DON'T THROW AWAY THE TAIL: ACTION UPCYCLING FOR POLICY ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern robot policies predict a chunk of future actions from a single observation, execute only a prefix, and discard the rest before replanning. Choosing the length of this prefix, the execution horizon, poses a trade-off between reactivity and efficiency. A short horizon keeps the policy reactive to the environment, but requires frequent policy calls. Recent test-time methods adaptively select the horizon for each chunk, but they either read model internals, where the signal must be chosen for each architecture, or draw extra samples, which adds cost. We propose *Action Upcycling*, a training-free algorithm that reuses actions the policy would otherwise discard, without accessing model internals or drawing extra samples. We find that discarded actions stay close to their replanned versions as long as the action velocity remains smooth. Action Upcycling therefore extends the execution horizon up to the point where the velocity begins to fluctuate. Extensive experiments on simulated and real-world manipulation tasks show that Action Upcycling reduces policy calls by 1.2–1.7$\times$ with no loss in success rate, across multiple Vision-Language-Action Models (VLAs) and even a World Action Model (WAM). It applies to any chunked policy at negligible cost and is orthogonal to other policy acceleration methods such as few-step sampling and streaming action decoding, opening a new axis for policy acceleration. Videos are available at `https://actionupcycling.github.io/`.

1 INTRODUCTION

Action chunking (Zhao et al., 2023; Chi et al., 2025) has become an essential ingredient in modern robotic imitation learning. Instead of predicting one action per observation, a policy predicts a sequence of H actions, executes the first $h \leq H$ of them in an open-loop manner, and then replans from a fresh observation. We call H the *prediction horizon* and h the *execution horizon*. This recipe is shared by modern VLA models such as SmolVLA (Shukor et al., 2025), GR00T (Bjorck et al., 2025), and the π series (Black et al., 2025b;a; Ai et al., 2026). Recent analyses (Lazzati et al., 2026; Zeng et al., 2026) explain why action chunking helps by showing that demonstrated actions are non-Markovian, so a motion is better predicted as a whole than one action at a time. For example, human demonstrations carry intent across many steps, such as slowing down before contact or pausing at decision boundaries. Predicting a single action cannot reproduce this behavior, whereas predicting a sequence restores the missing temporal correlation. Chunking also reduces compounding error and amortizes the latency of a large backbone over many control steps (Black et al., 2025c).

In practice, most policies do not execute the full action chunk but only a prefix of length h. A short h keeps the policy reactive, since it re-observes the environment more frequently and can adjust its behavior in response to execution errors or unexpected changes in the scene. However, each re-observation requires a full forward pass of a large backbone, so a short h increases the cost. Furthermore, the optimal value of h varies across tasks and even within a single task (Wang et al., 2026). Only recently have test-time methods (Wang et al., 2026; Xu et al., 2026; Liang et al., 2026; Chen et al., 2026) proposed adaptive selection of h for each action chunk instead of fixing it. They differ mainly in the signal they read. *Attention-based* methods track how attention of the action expert is distributed across the chunk (Wang et al., 2026; Xu et al., 2026). This requires looking inside the model, and which denoising steps, layers, and attention types to read must be chosen for each architecture. *Sampling-based* methods draw several candidate chunks and measure where they

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.

