FR3) collaborative robots. Each FR3 arm provides 7-DoF manipulation capability with torque-controlled joints and a parallel gripper. The workspace is monitored at 30 Hz by three RGB cameras, including a front-facing scene camera and two wrist-mounted cameras for local manipulation feedback. Control commands are sent at 30 Hz through the Franka Control Interface using Cartesian impedance control.

Computational resources. For non-decoupled baselines and REACT w/o DD, inference runs on a single NVIDIA RTX 4090 GPU (24 GB VRAM), with the VLM backbone and action expert loaded simultaneously. For full REACT, our evaluated deployment also uses a single NVIDIA RTX 4090 GPU (24 GB VRAM), colocating the action expert with $M=2$ asynchronous VLM workers. This evaluated configuration keeps the same single-GPU hardware budget as the non-decoupled baselines; the scheduler only requires the effective VLM-pool service rate R_{VLM} to satisfy $R_{VLM} \geq f_{cam}/S$. For the main setting ($S=10$, $f_{cam}=30$ Hz), the selected-observation rate is 3 Hz, which is well below the single-worker reference throughput of approximately $1/T_{VLM} = 27$ Hz reported in Appendix B. Thus, the evaluated single-GPU configuration satisfies the scheduler requirement without relying on linear throughput scaling from colocated workers. REACT also supports multi-GPU deployment,

Table 6: Training hyperparameters used for simulation and real-robot policies.

Hyperparameter	Value
Peak learning rate	2.5×10^{-5}
Final decay learning rate	2.5×10^{-6}
LR scheduler	Cosine decay with linear warmup
Warmup fraction	10% of the total training schedule
Optimizer	AdamW
AdamW β_1	0.9
AdamW β_2	0.95
AdamW ϵ	1×10^{-8}
Weight decay	1×10^{-10}
Simulation batch size	32
Simulation training length	30,000 optimization steps
Real-robot training length	20 epochs
Acone/ARX X5 batch size	128
Franka R3 batch size	256 for all real-robot tasks except Reaction Game; 32 for Reaction Game
Gradient clipping	1.0
Mixed precision	bfloat16

where multiple VLM workers can be distributed across devices when additional VLM service rate or memory headroom is needed.

C.2 Model Architecture

Base VLA. All policy variants use $\pi_{0.5}$ [7] as the base model, matching the experimental setup in §4.1. The model combines a PaliGemma-3B vision-language model with a 300M-parameter action expert. The VLM encodes multi-view images with SigLIP and fuses them with language instructions, while the action expert generates H -step action chunks. In standard $\pi_{0.5}$ inference, each generated chunk uses $N=10$ flow-matching denoising iterations. We keep the backbone, tokenizer, image preprocessing, and action parameterization unchanged across baselines, REACT w/o DD, and full REACT. The $\pi_{0.5}$ baseline schedules use the standard $N=10$ denoising budget per generated chunk, whereas REACT w/o DD and full REACT use the same action expert in a rolling schedule: each control update performs only one Euler denoising step on the H -step buffer. An emitted REACT block has received multiple refinements because it persists across successive updates, not because a single REACT update runs a multi-step denoising loop. REACT therefore changes the training timestep layout and the inference-time scheduling of the action expert rather than the model architecture.

Action representation. For Acone/ARX X5, each action is represented in joint space: the two 6-DoF arm joint states plus two gripper states form a 14-dimensional vector (12+2). For Franka R3, each action is represented in end-effector space: each arm uses Cartesian position and the continuous 6D rotation representation of Zhou et al. [71], which avoids the discontinuities of lower-dimensional 3D rotation parameterizations in neural-network regression; the two gripper states are appended, yielding a 20-dimensional vector ($2 \times (3+6) + 2$). All REACT configurations partition the same $H=50$ horizon into K contiguous blocks of size S so that $H = KS$.

C.3 Training Configuration

The training hyperparameters are summarized in Table 6.

