RAVEL: ASYNCHRONOUS ROLLING INFERENCE FOR FLOW-BASED VISION–LANGUAGE–ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Flow-based vision–language–action (VLA) models are highly effective for generalist robot manipulation, yet their reliance on computationally expensive VLM encoding and multi-step iterative action generation imposes a significant latency bottleneck. The resulting inference latency makes it difficult for robots to respond quickly, especially in dynamic environments. We address this limitation with **RAVEL** (**R**olling **A**synchronous **V**LA **E**nabling **L**ow-Latency Control), an asynchronous inference framework that addresses the computational bottlenecks of both the VLM backbone and the action expert. To reduce the delay from multi-step action denoising, RAVEL allows near-term actions to be executed after a single denoising step by carrying partially denoised future actions forward in a rolling buffer. To avoid blocking on slow VLM encoding, RAVEL decouples VLM encoding from rolling action generation, allowing the action expert to operate continuously using the latest available VLM context, while a lightweight Fast Observation Pathway (FOP) directly conditions the action expert on current observations. Across simulated and real-world manipulation tasks, RAVEL consistently achieves substantially lower response latency while maintaining the task capability of the underlying VLA, enabling high-frequency and responsive closed-loop control.

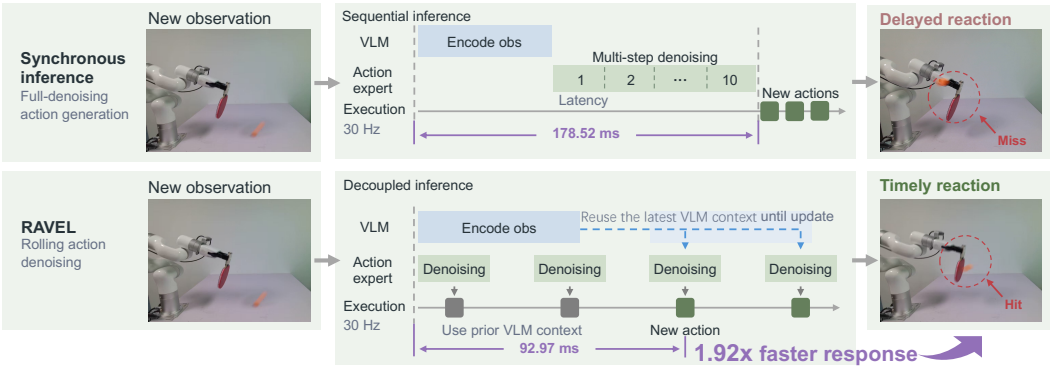

Figure 1: **Responsive control with RAVEL.** Synchronous inference waits for VLM encoding and multi-step action denoising before executing new actions. RAVEL runs VLM context updates in parallel with rolling action denoising and execution, enabling faster responses to new observations.

1 INTRODUCTION

Vision–language–action (VLA) models build on pretrained vision–language models (VLMs) to learn generalist robot policies (Zitkovich et al., 2023; Kim et al., 2024; Team et al., 2025). A common strategy in robot learning is to predict a sequence of future actions, or an *action chunk* (Zhao et al., 2023; Chi et al., 2025). Recent VLAs such as $\pi_{0.5}$ and X-VLA generate these action chunks using flow matching (Intelligence et al., 2025; Zheng et al., 2026; Bjorck et al., 2025). We refer to this family of models as *flow-based VLAs*. Given an observation and the current robot state, the model first encodes the observation with the VLM and then uses an action expert to generate an action chunk through multiple denoising steps. Unfortunately, the sequential VLM encoding and multi-step

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

action generation result in high inference latency (Ma et al., 2025). Under conventional synchronous inference, the robot may pause between action chunks while waiting for inference to finish. Executing multiple actions from a fixed chunk also limits how frequently new observations can influence control.

To address these limitations, asynchronous methods run inference while the robot executes the current action chunk, reducing pauses between action chunks (Shukor et al., 2025; Zhao et al., 2025). However, under naive asynchronous execution, a newly generated chunk may no longer match the robot state when execution begins, leading to discontinuities between chunks. Methods such as RTC improve continuity across chunk transitions (Black et al., 2025a; Tang et al., 2025; Liu et al., 2026; Black et al., 2025b), but each new action update still requires VLM encoding followed by iterative denoising. Recent approaches reduce the iterative computation required before near-term actions become executable (Du et al., 2026; Jiang et al., 2025), using strategies such as horizon-aware noise schedules or rolling denoising (Ruhe et al., 2024; Chen et al., 2024; Lu et al., 2026; Høeg et al., 2024; Jung et al., 2025). However, the latency introduced by VLM encoding remains a bottleneck that blocks continuous action generation in VLAs.

To bridge this gap, we introduce RAVEL, an asynchronous inference framework for flow-based VLAs that addresses the latency of both multi-step action denoising and VLM encoding (Figure 1). To reduce denoising latency, we progressively denoise future actions across control steps using a rolling action buffer. At each control step, the action expert denoises the buffer once, releases the first action, and appends a new noisy action while retaining the remaining actions for further denoising. The next action therefore requires only one final denoising update before execution. Meanwhile, we decouple VLM encoding from action generation to prevent expensive VLM forward passes from blocking this rolling process. VLM context is updated asynchronously, while the action expert continues denoising the buffer using the latest available VLM context. To generate actions for the current control step from delayed VLM context, we condition the action expert on the temporal offset between observation capture and action execution. We further introduce a lightweight Fast Observation Pathway (FOP) that directly supplies current visual and proprioceptive features to the action expert, allowing upcoming actions to respond to recent observations without waiting for VLM encoding to finish.

Our evaluation spans both static and dynamic manipulation tasks in simulation and the real world. Results on LIBERO (Liu et al., 2023) and RoboTwin 2.0 (Chen et al., 2025b) show that adapting the base VLAs to our framework maintains comparable task performance. Furthermore, evaluations on the Dynamic Object Manipulation benchmark (Xie et al., 2026) show improved task success relative to the evaluated baselines. Separate latency measurements show reduced observation-to-command response times. Finally, real-world experiments on table-tennis return and tennis-ball grasping demonstrate our method's ability to respond rapidly and produce smooth motion.

Our main contributions are as follows: (1) we introduce RAVEL, an asynchronous inference framework for flow-based VLAs that decouples expensive VLM encoding from action generation, enabling continuous action generation while using a lightweight Fast Observation Pathway (FOP) to maintain timely perception of current observation; (2) we combine a rolling action buffer with temporal-offset conditioning to enable earlier execution of near-term actions and account for inference delays during action generation; and (3) we demonstrate that RAVEL preserves the manipulation capabilities of pretrained VLAs while substantially reducing response latency, improving success rates in dynamic simulation, and enabling faster responses and smoother motion in real-world manipulation.

2 Related Work

Flow-based VLA and action chunking. Recent VLAs (Li et al., 2024; Shao et al., 2025) often combine a pretrained vision-language model (VLM) with a flow-matching action expert (Lipman et al., 2022), as in $\pi_{0.5}$ and X-VLA (Intelligence et al., 2025; Zheng et al., 2026), among other recent models (Black et al., 2024; Bjorck et al., 2025; Shukor et al., 2025; Wang et al., 2026b; Wu et al., 2026). Since running the full VLA policy is expensive, these methods typically predict a chunk of future actions (Zhao et al., 2023; Liu et al., 2025) at each inference step. However, action chunking still requires VLM encoding followed by multi-step action denoising before the first action becomes available. Our work addresses this latency bottleneck in flow-based VLAs.