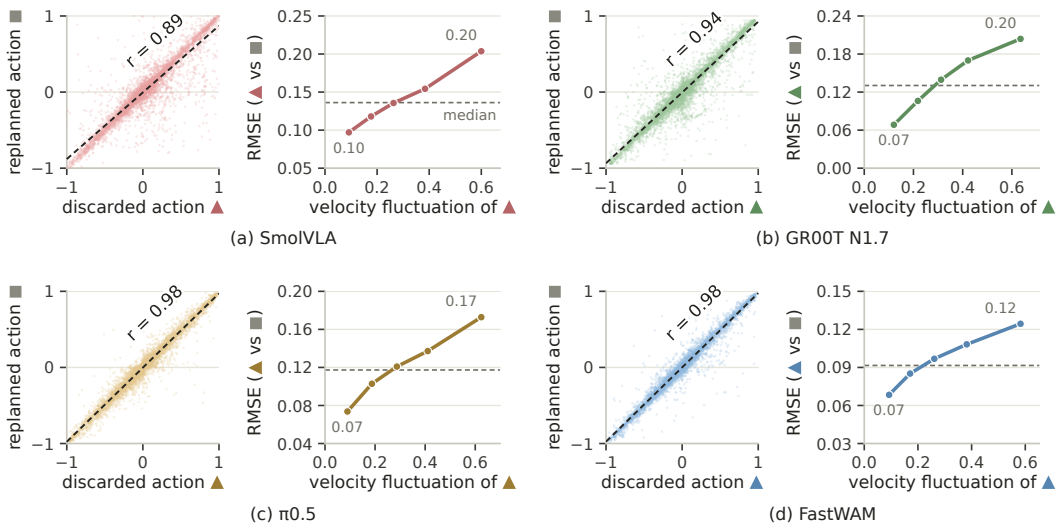


Figure 1: **Discarded actions are worth keeping when motion is smooth.** We compare the discarded actions of each chunk with their replanned versions on LIBERO for (a) SmolVLA, (b) GR00T N1.7, (c) $\pi_{0.5}$, and (d) FastWAM. (*left*) They are strongly correlated. (*right*) Their RMSE grows with the velocity fluctuation of the discarded actions. *Action Upcycling* exploits this property to decide how many discarded actions to reuse.

disagree (Liang et al., 2026; Chen et al., 2026). This requires multiple forward passes, and each additional pass increases the cost.

In this work, we propose *Action Upcycling*, a training-free algorithm that reuses actions the policy would otherwise discard, without accessing model internals or drawing extra samples. It starts from a key observation. As shown in Figure 1 (*left*), the discarded actions are strongly correlated with their replanned versions (Pearson correlation between 0.89 and 0.98 across all four policies). Figure 1 (*right*) further shows that the discarded actions stay close to them as long as velocity remains smooth. Action Upcycling is designed to exploit this property, extending the execution horizon up to the point where the predicted velocity begins to fluctuate. Since the signal is computed from the action chunk that has already been sampled, Action Upcycling is applicable to any chunked policy at negligible cost and is orthogonal to other policy acceleration methods such as few-step sampling and streaming action decoding (Li et al., 2026), opening a new axis for policy acceleration.

We evaluate Action Upcycling on four policies (SmolVLA, $\pi_{0.5}$, GR00T N1.7, FastWAM) across three benchmarks (LIBERO (Liu et al., 2023), LIBERO-Plus (Fei et al., 2025), RoboTwin 2.0 (Chen et al., 2025)), as well as in real-world robot manipulation tasks. Across diverse experiments, it consistently reduces the number of policy calls by 1.2–1.7$\times$ with no loss in success rate. We also demonstrate that Action Upcycling can be seamlessly combined with other policy acceleration methods such as few-step sampling and streaming action decoding. Notably, it works even on a World Action Model without any modification or tuning, supporting its generality beyond VLAs.

Our contributions are:

- We show that discarded actions are strongly correlated with their replanned versions and stay close as long as the action velocity remains smooth.
- We propose Action Upcycling, a training-free algorithm that exploits this property to reuse actions the policy would otherwise discard, without accessing model internals or drawing extra samples.
- Across four policies (three VLAs and a World Action Model), three benchmarks, and real-world robot manipulation tasks, Action Upcycling consistently reduces policy calls by 1.2–1.7$\times$ with no loss in success rate and combines with other policy acceleration methods.

2 Related Work

Action chunking and the execution horizon. Predicting a chunk of H future actions per policy call was popularized by ACT and Diffusion Policy (Zhao et al., 2023; Chi et al., 2025) and is now standard in VLAs such as SmolVLA, GR00T, and the π series (Shukor et al., 2025; Bjorck et al., 2025; Black et al., 2025b;a). Recent analyses attribute its benefit mainly to the non-Markovian nature of demonstrated actions and reduced compounding error (Lazzati et al., 2026; Zeng et al., 2026). However, a policy executing a chunk in an open-loop manner does not see the environment until the chunk ends. BID (Liu et al., 2025a) formalizes this as a trade-off, in which longer execution improves long-term consistency but reduces short-term reactivity. Therefore, most policies execute only the first $h \leq H$ actions and discard the rest in practice. Action Upcycling reuses actions the policy would otherwise discard, reducing unnecessary policy calls and thereby accelerating the policy.