C.4 Inference Configuration

Rolling-buffer warm-up. At the beginning of each episode, the REACT buffer is initialized from the robot reset state rather than from pure noise at every position. Let $\bar{s}_0 \in \mathbb{R}^{S \times D}$ denote the initial robot state tiled across one action block, and let $\tau_k = (k+1)/K$ be the blockwise staircase time. The

initial block at position k is

$$B_k^{(0)} = (1 - \tau_k)\bar{s}_0 + \tau_k\epsilon_k, \quad \epsilon_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad k = 0, \dots, K-1.$$

Thus, for example, when $K=5$ the buffer begins with $0.8\bar{s}_0 + 0.2\epsilon$ at the front and progresses through $0.6\bar{s}_0 + 0.4\epsilon, \dots$, to pure noise at the tail. During warm-up, the policy runs the same rolling update used in steady state, but conditions the $K-1$ updates on the initial observation $\mathbf{o}_0 = (\{I_c^0\}_{c=1}^C, l, s_0)$: it performs one action-expert denoising pass, shifts the buffer left, and appends a fresh noise block at the tail. The front block produced during these warm-up iterations is not used for evaluation; the controller holds its reset pose or a task-specific safe pose while the pipeline is filled. After $K-1$ warm-up iterations, the next emitted block has traversed all K staircase stages from the tail noise block to the executable front block. This one-time startup cost is incurred only at episode reset; all subsequent control iterations emit one block of S actions while incorporating the latest available observation.

Dual Decoupling workers. For full REACT, VLM workers run asynchronously and publish completed visual-language embeddings to the latest-ready cache described in §3.2. This scheduling removes the VLM pass from the DiT critical path while preserving the same policy weights as REACT w/o DD.

Latency reporting. Table 7 reports the synchronous per-update latency. Because each REACT update performs only one action-expert denoising step, REACT also obtains the inference-speed benefit targeted by recent single-step or near-single-step flow-policy methods [44, 14].

Table 7: **Real-robot per-update inference latency.** Mean synchronous policy-update latency measured with `torch.compile` enabled.

Policy	Latency (ms) ↓	Speedup	Notes
Standard $\pi_{0.5}$	66.04	1.00×	VLM embedding + 10 denoising steps.
REACT w/o DD	39.96	1.65×	VLM embedding + single denoising step.

C.5 Train-Inference Consistency

Let $k \in \{0, 1, \dots, K-1\}$ denote the block position in the rolling buffer, ordered from the executable front block to the far-future tail block. The staircase training objective aligns with rolling inference by using the same time-position grid at every update and along each block’s rolling trajectory.

Per-step alignment. Before each REACT inference update, block k is queried with flow time $\tau_k = (k+1)/K$, matching the staircase profile in Eq. (1). The action expert therefore receives the same relative block position and flow-time pair during deployment as it receives during training. After one Euler update with step size $1/K$, the front block moves from $\tau_0 = 1/K$ to the clean endpoint and is dispatched for execution, while every later block advances to the denoising stage previously occupied by its predecessor. Thus, every time-position pair queried during a REACT update corresponds to a blockwise noise level supervised by the training loss of Eq. (5).

Rolling-buffer invariant. The block shift preserves this alignment across time. After the Euler update, REACT removes the executed front block, shifts blocks $1, \dots, K-1$ one position toward the front, and appends a fresh Gaussian-noise block at the tail. For $k = 0, \dots, K-2$, the shifted block that was previously at flow time $(k+2)/K$ now occupies position k and has just been advanced by one Euler step, so its effective flow time becomes $(k+1)/K$. The appended tail block is initialized as pure noise and occupies flow time 1. By induction, the rolling buffer recovers the same staircase profile before every inference update.

Block-trajectory alignment. A block enters the buffer at position $K-1$ as pure noise, receives one observation-conditioned denoising update at each subsequent control iteration, and exits at position 0 after traversing the K staircase stages. Training does not require a separate inference-specific loss for this trajectory: the staircase objective supervises all intermediate stages $\{1/K, 2/K, \dots, 1\}$ within each demonstration chunk. Consequently, every time-position query that a block receives during

rolling inference has a corresponding supervised blockwise noise level in training, and the deployed scheduler does not query unseen time-position combinations.