Real-time and asynchronous action generation. Asynchronous action-chunking methods overlap inference with execution while improving alignment between predicted actions and their execution times (Black et al., 2025a;b; Tang et al., 2025; Xie et al., 2026; Wang et al., 2026a). To reduce the latency of multi-step denoising, recent methods make executable actions available without fully denoising an entire action chunk from scratch at every update (Høeg et al., 2024; Jung et al., 2025; Chen et al., 2025c;a; Lu et al., 2026; Jiang et al., 2025). Other approaches improve feedback freshness by decoupling slow vision-language processing from fast control or correcting predicted actions using current observations (Zhang et al., 2024; Guo & Liu, 2026; Sendai et al., 2025; Ma et al., 2025). Compared with existing methods, our approach not only reduces multi-step denoising latency through rolling action generation, but also enables timely responses to current observations by decoupling VLM encoding from action generation and incorporating a lightweight Fast Observation Pathway (FOP).

3 METHOD

3.1 OFFSET-AWARE ROLLING FLOW DENOISING

RAVEL maintains a rolling buffer of future actions, where actions scheduled for later execution have higher noise levels (Figure 2). Let K denote the buffer length. At control timestep t_a, the buffer stores K future actions:

$$\mathbf{A}_{t_a} = [a^{\tau_1}_{t_a+1}; \ldots; a^{\tau_k}_{t_a+k}; \ldots; a^{\tau_K}_{t_a+K}], \tag{1}$$

where the k-th slot corresponds to the action executed at $t_a + k$, and $\tau_k = 1 - k/K$ denotes its flow timestep.

To update the actions in the buffer, the action expert uses a representation produced by the VLM backbone, such as layer-wise key–value caches (Black et al., 2024; Intelligence et al., 2025) or VLM features (Bjorck et al., 2025; Shukor et al., 2025; Zheng et al., 2026), which we refer to as the *VLM context*. Given an observation o_{t_o}, proprioceptive state s_{t_o}, and language instruction ℓ, we denote the VLM context as

$$\mathbf{c}_{t_o} = E(o_{t_o}, s_{t_o}, \ell), \tag{2}$$

At control timestep t_a, the action expert uses a VLM context computed from an earlier observation at t_o. To account for this temporal mismatch, we introduce a normalized *temporal offset* $\delta_p = p/K$, where $p = t_a - t_o$, as an additional conditioning signal for the action expert. Given the buffer, VLM context, and temporal offset, one action-expert forward pass advances every slot by $\Delta\tau = 1/K$:

$$a^{\tau_{k-1}}_{t_a+k} = a^{\tau_k}_{t_a+k} + \Delta\tau \left[\mathbf{v}_\theta(\mathbf{A}_{t_a}, \boldsymbol{\tau}, \mathbf{c}_{t_o}, \delta_p)\right]_{[k]}, \qquad k = 1, \ldots, K, \tag{3}$$

where $\mathbf{v}_\theta$ is the flow-based action expert, $[\cdot]_{[k]}$ selects the velocity for slot k, and $\tau_0 = 1$ denotes the clean endpoint.

After the update, the first action reaches τ_0 and is released for execution. The remaining $K-1$ actions are kept in the buffer, and a new action initialized from Gaussian noise, $a^{\tau_K}_{t_a+K+1} \sim \mathcal{N}(0, I)$, is added as the last slot. The buffer therefore again contains K future actions with flow timesteps $\tau_1, \ldots, \tau_K$ at the next control timestep. In this way, each control timestep requires only one action-expert forward pass, while each action is refined over K consecutive updates before execution.

3.2 ASYNCHRONOUS CONTEXT UPDATES AND FAST OBSERVATION PATHWAY

Computing the VLM context is much slower than one action denoising step. We therefore decouple the VLM from the action expert, allowing context updates and action denoising to run asynchronously (Guo & Liu, 2026). At control timestep t_a, the action expert uses the latest available context $\mathbf{c}_{t_o}$ together with its temporal offset δ_p. The same context is reused across several rolling buffer updates until a new one becomes available.

Since the VLM context is computed from an earlier observation, we introduce a Fast Observation Pathway (FOP) to provide the action expert with more recent visual and proprioceptive information. FOP uses a lightweight image encoder together with a projection of the current proprioceptive state:

$$z_{t_a} = F(o_{t_a}, s_{t_a}). \tag{4}$$

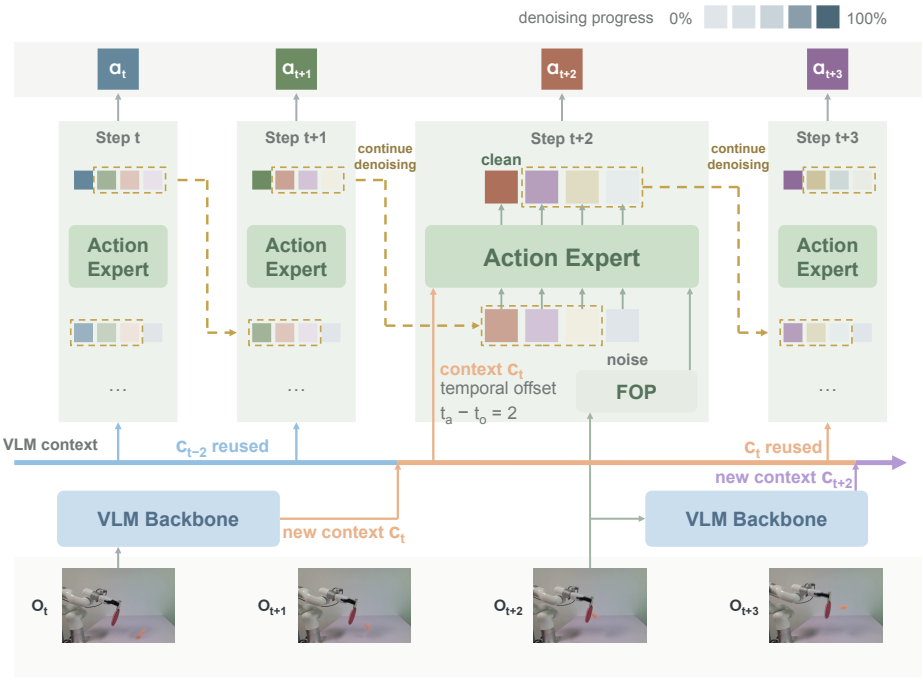

Figure 2: Overview of RAVEL. The VLM begins encoding observation o_t at time t and produces VLM context $\mathbf{c}_t$ between $t+1$ and $t+2$, while the action expert continues using the previously available context. At $t+2$, the expert uses $\mathbf{c}_t$ with a temporal offset of two control steps. Meanwhile, FOP directly conditions the action expert on the current observation o_{t+2}. Conditioned on these inputs, the expert advances the rolling buffer by one denoising step and releases the first action. The remaining actions are retained, and a new noisy action is appended.

The resulting tokens are injected into selected action-expert layers through gated cross-attention adapters. The VLM context encodes semantic information from the instruction and observation at t_o, while FOP incorporates the current observation to refine the action prediction. This allows upcoming actions to respond to new observations without waiting for the next VLM context update.

3.3 Streaming-Aligned Training

Let $\hat{a}_t$ denote the target action at timestep t. Given a training trajectory, we sample an observation timestep t_o and a temporal offset p, with $t_a = t_o + p$. The VLM context $\mathbf{c}_{t_o}$ is computed from the observation at t_o, with the K-step action horizon $\hat{\mathbf{A}}_{t_a} = [\hat{a}_{t_a+1}; \ldots; \hat{a}_{t_a+K}]$ as the training target. By varying p and providing the normalized offset $\delta_p = p/K$ to the action expert, we train it to generate actions for different control timesteps from a reused VLM context.