Adaptive execution horizons. A concurrent line of work replaces the fixed h with an adaptive execution horizon for each action chunk. These methods differ mainly in the signal used to set the horizon. *Attention-based* methods read attention weights inside the model. AutoHorizon (Wang et al., 2026) uses self-attention of the action expert, and Knowing When to Stop (Xu et al., 2026) uses the entropy of cross-attention between action and visual tokens. These methods require access to model internals, and which layers, attention types, and denoising steps to read must be chosen for each architecture. *Sampling-based* methods obtain their signal from multiple samples. AAC (Liang et al., 2026) estimates the entropy across the samples and sets the horizon where the entropy increases most rapidly. A3 (Chen et al., 2026) also samples multiple chunks, selects their medoid as a representative chunk, and executes it only up to the point verified by an additional forward pass. These methods require additional forward passes, which increase the computational cost. Unlike these methods, Action Upcycling reads its signal only from a single action chunk, requiring no model internals or extra samples, making it applicable to any chunked policy at negligible cost.

Policy acceleration. Most acceleration methods reduce the cost of each policy call, for example by distilling the sampler into fewer denoising steps (Prasad et al., 2024; Wang et al., 2025), pruning visual tokens or caching intermediate computations (Liu et al., 2025b; Xu et al., 2025; Yang et al., 2025), or skipping unnecessary layers of the backbone (Yue et al., 2024). FlashVLA (Li et al., 2026) further accelerates inference by streaming actions from a buffer of chunks at different noise levels. Action Upcycling reduces the number of policy calls rather than the cost of each call, so the two directions are complementary. We verify this by combining Action Upcycling with both few-step sampling and FlashVLA in Section 4.

3 Action Upcycling

Action Upcycling builds on the observation that discarded actions stay close to their replanned versions as long as the predicted velocity remains smooth (Figure 1). Based on this observation, it decides how many additional actions to execute before replanning, using only the predicted action chunk. In the following, we first formalize the problem (Section 3.1), and then describe the two components of the method (Figure 2): a velocity fluctuation signal (Section 3.2) and an adaptive horizon selection strategy that converts the signal into an execution horizon (Section 3.3).

3.1 Problem Formulation

In the i-th policy call, the policy π predicts a chunk of H future actions from the observation o_{t_i} received at control step t_i:

$$A_i \;=\; \pi(o_{t_i}) \;=\; (a_1, \ldots, a_H). \tag{1}$$

The controller executes the prefix $a_1, \ldots, a_h$ and discards the remaining actions $a_{h+1}, \ldots, a_H$, which we call the *tail*. Action Upcycling reuses the trustworthy part of the tail, so that the i-th call executes $h + \Delta h_i$ actions, where $\Delta h_i \in [0, H - h]$ is the number of reused tail actions. We define the *mean execution length* $\bar{h}$ as the average number of executed actions per call, and the