C.6 Benchmarks and Evaluation Protocol

Simulation benchmark. RoboTwin 2.0 [65] provides a scalable bimanual manipulation benchmark with 50 complex tasks, expert demonstrations, and domain randomization for robust policy evaluation. We use seven representative RoboTwin 2.0 tasks: `beat_block_hammer`, `click_bell`, `move_playingcard_away`, `pick_diverse_bottles`, `press_stapler`, `rotate_qrcode`, and `turn_switch`. These tasks cover tool use, precision contact, grasping, card manipulation, and articulated-object manipulation. Each policy is trained with 50 expert demonstrations per task from the official `demo_clean` split and evaluated under two conditions: **Clean**, which uses the default scene distribution, and **Randomized**, which applies strong domain randomization. We run 100 evaluation rollouts per task and condition following the default RoboTwin protocol.

Real-world tasks. For real-world experiments, we deploy on two bimanual robot platforms: ARX Robotics X5 (ARX X5), an ALOHA-style dual-arm platform, and Franka Research 3 (Franka R3), a dual-arm setup. The two platforms share **Bowl Stacking by Size** and **Bottle Cap Unscrewing**; ARX X5 additionally evaluates **Cable Hanging**, while Franka R3 evaluates **Keyring Hanging**. Together, these real-world tasks test size-based reasoning, bimanual coordination, and deformable or articulated object manipulation. Each task is tested over 30 trials, and all methods use the same randomized object placements to isolate policy differences from scene variation.

C.7 Evaluation Metrics

Success rate. For task-completion evaluations, success rate is the percentage of completed trials among all evaluation trials under the corresponding protocol. The simulation benchmark uses 100 rollouts per task and condition, while each real-world general task uses 30 trials.

Trajectory smoothness. Trajectory smoothness measures motion quality as the mean jerk norm of the executed joint trajectory. Given a trajectory $\mathbf{x}(t) \in \mathbb{R}^D$ over T timesteps, we estimate jerk $\mathbf{j}(t)$ by applying central differences three times to approximate the third derivative and report $\frac{1}{T} \sum_t \|\mathbf{j}(t)\|_2$. Lower values indicate smoother motion. We average this metric only over successful episodes, so comparisons are not inflated by methods that fail early or terminate before completing the task.

Specialized-task metrics. Pour Rice reports task success rate on the ARX X5 platform. Reaction Game reports mean $\pm$ standard deviation latency in milliseconds on the Franka R3 platform, measured from the visual stimulus change to the robot mouse click. We additionally discuss excess latency over the human teleoperation average in the main text to isolate the policy-induced component of the reaction time.

Reaction Game timing apparatus. We measure Reaction Game latency with a single closed-loop timing script that drives both the visual stimulus and the latency readout on the same monitor. The script controls a colored indicator displayed on the monitor and switches it from a neutral color to green at a randomized onset time t_{green} , recorded with the system monotonic clock at the moment the green frame is committed to the display buffer. A USB keyboard sits in front of the robot, and the policy outputs end-effector actions that drive the gripper to press a designated key on this keyboard. A background keyboard listener registers the keypress timestamp t_{click} at the operating-system event level. The reported latency is $\Delta t = t_{\text{click}} - t_{\text{green}}$ in milliseconds. After the keypress is detected, the script immediately renders Δt as a large numeric overlay on the same monitor, so each trial’s measured latency is both logged to disk and visible on-screen for verification during evaluation. The human teleoperation baseline uses the identical script, monitor, keyboard, and timing path, with a human operator teleoperating the same Franka R3 arm to press the same key; this controls for monitor refresh, keyboard polling, and USB transport delays, so excess-latency comparisons against the human average isolate the policy-induced component of the reaction time.

D Full Trajectory Smoothness Results

Table 8 reports the per-task trajectory smoothness values summarized in Figure 3. For reference, the same metric computed on the imitation-learning demonstrations averages 570.1 in simulation, 41.9 on ARX X5, and 15.3 on Franka; the full-execution-horizon H50E50 policy is closest to these values, while full REACT substantially reduces jerk relative to the real-time Async+RTC baseline.