We construct the training buffer $\mathbf{A}_{t_a}$ using the same flow-timestep profile $\tau_k = 1 - k/K$ as in Section 3.1. For each slot k, we sample $\epsilon_{t_a+k} \sim \mathcal{N}(0, I)$ and construct

$$a_{t_a+k}^{\tau_k} = (1 - \tau_k)\epsilon_{t_a+k} + \tau_k \hat{a}_{t_a+k}. \tag{5}$$

This yields the same position-dependent noise profile as the rolling buffer used during inference, with later actions receiving higher noise levels.

The flow-matching target for each slot k is $\hat{a}_{t_a+k} - \epsilon_{t_a+k}$. We train the action expert to predict these slot-wise velocities conditioned on the training buffer, VLM context, and temporal offset:

$$\mathcal{L}_{\text{train}} = \mathbb{E}_{t_o,\, p,\, k}\left[\left\| [\mathbf{v}_\theta(\mathbf{A}_{t_a}, \boldsymbol{\tau}, \mathbf{c}_{t_o}, \delta_p)]_{[k]} - (\hat{a}_{t_a+k} - \epsilon_{t_a+k}) \right\|_2^2\right]. \tag{6}$$

When FOP is enabled, the FOP features z_{t_a} defined in Eq. (4) are additionally provided to the action expert.

4 EXPERIMENTS

Our evaluation spans both static and dynamic manipulation tasks in simulation and the real world, covering manipulation capability, responsiveness, and inference efficiency. We first use synchronous evaluations on LIBERO (Liu et al., 2023) and RoboTwin 2.0 (Chen et al., 2025b) to test whether RAVEL preserves the manipulation capability of the underlying VLAs. We then evaluate RAVEL on the Dynamic Object Manipulation (DOM) benchmark (Xie et al., 2026), where the environment continues to evolve during inference, to assess responsiveness under continuous motion. Finally, we deploy RAVEL on real-world table-tennis return and tennis-ball grasping to evaluate its effectiveness under real-world timing and execution constraints. We further analyze latency and ablate the key components of RAVEL.

4.1 SYNCHRONOUS SIMULATION

Experimental Setup. We evaluate whether adapting pretrained flow-based VLAs to RAVEL preserves their manipulation capability. We test $\pi_{0.5}$ and X-VLA on LIBERO (Liu et al., 2023) and RoboTwin 2.0 (Chen et al., 2025b), covering single-arm tabletop and bimanual manipulation tasks. Both benchmarks use synchronous simulation: the environment pauses during policy inference and resumes only after the predicted actions are returned. Inference latency therefore does not affect the rollout, allowing us to isolate manipulation capability from the responsiveness benefits of asynchronous execution.

For each evaluation setting, we compare the baseline with a RAVEL variant initialized from the same publicly released benchmark-specific checkpoint. We use RAVEL without FOP to assess whether it preserves the baseline performance under the same generation budget. RAVEL is trained with streaming-aligned training (Section 3.3) while keeping the vision–language backbone frozen. Both methods retain the native action representation and horizon and use the same generation budget of one VLM prefill and ten action-expert forward passes. The baseline performs ten full-horizon denoising steps, whereas RAVEL performs ten rolling updates. All methods use the same task configurations, environment seeds, and language instructions.

We additionally measure first-action latency from receiving an observation to making the first predicted action available for execution. Measurements are conducted on one RTX 4090, and we report the median latency. Further training, evaluation, and buffer-handling details are provided in Appendix A.1.

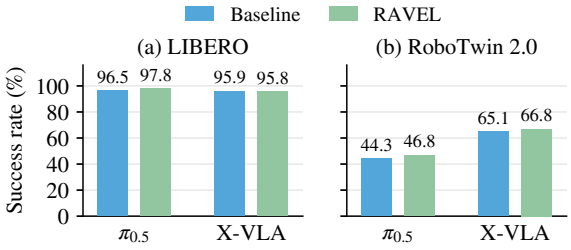

Figure 3: Synchronous evaluation of the baseline and RAVEL under matched generation budgets. Results aggregate 2,000 LIBERO trials and 5,000 RoboTwin 2.0 trials.

Table 1: Median first-action latency for X-VLA and $\pi_{0.5}$ using PyTorch inference on one RTX 4090, measured from observation receipt to the first predicted action becoming available for execution.

Model	Full denoising (ms)↓	Rolling inference (ms)↓	Speedup
X-VLA	142.58	**54.97**	2.59×
$\pi_{0.5}$	272.86	**93.45**	2.92×

Results. As shown in Figure 3, RAVEL maintains the task performance of both pretrained VLAs across the two benchmarks, with comparable or slightly improved success in all settings. The small gains may come from streaming-aligned training and the temporal consistency enabled by progressive action refinement. These results show that rolling inference can be introduced without degrading the original manipulation capability.

RAVEL also substantially reduces first-action latency. As shown in Table 1, it achieves a 2.59–2.92× speedup on one RTX 4090. This speedup is expected, as rolling inference requires only a single additional action-expert forward pass to release each new action.

4.2 ASYNCHRONOUS SIMULATION

Experimental Setup. We evaluate on the Dynamic Object Manipulation (DOM) benchmark introduced with DynamicVLA (Xie et al., 2026), which uses moving-object grasp-and-place tasks to assess a model's ability to interact with moving objects, perceive dynamic scenes, and generalize across variations in appearance and motion. Unlike the synchronous benchmarks above, the simulator continues to advance during policy inference. Objects may move or change direction before the predicted actions are executed. This setting exposes the effect of inference latency on task execution. We use DOM to evaluate the performance of RAVEL in dynamic environments.

All methods are initialized from the released DynamicVLA checkpoint. We evaluate RAVEL with and without the Fast Observation Pathway (FOP), and compare against synchronous DynamicVLA, Continuous Inference with Latent-aware Action Streaming (CI & LAAS) (Xie et al., 2026), VLASH (Tang et al., 2025), and FASTER (Lu et al., 2026). Each method is evaluated across 9 dimensions, with 10 scenes per dimension and 20 trials per scene, yielding 1,800 episodes in total. We report the overall success rate and the mean task completion time over episodes successful for all six methods, where completion time reflects environment execution time rather than model inference latency. All experiments are run locally on two NVIDIA RTX 4090 GPUs, with simulation and policy inference running on separate GPUs. Additional implementation details and measured inference latencies are provided in Appendix A.2 and Table 5.

Table 2: Success rates (%) and mean completion time over common-success episodes on DOM. All results are obtained from local evaluations using two NVIDIA RTX 4090 GPUs, with simulation and policy inference on separate GPUs. **Bold** and underlined values indicate the best and second-best results, respectively.

	Interaction			Perception			Generalization			Average	
Method	CR	DA	LS	VU	SR	MP	VG	MG	DR	SR ↑	Time ↓
DynamicVLA	25.5	22.0	13.5	38.5	20.0	23.0	22.0	53.5	16.5	26.06	7.85
+ CI & LAAS	46.0	23.5	28.5	38.5	39.0	23.0	39.5	58.5	23.0	35.50	6.58
+ VLASH	18.5	16.0	9.5	13.5	13.5	17.0	23.5	40.0	17.5	18.78	7.13
+ FASTER	37.0	19.0	14.5	20.5	22.5	24.0	19.0	45.0	12.5	23.78	7.11
+ RAVEL	49.0	29.0	38.0	48.0	46.0	38.5	**50.0**	70.5	**27.5**	44.06	**6.04**
+ RAVEL w/ FOP	**62.5**	**32.5**	**40.5**	**49.0**	**49.0**	**43.0**	49.0	**76.5**	20.5	**46.94**	6.19

CR: Closed-loop Reactivity; DA: Dynamic Adaptation; LS: Long-horizon Sequencing; VU: Visual Understanding; SR (Perception): Spatial Reasoning; MP: Motion Perception; VG: Visual Generalization; MG: Motion Generalization; DR: Disturbance Robustness; SR (Average): Success Rate.