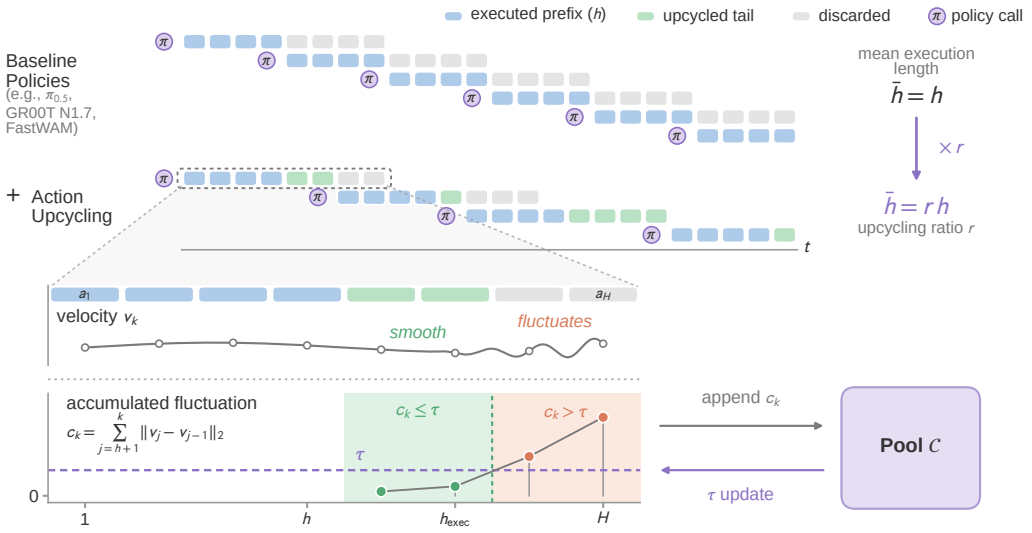


Figure 2: **Overview of Action Upcycling.** *Top*: A chunked policy predicts H actions per call, executes only the first h, and discards the rest (the *tail*). Action Upcycling additionally executes the trustworthy part of the tail, raising the mean execution length from h to rh and reducing policy calls accordingly. *Bottom*: The accumulated velocity fluctuation c_k stays small while the predicted motion is smooth and rises once it fluctuates. Action Upcycling executes tail actions while $c_k \leq \tau$, where τ is determined from a pool $\mathcal{C}$ of signals c_k, so that the mean execution length reaches rh.

upcycling ratio r as its ratio to h:

$$\bar{h} \;=\; \frac{1}{n}\sum_{i=1}^{n}\left(h+\Delta h_i\right), \qquad r \;=\; \frac{\bar{h}}{h} \;\in\; [1,\, H/h], \tag{2}$$

where n is the total number of policy calls. For an episode that requires a given number of actions, a larger r proportionally reduces the number of policy calls. For example, $r = 2$ halves the number of policy calls, and $r = 1$ recovers the default setting of each policy. The number of reused tail actions Δh_i is determined by where the predicted velocity within the chunk begins to fluctuate, as described in Sections 3.2 and 3.3.

3.2 Velocity Fluctuation as a Signal

Let v_k denote the velocity induced by the k-th action in the chunk. Its definition depends on the action parameterization, with $v_k = a_k$ for relative displacements (e.g., LIBERO) and $v_k = a_k - a_{k-1}$ for absolute joint positions (e.g., RoboTwin 2.0). We measure how much the velocity fluctuates beyond the execution horizon h by accumulating the change between consecutive velocities:

$$c_k \;=\; \sum_{j=h+1}^{k} \left\|v_j - v_{j-1}\right\|_2, \qquad h < k \leq H, \tag{3}$$

with $c_h = 0$. Since every summand is non-negative, c_k is non-decreasing along the tail. It stays small while the predicted motion is steady and rises as soon as the velocity fluctuates. Figure 1 motivates this choice, showing that the RMSE between the discarded actions and their replanned versions grows with the velocity fluctuation. Therefore, Action Upcycling uses c_k as a signal for how far the tail can be trusted.

Algorithm 1 Action Upcycling

Require: policy π, execution horizon h, prediction horizon H, upcycling ratio r

$\mathcal{C} \leftarrow$ signals $c_{h+1}, \ldots, c_H$ of chunks predicted by π ▷ Pool construction, Section 3.3

$\tau \leftarrow$ smallest element of $\mathcal{C}$ with $\bar{h}(\tau) \geq r\,h$ ▷ Threshold update, Section 3.3

while task not done **do**

  $A = (a_1, \ldots, a_H) \leftarrow \pi(o_t)$

  $c_k \leftarrow \sum_{j=h+1}^{k} \|v_j - v_{j-1}\|_2, \quad h < k \leq H$ ▷ Velocity fluctuation, Equation (3)