Table 8: **Full trajectory smoothness results.** Mean jerk norm of executed joint trajectories for each task and platform; lower indicates smoother motion. Best in **bold**, second best underlined.

Setting	Task	$\pi_{0.5}$ H50E50	$\pi_{0.5}$ H50E10	$\pi_{0.5}$ H10E10	Async+RTC	REACT w/o DD	REACT
Simulation	beat_block_hammer	463.4	996.7	–	1755.0	<u>664.0</u>	775.9
	click_bell	502.4	1087.2	1341.8	1946.4	<u>690.8</u>	712.0
	move_playingcard_away	814.3	2012.7	1784.5	1832.2	<u>1165.8</u>	1341.8
	pick_diverse_bottles	857.7	1391.2	–	1688.0	<u>1253.4</u>	1621.8
	press_stapler	567.9	1295.5	1425.7	3168.2	<u>1136.4</u>	1204.7
	rotate_qrcode	965.5	1663.9	1638.8	1944.5	<u>1296.1</u>	1422.3
	turn_switch	441.1	1798.8	<u>1153.1</u>	1821.2	1246.7	1454.3
	<i>Simulation Avg.</i>	658.9	1463.7	1468.8	2022.2	<u>1064.7</u>	1219.0
ARX X5	Bowl Stacking by Size	55.3	76.8	78.5	189.3	<u>70.9</u>	80.3
	Bottle Cap Unscrewing	53.7	–	–	–	<u>69.4</u>	87.3
	Cable Hanging	68.9	95.4	95.0	378.8	<u>87.6</u>	102.7
	<i>ARX X5 Avg.</i>	59.3	86.1	86.8	284.1	<u>76.0</u>	90.1
Franka	Bowl Stacking by Size	15.1	27.3	–	28.5	<u>24.3</u>	27.6
	Bottle Cap Unscrewing	28.1	–	–	34.2	<u>29.8</u>	30.4
	Keyring Hanging	12.1	29.7	–	32.0	<u>25.5</u>	27.7
	<i>Franka Avg.</i>	18.4	28.5	–	31.6	<u>26.5</u>	28.6

E Ablation Studies

We ablate the two core granularity parameters in REACT: the block length S and the number of rolling blocks K . The block length S controls how dynamic the controller is, because the policy ingests a new observation and emits the next action block every S executed actions; smaller S therefore gives more frequent observation updates and more reactive closed-loop control. The block count K controls how many observations jointly shape each executed action block. A block appended as noise at the tail is refined once per rolling update and reaches execution after traversing K staircase stages, so each output action block has been influenced by K recent observations and has undergone K denoising refinements. Larger K therefore corresponds to stronger action-space convergence: more visual evidence is accumulated before an action is committed.

Scope of the evaluated $S \times K$ grid. The hatched cells in Figure 11 are not arbitrary omissions; they are excluded by a backbone-level constraint of $\pi_{0.5}$. We initialize from the publicly released $\pi_{0.5}$ VLA checkpoint, which was pretrained end-to-end (VLM backbone, cross-modal conditioning, and flow-matching action expert) at a fixed action horizon of 50. The horizon H is therefore not a free hyperparameter of the action expert alone: the per-position time embeddings, the VLM–action cross-attention layout, and the action-token positional encodings are all baked in at $H=50$ during VLA pretraining. To inherit this pretrained VLA prior, every REACT configuration must satisfy $H = K \cdot S \leq 50$. Consequently, configurations such as $K=5, S=5$ ($H=25$), $K=10, S=5$ ($H=50$), $K=1, S \leq 50$, and other $K \times S \leq 50$ corners are admissible, while pairs with $K \cdot S > 50$ (e.g., $K=10, S=10$, $K=5, S=20$, $K=5, S=50$) would require re-pretraining the *entire* VLA from scratch at a longer horizon—not only the action expert but also the cross-modal conditioning and positional encodings tied to action-token layout. Such re-pretraining would consume orders of magnitude more compute than our ablation budget and, more importantly, would conflate the effect of the rolling schedule with that of a different pretrained VLA, making the comparison non-isolating. We therefore restrict the ablation to the $K \cdot S \leq 50$ region and report the densest grid within it. The same constraint explains the upper-triangular appearance of the heatmap: cells in the disallowed region cannot be evaluated under the same pretrained VLA, not because their performance is uninteresting. Within this admissible region, $S=10, K=5$ saturates the full $H=50$ pretrained horizon while still providing