Time: Mean environment steps divided by 25 Hz, in seconds, over the same 36 scene–trial pairs successful for all six methods.

Results. Table 2 summarizes the results on DOM. RAVEL achieves an overall success rate of 44.06%, outperforming CI & LAAS by 8.56 percentage points and improving performance across all nine evaluation dimensions. Adding FOP further increases the overall success rate to 46.94%. In particular, RAVEL with FOP reaches 62.5% in Closed-loop Reactivity and 43.0% in Motion Perception, highlighting the benefit of incorporating recent observations for tasks that require timely responses to changing object states. RAVEL also completes tasks in fewer execution steps, achieving the shortest mean completion time of 6.04 s, a 23.0% reduction relative to DynamicVLA. Together with the higher overall success rate, this result suggests that RAVEL's latency reduction translates into improved task execution efficiency in dynamic manipulation. Overall, these results show that reducing action-generation latency substantially improves performance in dynamic manipulation, while incorporating recent visual feedback through FOP further improves adaptation to changing object states. We also observe that several existing methods do not improve over the DynamicVLA baseline under our local evaluation setting. Further implementation details and discussion are provided in Appendix A.2.

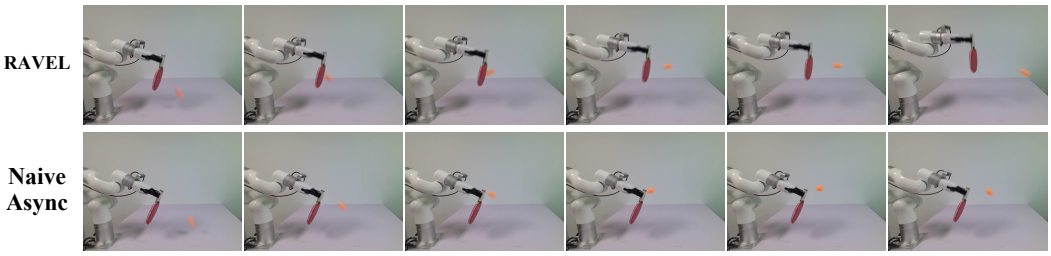

Figure 4: Real-world table-tennis return sequences for RAVEL (top) and Naive Async (bottom). The faster response of RAVEL enables a timely paddle swing that contacts the incoming ball and returns it farther. Slow inference in Naive Async delays the swing, so the ball hits the paddle handle before the robot starts to move. Frames progress from left to right within each rollout; columns do not indicate synchronized timestamps across methods.

4.3 Real-World Evaluation

Experimental Setup. We evaluate RAVEL on two real-world manipulation tasks using an xArm7 robot: table-tennis return and tennis-ball grasping, with 20 trials per method for each task. All real-world policy inference runs on an NVIDIA RTX 5090 GPU.

For table-tennis return, the robot must intercept and return an incoming ball. All policies are trained on the same 237 teleoperated trajectories. We compare RAVEL with synchronous inference (Sync), naive asynchronous inference (Naive Async), RTC (Black et al., 2025a), VLASH (Tang et al., 2025), and FASTER (Lu et al., 2026). The robot exhibits a noticeable execution delay (Appendix B.4). Compared with the other methods, RAVEL can compensate for this delay by adding a temporal-offset bias. This bias shifts the prediction to the time when the robot is expected to reach the target state. We use $\delta_0 = 0.3$ and ablate it in Section 4.4. To evaluate task performance and response latency, we measure return success rate and observation age. The observation age of an action is the elapsed time from receiving its source image to dispatching the action to the robot. We report the median (P50) and 95th percentile (P95) over dispatched actions. Example rollouts are shown in Figure 4.

For tennis-ball grasping, a trial succeeds when the robot grasps the ball and successfully places it in the basket. We compare RAVEL with Sync, Naive Async, and RTC. In addition to grasping success rate, we evaluate motion quality to examine whether the different inference schemes affect the smoothness and stability of robot execution. Specifically, we report the high-frequency residual power E_{HF}, inspired by the spectral analysis of Sojib et al. (2026), to quantify high-frequency oscillations in the TCP trajectory, and TCP jerk RMS to measure abrupt changes in motion. Lower E_{HF} indicates less high-frequency trajectory variation, while lower jerk RMS indicates smoother changes in acceleration. Together, these metrics allow us to evaluate not only whether the robot completes the grasp, but also how smoothly and stably the motion is executed. Detailed metric definitions and measurement protocols are provided in Appendix B.

Results. As shown in Table 3, RAVEL achieves a 90% return success rate on table tennis, compared with 25% for both FASTER and VLASH. Because policy inference takes time and the xArm introduces a non-negligible delay between command dispatch and physical execution, conventional inference methods cannot react to the incoming ball in time. RAVEL reduces policy response latency through rolling inference and uses temporal-offset conditioning to compensate for execution delay, helping align the predicted actions with the ball state at their actual execution time.

This advantage is also reflected in response speed. RAVEL achieves a median observation age of 121.76 ms and a 95th percentile of 132.20 ms, both the lowest reported values in Table 3. Sync uses increasingly stale observations as it executes each chunk. RTC adds update latency through vector–Jacobian products in its inference-time guidance. FASTER reduces the action-generation cost, but VLM encoding and action inference remain sequential. In contrast, RAVEL runs VLM context encoding and rolling action updates in separate workers, allowing action inference to continue without waiting for each VLM update and maintaining consistently fresher observations.

Table 3: **Real-world evaluation.** Success rates are computed over 20 trials per method for each task. Observation age measures visual-input freshness at action dispatch. P50 and P95 denote the median and 95th percentile of observation age, respectively. E_{HF} and TCP jerk RMS characterize trajectory oscillations and motion smoothness, respectively. Bold indicates the best result for each metric.

Method	**Table-tennis return**		**Tennis-ball grasping**		
	Success (%)↑	Obs. age (ms)↓ P50 / P95	Success (%)↑	$E_{\mathrm{HF}}\downarrow$ (mm^2)	TCP jerk↓ ($\mathrm{mm/s}^3$)
Sync	0	327.05 / 482.54	40	14.45	1983
Naive Async	0	221.88 / 287.35	**45**	19.43	3208
RTC	0	418.88 / 563.39	40	101.70	8398
VLASH	25	283.53 / 441.56	–	–	–
FASTER	25	149.63 / 212.44	–	–	–
RAVEL	**90**	**121.76 / 132.20**	**45**	**7.01**	**1800**

On tennis-ball grasping, RAVEL achieves comparable task success while exhibiting the lowest trajectory variation across both E_{HF} and TCP jerk RMS. Together, these metrics indicate fewer high-frequency oscillations and fewer abrupt motion changes during execution. The chunk-based baselines update actions less frequently, so latency can produce larger discontinuities when switching to a newly generated chunk. RAVEL instead streams actions through rolling inference and continuously refines future actions across control cycles, resulting in smoother transitions and more stable execution. We also find that successful grasps occur at similar ball locations across methods. This is likely because the methods share similar manipulation capabilities, while RAVEL mainly improves how the actions are executed.