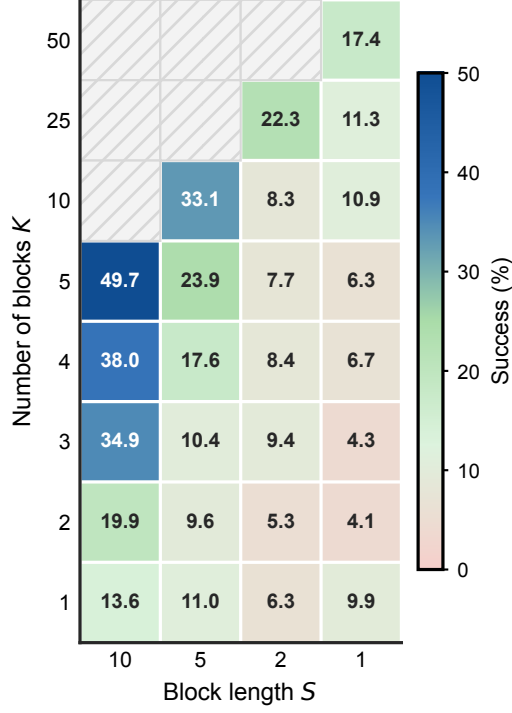


Figure 11: **Ablation on block length and block count.** Mean RoboTwin clean success rate for each $S \times K$ configuration; hatched cells indicate configurations not evaluated.

$K=5$ rolling refinements per emitted block, which is the largest multi-observation convergence budget the pretrained horizon allows.

Aggregate trend. Figure 11 shows both effects. Holding the block length fixed at $S=10$, increasing K consistently improves success: the best setting $K=5$ reaches 49.7%, compared with 38.0% for $K=4$, 34.9% for $K=3$, and 19.9% for $K=2$. This trend supports the role of K as a convergence parameter: when each action block is denoised through more rolling stages and conditioned on more observations, the final action is more reliable. At the same time, the ablation shows that REACT still faces a dynamism–success trade-off. Smaller S makes the policy more dynamic by refreshing observations more often, but the $S=1$ and other short-block configurations do not surpass $S=10, K=5$, indicating that excessive reactivity can reduce the temporal/action context available for stable execution. We therefore use $S=10, K=5$ as the final configuration because it achieves the best observed success while preserving frequent enough observation updates and sufficient multi-observation denoising for action-space convergence.

Schedule sensitivity and practical selection. Success is sensitive to (K, S) : at fixed $S=10$, mean success increases from 13.6% to 49.7% as K grows from 1 to 5, and at fixed $K=5$, it increases from 6.3% to 49.7% as S grows from 1 to 10. A larger K gives each block more observation-conditioned refinements, whereas, at fixed K , a very small S shortens the rolling horizon $H=KS$ and reduces action context and temporal coherence. This suggests a simple recipe before training: choose S according to the required reaction timescale and the available compute budget, since smaller S requires more frequent VLM and DiT updates, and then use the largest integer K satisfying $KS \leq 50$. We did not tune (K, S) per task or platform: the same $(K, S) = (5, 10)$ setting is used in all main experiments and is optimal or tied for optimal on 6 of the 7 RoboTwin tasks, the exception being `turn_switch`. The single denoising update per control cycle is not an independently tuned hyperparameter: under the matched K -stage schedule, each block undergoes exactly K refinements before execution.

Per-task trend. Figure 12 decomposes the mean success rate from Figure 11 into per-task heatmaps, revealing that the best block granularity depends on the manipulation structure of each task.