4.4 Analysis and Ablation Studies

Ablation of Training-Time Alignment Components. We ablate the training components of RAVEL using $\pi_{0.5}$ on LIBERO and RoboTwin 2.0, with the rolling inference procedure fixed across all four variants. Figure 5(a) shows that directly applying rolling inference to the pretrained checkpoint performs poorly, and matching the training noise profile to the rolling buffer is insufficient to recover performance. Training on action horizons shifted relative to the observation improves success rates to 78.00% on LIBERO and 31.54% on RoboTwin 2.0. However, without temporal-offset conditioning, the model is not told which future horizon to predict from a given observation. Providing this offset further improves success rates to 97.85% and 46.82%, respectively. These results suggest that adapting a pretrained policy to rolling inference requires both training on shifted action horizons and conditioning on their temporal offsets.

Temporal Alignment under Inference Latency. Inference latency introduces a delay between observation capture and action execution. Temporal-offset conditioning informs the action expert of this delay, allowing it to predict actions for the intended execution time. We examine this temporal alignment using controlled observation delays on LIBERO and an offset ablation under asynchronous execution on DOM.

We first simulate inference latency in a synchronous environment using delayed observations. On LIBERO with X-VLA, we vary the observation delay from 0 to 15 control steps. Across these delays, RAVEL maintains 95.00–97.25% success, whereas the baseline decreases from 98.25% to 92.50% (Figure 5(b)). This indicates that temporal-offset conditioning improves robustness to the temporal misalignment caused by inference latency.

We further evaluate the offset under asynchronous execution on DOM, where continuously changing object states make the policy more sensitive to temporal misalignment. Because VLM encoding takes multiple control steps, a newly available context is already stale. RAVEL uses the actual offset $p = t_a - t_o$, while the comparison variant resets the offset to zero whenever a new context becomes available, thereby ignoring the VLM computation delay. With the checkpoint and non-blocking runtime fixed, success drops from 44.89% to 16.67% over 450 paired episodes. This result highlights

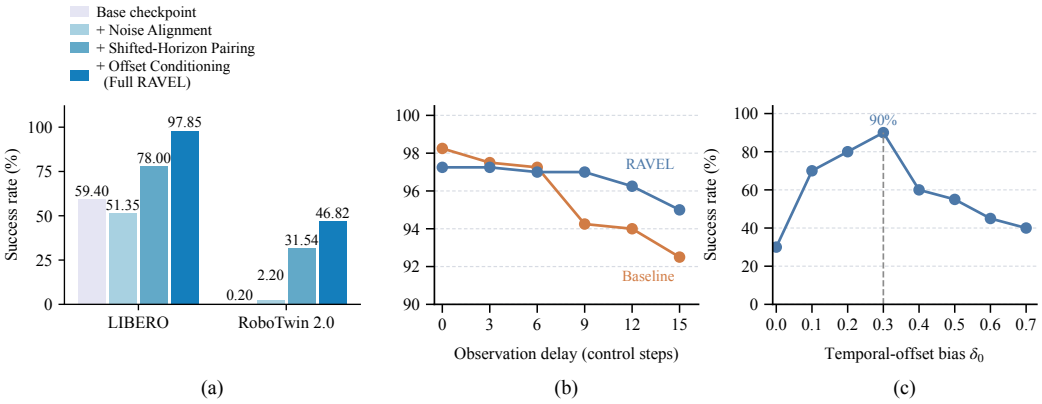

Figure 5: Ablation of temporal alignment components. (a) Training-time alignment for $\pi_{0.5}$ on LIBERO and RoboTwin 2.0: the base checkpoint under rolling inference, noise alignment only, shifted-horizon training without temporal-offset conditioning, and full RAVEL. All variants use 2,000 episodes on LIBERO and 5,000 episodes on RoboTwin. (b) Robustness to delayed observations on LIBERO with X-VLA under synchronous rollouts; each point uses 400 trials. (c) Temporal-offset bias for execution-delay compensation in real-world table-tennis return with a fixed policy checkpoint. Success peaks at 90% with $\delta_0 = 0.3$.

the importance of including VLM computation time in the temporal offset to align action predictions with their intended execution times.

Effect of Temporal-Offset Bias. We evaluate how the temporal-offset bias adapts RAVEL to the large execution lag of the real robot. The base offset accounts for observation age at command dispatch, while the added bias advances the predicted target to compensate for the subsequent physical tracking lag. This adjustment uses the policy's existing temporal-offset conditioning while keeping the policy weights and inference procedure fixed.

Figure 5(c) examines the effect of temporal-offset bias on the real-world table-tennis task by sweeping eight values of δ_0 with the same policy checkpoint. When δ_0 is small, the execution delay is under-compensated, causing the robot to swing too late. Increasing δ_0 better aligns the predicted action with its actual execution time, and success peaks at 90% when $\delta_0 = 0.3$. Beyond this point, a larger offset shifts the predicted action too far forward in time, causing the robot to swing too early. It also requires the policy to predict farther into the future, making accurate action prediction more challenging. Consequently, success declines at larger offsets, reaching 40% at $\delta_0 = 0.7$.

5 Limitations and Future Work

RAVEL provides a rolling asynchronous inference framework for improving the responsiveness of flow-based VLA models. It currently requires streaming-aligned training, while temporal-offset conditioning and FOP involve architecture-specific modifications to the action expert because existing VLA models do not share a unified action-expert design. The framework is presently formulated for flow-based VLAs, and its action-generation latency is bounded below by the time required for a single action-expert forward pass. FOP improves observation freshness by providing the action expert with current observations, but the subsequent delay between action prediction and physical execution remains in real-world systems, potentially reducing its benefits. Future work could explore more general and efficient streaming inference across VLA architectures, improving compatibility with diverse action-expert designs or lowering the computational cost of action generation. Future work could also investigate tighter integration between streaming action generation, robot motion planning, and execution feedback, accounting for physical delays and the robot's evolving state for more responsive closed-loop control.

AI USE STATEMENT

We used generative AI tools to assist with code implementation, literature search and review, manuscript drafting and editing, figure preparation, and analysis and interpretation of experimental results. These tools were not used to develop the proposed method or to design or provide feedback on the research methodology or experiments.

We have reviewed all AI-assisted work. The authors take responsibility for the final content of this paper, including text, claims, and artifacts produced with the assistance of generative AI.

REFERENCES

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A unified latent action world model, 2025. URL https://arxiv.org/abs/2512.13030.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

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

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 33383–33407. Curran Associates, Inc., 2025a. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/300ccb2187dedd4edcc07f7e76d8e553-Paper-Conference.pdf.

Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025b.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

Haojun Chen, Minghao Liu, Chengdong Ma, Xiaojian Ma, Zailin Ma, Huimin Wu, Yuanpei Chen, Yifan Zhong, Mingzhi Wang, Qing Li, et al. Falcon: Fast visuomotor policies via partial denoising. *arXiv preprint arXiv:2503.00339*, 2025a.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu 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*, 2025b.

Zhuoqun Chen, Xiu Yuan, Tongzhou Mu, and Hao Su. Responsive noise-relaying diffusion policy: Responsive and efficient visuomotor control. *arXiv preprint arXiv:2502.12724*, 2025c.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Fan Du, Feng Yan, Jianxiong Wu, Xinrun Xu, Weiye Zhang, Weinong Wang, Yu Guo, Bin Qian, Zhihai He, Fei Wang, et al. Cf-vla: Efficient coarse-to-fine action generation for vision-language-action policies. *arXiv preprint arXiv:2604.24622*, 2026.

Yuanchun Guo and Bingyan Liu. Reflex: Real-time vla control through streaming inference. *arXiv preprint arXiv:2607.14695*, 2026.

Sigmund H Høeg, Yilun Du, and Olav Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.

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

Sunshine Jiang, Xiaolin Fang, Nicholas Roy, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Siddharth Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories. *arXiv preprint arXiv:2505.21851*, 2025.

Chanhyuk Jung, Dasom Ahn, Sangwon Kim, In-su Jang, Kwang-Ju Kim, Sungkeun Yoo, and Byoung Chul Ko. Rolling diffusion policy for robotic action prediction: Enhancing efficiency and temporal awareness. In *ICRA 2025 Workshop on Foundation Models and Neuro-Symbolic AI for Robotics*, 2025.

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

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

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

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/49f80e4d2471ad4f2edf4f5f1ab62339-Paper-Conference.pdf.

Yufeng Liu, Hang Yu, Juntu Zhao, Bocheng Li, Di Zhang, Mingzhu Li, Wenxuan Wu, Yingdong Hu, Junyuan Xie, Junliang Guo, et al. Learning native continuation for action chunking flow policies. *arXiv preprint arXiv:2602.12978*, 2026.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11966–11976. IEEE, 2022.

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

Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.

David Ruhe, Jonathan Heek, Tim Salimans, and Emiel Hoogeboom. Rolling diffusion models. *arXiv preprint arXiv:2402.09470*, 2024.

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

Rui Shao, Wei Li, Lingsen Zhang, Renshan Zhang, Zhiyang Liu, Ran Chen, and Liqiang Nie. Large vlm-based vision-language-action models for robotic manipulation: A survey. *arXiv preprint arXiv:2508.13073*, 2025.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. Smolvla: A vision-language-action model for affordable and efficient robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Noushad Sojib, Sajay Arthanat, and Momotaz Begum. An efficient metric for data quality measurement in imitation learning. *arXiv preprint arXiv:2605.01544*, 2026.

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

Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

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

Qiuyue Wang, Mingsheng Li, Jian Guan, Jinhui Ye, Sicheng Xie, Yitao Liu, Junhao Chen, Zhixuan Liang, Jie Zhang, Xintong Hu, et al. Qwen-vla: Unifying vision-language-action modeling across tasks, environments, and robot embodiments. *arXiv preprint arXiv:2605.30280*, 2026b.

Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators, 2024. URL https://arxiv.org/abs/2309.13037.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, et al. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.

Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. Hirt: Enhancing robotic control with hierarchical robot transformers. *arXiv preprint arXiv:2410.05273*, 2024.

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

Yongsheng Zhao, Lei Zhao, Baoping Cheng, Gongxin Yao, Xuanzhang Wen, and Han Gao. Vla-rail: A real-time asynchronous inference linker for vla models and robots. *arXiv preprint arXiv:2512.24673*, 2025.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 60580–60606, 2026.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The*

7th Conference on Robot Learning, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A Simulation Implementation Details

A.1 LIBERO and RoboTwin 2.0

We use the official releases for X-VLA on LIBERO and RoboTwin 2.0, and for $\pi_{0.5}$ on LIBERO; for $\pi_{0.5}$ on RoboTwin 2.0, we use the checkpoint released by Motus Robotics (Bi et al., 2025). For $\pi_{0.5}$ on LIBERO, we convert OpenPI's released JAX `pi05_libero` weights to PyTorch using its checkpoint-conversion script. All four settings use PyTorch inference.

All four configurations use $K = 10$ buffer slots. Each slot contains h consecutive low-level actions, giving a total horizon of $H = Kh$; a rolling update releases one slot. We do not perform any additional initialization. The first target action is predicted directly with a single denoising update. As rolling updates proceed, subsequent actions undergo progressively more denoising steps before execution, until each action receives K updates in steady state.

The temporal offset $\delta_p = p/K$ encodes the age of the VLM context. For $\pi_{0.5}$, we encode δ_p with an MLP and add the resulting embedding to the flow-timestep embedding, jointly conditioning the action expert on context age and denoising progress. For X-VLA, we concatenate δ_p with the noisy action, proprioception, and flow-timestep embedding, and feed the combined features to the action encoder. All action slots share the same offset and use their individual flow timesteps.

We fine-tune the released checkpoints described above to obtain the four RAVEL models. For $\pi_{0.5}$, we freeze the vision–language backbone and train the action expert, action input/output projections, flow-timestep MLP, and added temporal-offset MLP. For X-VLA, we freeze the vision–language backbone and the soft prompts, and train the action Transformer and action input/output heads, including the input mapping extended to accept the temporal offset. Table 4 summarizes the training and evaluation configurations for all four settings.

Table 4: Training and evaluation configurations for synchronous evaluation. Batch size is global; steps denote fine-tuning steps for RAVEL. EE denotes end-effector pose.

Benchmark	Backbone	Action representation	H	h	Steps	Batch
LIBERO	$\pi_{0.5}$	Relative EE pose	10	1	5K	128
LIBERO	X-VLA	Absolute EE pose	30	3	5K	128
RoboTwin 2.0	$\pi_{0.5}$	Relative joint position	30	3	10K	128
RoboTwin 2.0	X-VLA	Absolute EE pose	30	3	10K	128

During synchronous evaluation, each policy query uses one VLM prefill and ten action-expert forward passes. The baseline performs ten full-horizon denoising steps, while RAVEL performs ten rolling updates to produce H actions.

For $\pi_{0.5}$ on RoboTwin 2.0, the model predicts relative joint actions, expressed as offsets from the current robot state. Rolling denoising cannot directly reuse these relative actions across observations because their reference state changes as the robot moves. Interpreting a stored relative action with respect to a new state would change its intended joint target. We therefore store normalized absolute joint targets in the clean buffer and convert them to relative actions with respect to the latest state before each action-expert update. The predicted clean relative actions are converted back to absolute targets before being written to the buffer. The paired noise remains in model coordinates. This conversion preserves the intended targets across rolling updates.

A.2 Dynamic Object Manipulation

DynamicVLA inference latency. Table 5 reports local RTX 4090 inference latency. With the same model configuration as the official release, our median full-model inference latency is 370.7 ms,

compared with the 226 ms reported on RTX A6000 by the original paper (Xie et al., 2026). Because DOM involves dynamic manipulation, slower inference in our local setup likely contributes to the gap between our reproduced success rates and those reported in the original paper.

Table 5: DynamicVLA inference latency measured locally on an NVIDIA RTX 4090 GPU under our configuration. All values are in milliseconds; P50 and P95 denote the median and 95th percentile.

Component / inference path	P50 (ms)	P95 (ms)
VLM encoding	63.47	84.34
Action expert	31.75	48.75
Action expert with FOP	41.4	58.6
Full model	370.7	433.5

RAVEL. We extend DynamicVLA's action expert with temporal-offset conditioning. The offset is concatenated with the action and flow-timestep embeddings before the action-time MLP, and its sinusoidal embedding conditions a gated residual adapter on the resulting action tokens. FOP comprises a ConvNeXt-Base image encoder (Liu et al., 2022), visual/state projections, camera/state embeddings, and gated cross-attention adapters that inject current observations into the action expert.

We freeze the VLM and train the action expert and temporal adapter for 100K steps with global batch size 15, learning rates 10^{-6} and 3×10^{-5}, respectively, and progressive noise. FOP training starts from this checkpoint. With the policy and image encoder frozen, we train its projections, embeddings, and adapters for 50K steps with global batch size 120 and learning rate 10^{-4}.

DynamicVLA predicts relative actions: end-effector pose targets expressed relative to the reference robot state, with absolute gripper commands. To reuse the buffer across changing reference states, we apply the same coordinate conversion as for $\pi_{0.5}$ on RoboTwin 2.0 (Appendix A.1).