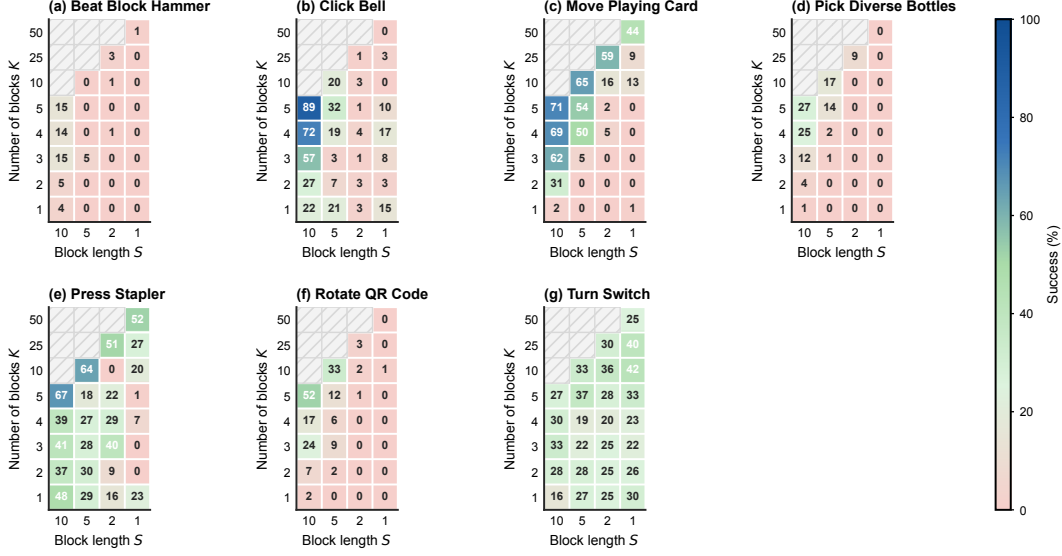


Figure 12: **Per-task ablation heatmaps.** Success rate for each of the seven RoboTwin tasks as a function of block length S and block count K . Hatched cells indicate configurations not evaluated. The color scale is shared across all panels (0–100%).

F OOD Dataset Action Prediction Open-Loop MSE

Figure 13 provides the aggregate open-loop action-prediction diagnostic referenced in §4.4. All open-loop MSE values and qualitative predictions in this appendix are evaluated using the final training checkpoints.

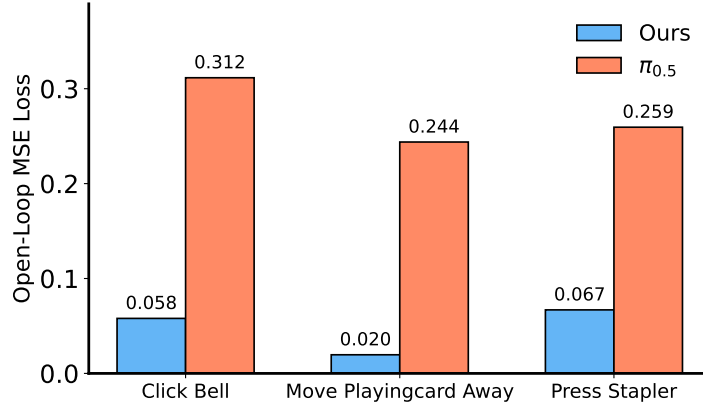
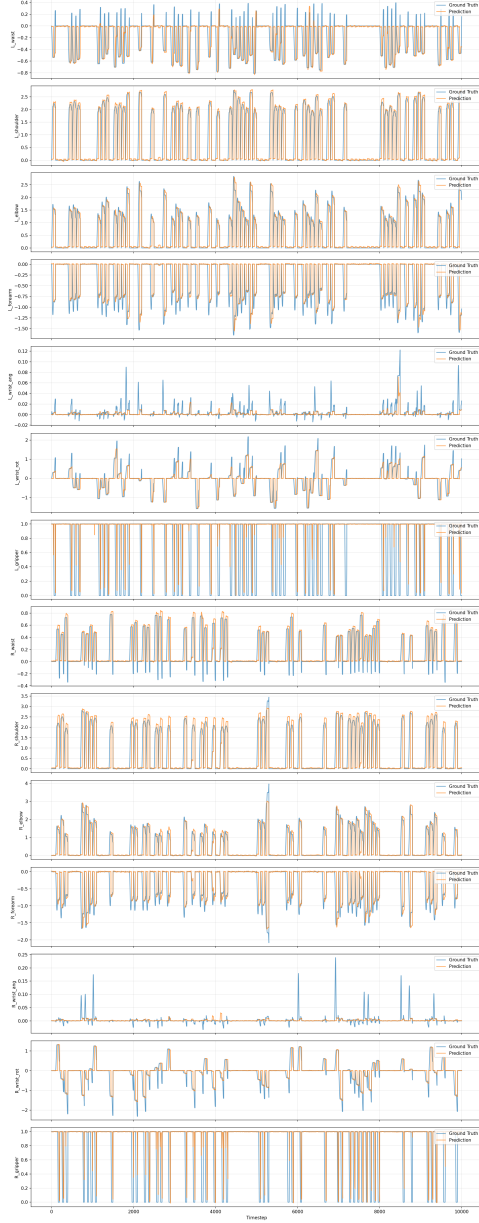


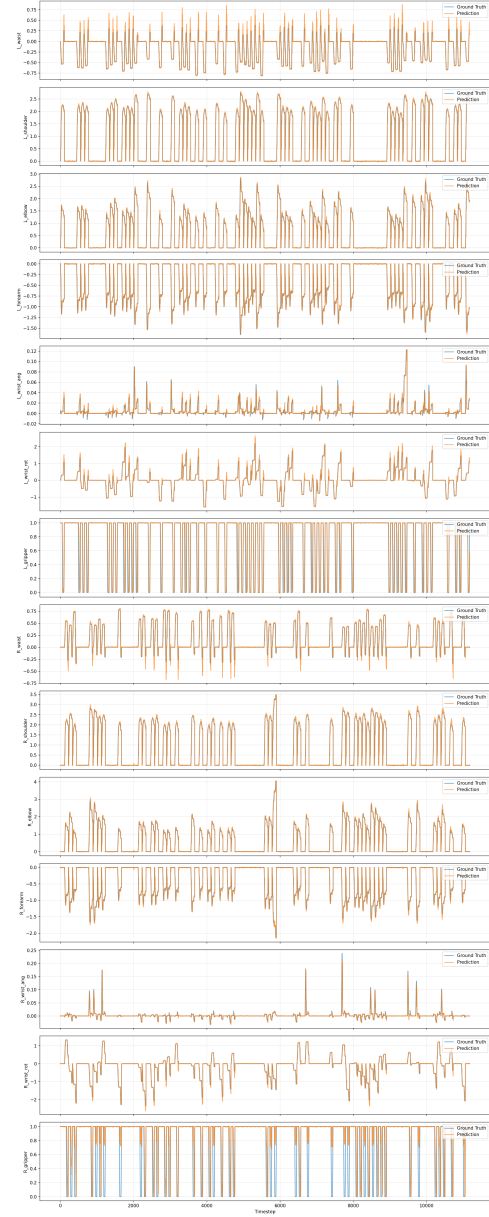
Figure 13: **OOD open-loop action prediction.** Open-loop action-prediction MSE on the demo_randomized OOD dataset for three representative RoboTwin tasks.

Aggregate OOD prediction. Across the three demo_randomized tasks, REACT reduces the average open-loop MSE from 0.272 to 0.048, an 82.4% relative reduction over the $\pi_{0.5}$ baseline. This offline result is consistent with the randomized-simulation gains in §4.2: the staircase-trained action model fits perturbed scenes more accurately after training, rather than only changing its inference-time control schedule.

Figure 14 shows qualitative open-loop predictions for a representative Move Playing Card Away example from the demo_randomized split.



(a) Baseline comparison



(b) REACT comparison

Figure 14: **Qualitative open-loop predictions on Move Playing Card Away.** Two visualizations of the same demo_randomized example show how the predicted action sequence compares with the demonstration trajectory under the baseline and REACT.