VLASH. Following VLASH's training framework, we implement adaptation training for DynamicVLA in our local codebase, fine-tuning the VLM backbone and action expert. Training uses global batch size 128, with learning rates of 5×10^{-5} for the action expert and 10^{-6} for the VLM backbone. Training pairs the original images and language with a future-state anchor and a demonstration action chunk shifted by a sampled delay of up to ten control steps. The anchor is the preceding demonstration action's pose, or the observed state at zero delay. We express the shifted pose targets relative to this anchor and optimize the flow-matching loss over the 20-action chunk, keeping gripper targets absolute. We select the best-performing checkpoint among 10K, 20K, and 30K steps, which is the 20K-step checkpoint. At inference, VLASH uses ten denoising steps per 20-action chunk and requests a new chunk when ten actions remain, giving a 400 ms overlap window. It estimates the handoff state from the predecessor's absolute action target and decodes new relative actions against that state, with action quantization ratio one. Predicting relative actions may make this adaptation sensitive to errors in the estimated handoff state: during training, both the conditioning state and the relative action targets depend on the chosen reference state. At inference, the estimated state conditions the prediction and anchors its decoding, but may differ substantially from the state actually reached at execution. This mismatch may contribute to the poor adaptation results.

FASTER. Following FASTER's training framework, we implement adaptation training for DynamicVLA in our local codebase, fine-tuning the VLM backbone and action expert. Training uses global batch size 128, with learning rates of 2.5×10^{-5} for the action expert and 10^{-6} for the VLM backbone. We mix constant and horizon-aware denoising schedules with probability 0.5, using $\alpha = 0.6$, $u_0 = 0.9$, and a maximum training delay of ten steps, and evaluate the 30K-step checkpoint. At inference, FASTER uses ten denoising steps over a 20-action horizon, an assumed delay of four control steps (160 ms), and an execution horizon of 12 actions (480 ms). The same α and u_0 are retained, and inference runs asynchronously without compilation. Using the baseline constant-schedule, direct full-chunk inference with the same checkpoint yields 28.56% success under time compensation, comparable to the baseline DynamicVLA model and higher than the 23.78% achieved with FASTER inference. This indicates that the checkpoint retains task capability. We qualitatively observed abrupt changes in FASTER's output actions. This may be because using fewer

denoising steps for near-term actions is less well suited to predicting relative actions, leading to these discontinuities.

B Real-World Evaluation Details

B.1 Table-Tennis Return

We evaluate each method over 20 trials at a control frequency of 30 Hz. A trial succeeds when the robot hits the incoming ball and returns it over a predefined distance.

Hardware and workspace. We use a seven-degree-of-freedom UFACTORY xArm 7 with a paddle for table-tennis return (Figure 6). To promote fair comparisons across methods, we use a ball launcher to serve the incoming table-tennis balls. The setup provides three RGB views: one external Azure Kinect camera at 1280×720 and two RealSense cameras, mounted at the wrist and outside the workspace, at 640×480. Images are captured at 30 Hz, and the policies use joint-state feedback to predict joint-position targets at a nominal control rate of 30 Hz. Real-world inference runs on a workstation with an NVIDIA GeForce RTX 5090 GPU, using Ubuntu 22.04, PyTorch 2.7.1, and CUDA 12.8.

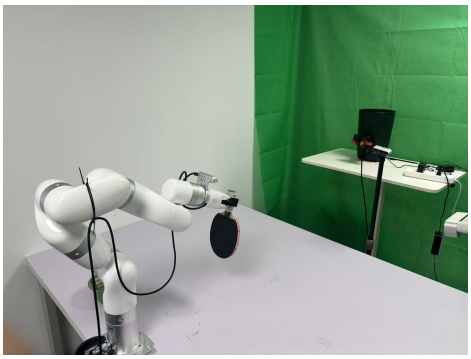

Figure 6: Real-world table-tennis setup.

Data and training. Using GELLO (Wu et al., 2024), we collect 241 teleoperated trajectories at 30 Hz (11.6 minutes), using 237 for training and four for validation. Each demonstration contains three camera views and joint-space targets with a prediction horizon of 10 steps. We construct the action labels by shifting the measured joint-state sequence forward by one timestep: the target paired with a state at time t is the measured joint state at $t + 1$, rather than the command recorded at t. Thus, the policy learns to predict future joint states, which are sent to the robot as joint-position targets during deployment. To obtain the base policy, we fine-tune the pretrained $\pi_{0.5}$ model using rank-64 LoRA adapters in the action expert, together with trainable action and flow-timestep projections, and select the checkpoint at 10,000 steps. Sync, Naive Async, and RTC use this base policy directly. VLASH, FASTER, and RAVEL train for a further 10K steps from the same base policy on the same data (Table 6).

Table 6: Training configurations for table-tennis return.

Setting	Task policy	VLASH	FASTER	RAVEL
Initialization	$\pi_{0.5}$		Task policy (10K)	
Training scope	LoRA	Full model	Full model	Action expert
Trainable parameters	29.89M	3.35B	3.35B	431.15M
Evaluation checkpoint step	10K	10K	10K	10K
Batch size	128	128	128	64
Learning rate	5×10^{-5}	5×10^{-5}	2.5×10^{-5}	5×10^{-5}
LR decay steps	20K	10K	10K	10K
Warmup steps	500	500	1,000	250

All configurations use AdamW, weight decay 10^{-10}, and gradient clipping norm 1.0. Batch size is global. Parameter counts cover trainable modules used in action prediction, including action and temporal-conditioning projections.

Deployment. Sync executes a 10-action chunk before requesting the next prediction, pausing execution during inference. Naive Async asynchronously requests 10-action chunks whenever the inference server is idle. RTC uses an execution horizon of 10 actions and an assumed inference delay

of four control steps. VLASH uses an execution horizon of 10 actions with four inference-overlap steps and an action quantization ratio of one. FASTER uses an execution horizon of four actions, an inference delay of three steps, $\alpha = 0.6$, and $u_0 = 0.9$. RAVEL uses $K = 10$ buffer slots, releases one action per rolling update, and sets $\delta_0 = 0.3$.

B.2 Tennis-Ball Grasping

We compare Sync, Naive Async, RTC, and RAVEL over 20 trials per method, with a 30-s timeout. Success requires grasping the ball and placing it in the basket.

Data and training. We collect 66 teleoperated demonstrations at 30 Hz (10.4 minutes), using 64 for training and two for validation. The policies use two camera views and predict joint targets and gripper commands. The task policy is fine-tuned from pretrained $\pi_{0.5}$. Sync, Naive Async, and RTC use its 10K-step checkpoint directly; RAVEL starts from the same checkpoint and trains for a further 5K steps. Table 7 summarizes the training configurations.

Table 7: Training configurations for tennis-ball grasping.

Setting	Task policy	RAVEL
Initialization	$\pi_{0.5}$	Task policy (10K)
Training scope	LoRA	Action expert
Trainable parameters	29.89M	431.15M
Evaluation checkpoint step	10K	5K
Batch size	128	128
Learning rate	5×10^{-5}	10^{-5}
LR decay steps	10K	5K
Warmup steps	500	250

All configurations use AdamW, weight decay 10^{-10}, and gradient clipping norm 1.0. Batch size is global.

B.3 Metric Definitions

Observation age. Observation age measures the time from receiving an action's source images to dispatching the action:

$$A_k = t_k^{\text{dispatch}} - t_{s(k)}^{\text{obs}}, \tag{7}$$

where $s(k)$ identifies the observation used to generate action a_k. We timestamp each observation when the last of its three camera images is received, and each action when the robot SDK call begins, using the same host clock. For RAVEL, we measure the age of the images used to compute the current VLM context. For Naive Async, each dispatched action is associated with the observation used to generate its source action chunk, and its age is computed using the same definition above. A reused action retains its original source-observation timestamp.

High-frequency residual power. Inspired by the end-effector spectral-power analysis of Sojib et al. (2026), we define E_{HF} using linear detrending, length normalization, and the 1–15 Hz frequency band. We resample TCP positions at 30 Hz and divide each trajectory into non-overlapping 3-s windows. Within each window, we subtract a fitted linear trend from each position coordinate and sum the remaining spectral power at frequencies of 1 Hz and above:

$$E_{\text{HF}} = \sum_{d \in \{x,y,z\}} \sum_{1\,\text{Hz} \le f \le 15\,\text{Hz}} \frac{c_f}{N^2} \left| \mathcal{F}[x_d(t) - (a_d t + b_d)]\,(f) \right|^2, \tag{8}$$

where $x_d(t)$ is the TCP position along axis d, $a_d t + b_d$ is its fitted trend, and N is the number of samples in the window. $\mathcal{F}$ denotes the unnormalized discrete Fourier transform. Within the selected band, $c_f = 2$ for $1 \le f < 15$ Hz and $c_f = 1$ at the 15 Hz Nyquist frequency. We take the median window score for each trajectory.

TCP jerk. We resample measured Cartesian velocity at 30 Hz and estimate its second derivative using a local cubic fit:

$$J_{\mathrm{RMS}} = \sqrt{\frac{1}{M}\sum_{i=1}^{M} \|\mathbf{j}(t_i)\|_2^2}, \qquad \mathbf{j}(t_i) = \left.\frac{\mathrm{d}^2\mathbf{v}(t)}{\mathrm{d}t^2}\right|_{t=t_i}, \tag{9}$$

where $\mathbf{v}(t)$ is the measured Cartesian TCP velocity, $\mathbf{j}(t_i)$ is the estimated jerk vector, and M is the number of valid samples in the analysis interval. We compute both metrics over each complete recorded trajectory, including successful and failed attempts. Table 3 reports the median of the trajectory-level scores for each method.

Both smoothness metrics depend on motion speed and range: slower or smaller movements can yield lower scores. They should therefore be interpreted alongside task success.

B.4 Action–State Delay on the xArm7

Figure 7 compares dispatched actions (joint-position commands), controller-reported targets, and measured joint states on the xArm7. We show four evenly spaced trials from the $\delta_0 = 0.3$ table-tennis deployment. Commands are timestamped at the arm SDK call and feedback at receipt, using the same host clock.

The measured joint states visibly lag behind the dispatched actions: sending a target does not mean that the arm has reached it. The traces therefore reveal an action–state delay during robot execution, beyond the time spent on policy inference. Observation age, which ends at action dispatch, does not include this subsequent tracking delay. These traces illustrate the lag during motion rather than establish a single fixed delay for every joint and target.

This action–state delay motivates the temporal-offset bias evaluated in Section 4.4: requesting a target further along the predicted trajectory provides additional lead time for the arm to track it. With $K = 10$ at 30 Hz, $\delta_0 = 0.3$ corresponds to three control steps (100 ms) of lookahead. This is an empirically selected compensation setting, rather than a measured constant execution delay.

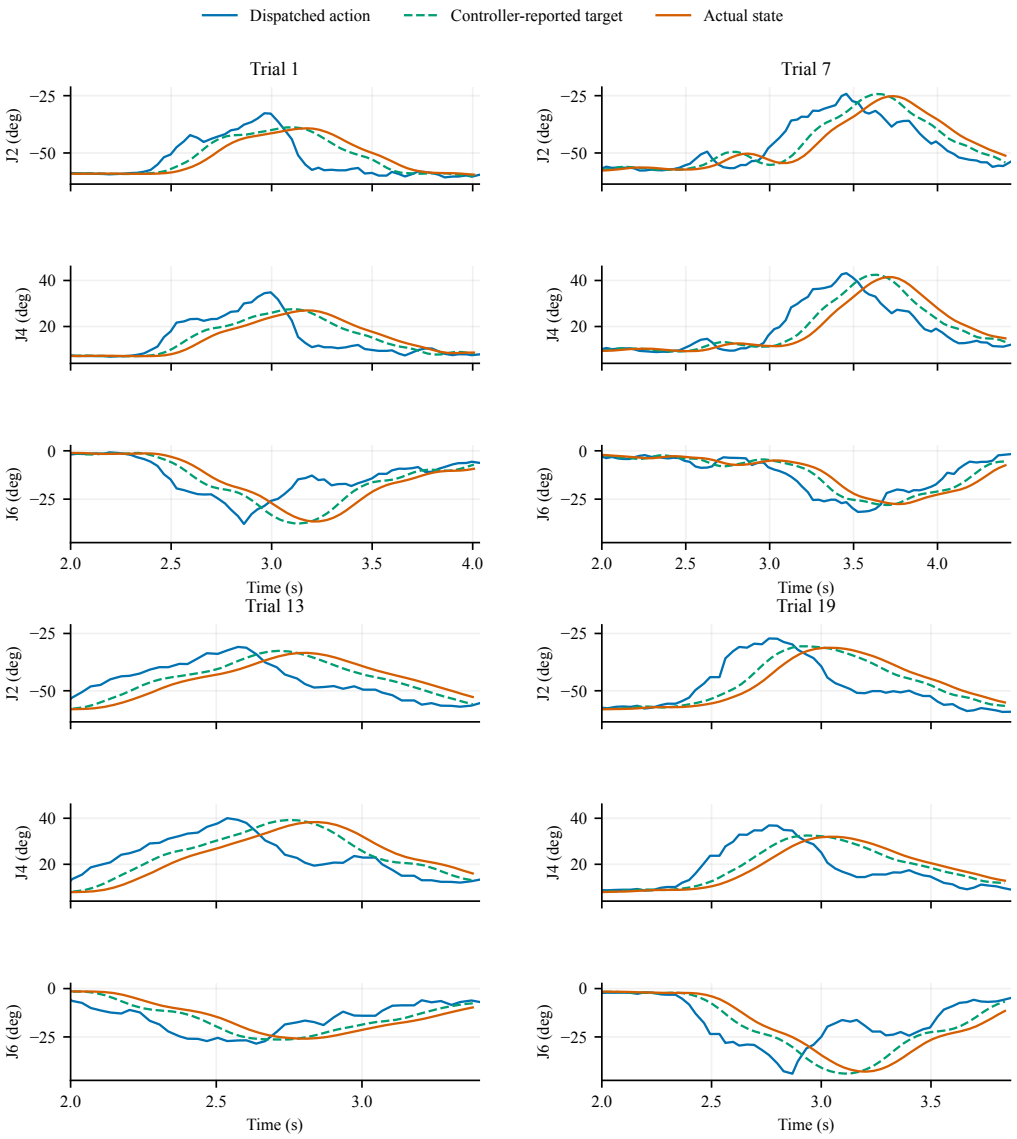

Figure 7: Action–state delay on the xArm7. Joint-space execution traces from four table-tennis trials with temporal-offset bias $\delta_0 = 0.3$. Each panel shows Joints 2, 4, and 6 from 2 s after the first command, omitting the initial stationary period. These joints have the largest commanded excursions; each joint uses the same vertical scale across trials. Blue: dispatched action; dashed green: controller-reported target; orange: measured state.