  $h_{\text{exec}} \leftarrow \max_{c_k \leq \tau} k, \quad h \leq k \leq H$ ▷ Adaptive horizon selection, Equation (4)

  execute $a_1, \ldots, a_{h_{\text{exec}}}, \quad t \leftarrow t + h_{\text{exec}}$

end while

3.3 Adaptive Horizon Selection

Given a fluctuation threshold $\tau \geq 0$, we reuse tail actions as long as their accumulated fluctuation does not exceed τ, so that the execution length of the chunk is

$$h_{\text{exec}}(A_i; \tau) \;=\; h + \Delta h_i \;=\; \max_{c_k \leq \tau} k, \qquad h \leq k \leq H, \tag{4}$$

where Δh_i is the number of reused tail actions. Setting $\tau = \infty$ executes every chunk as a whole, and $\tau = 0$ effectively recovers the default setting of each policy.

Since a larger τ allows more tail actions to be reused, the upcycling ratio r grows with τ. This property lets us find the threshold for a given r by a simple search. Specifically, we collect the velocity fluctuation signals $c_{h+1}, \ldots, c_H$ of predicted chunks into a pool $\mathcal{C}$, which requires no additional rollouts since c_k depends only on the chunk itself. We then set τ to the smallest element of $\mathcal{C}$ for which the mean execution length over the pool satisfies $\bar{h}(\tau) \geq r\,h$. The pool $\mathcal{C}$ can be built from rollouts of the policy itself, without demonstrations or success labels. Further analysis of pool construction, including the required pool size and the collection strategy, is provided in Section B.1. The complete Action Upcycling pipeline is summarized in Algorithm 1.

4 Experiments

To validate the effectiveness of Action Upcycling, we conduct extensive simulation and real-world experiments. In the simulation experiments, we evaluate four policies (three VLAs and one World Action Model) on three benchmarks (Section 4.1). In the real-world experiments, we deploy two of these policies on a physical robot and evaluate them on diverse manipulation tasks (Section 4.2).

4.1 Simulation Experiments

We apply Action Upcycling to four representative policies, $\pi_{0.5}$ (Black et al., 2025a), SmolVLA (Shukor et al., 2025), GR00T N1.7 (Bjorck et al., 2025), and FastWAM (Yuan et al., 2026), using their publicly released checkpoints without fine-tuning. All policies predict a chunk of H actions per call and execute the first h actions under their default setting. The horizons and a detailed description of each policy are provided in Section A.1. We evaluate on LIBERO (Liu et al., 2023), LIBERO-Plus (Fei et al., 2025), and RoboTwin 2.0 (Chen et al., 2025), following the official evaluation protocol of each benchmark. The evaluation covers the four task suites of LIBERO, the seven perturbation dimensions of LIBERO-Plus, and the 50 bimanual manipulation tasks of RoboTwin 2.0 in both clean and randomized settings. Please refer to Section A.2 for detailed descriptions of each benchmark.

4.1.1 Quantitative Results

Benefit of Action Upcycling. Table 1 reports the success rate, the number of policy calls per episode, the latency per policy call, and the inference time per episode for four policies on three benchmarks. Across all combinations, Action Upcycling matches or improves the success rate of the baseline policies while reducing policy calls by 1.2–1.7$\times$ in a training-free manner. Since the latency per call remains unchanged, the inference time per episode decreases in proportion to the

Table 1: **Benefit of Action Upcycling.** Results of four policies on three benchmarks. We report success rate (%), policy calls per episode, latency per policy call (ms), and inference time per episode (s). Action Upcycling consistently reduces policy calls by 1.2–1.7× while matching or even improving the success rate. Better value in **bold**.

		Baseline Policy				+ Action Upcycling			
Benchmark	Model	succ. ↑	calls / ep ↓	ms	s / ep ↓	succ. ↑	calls / ep ↓	ms	s / ep ↓
LIBERO	$\pi_{0.5}$	96.9	32.4 (1×)	136	4.41	**97.9**	**21.9 (1.5×)**	137	**3.00**
	SmolVLA	82.6	19.3 (1×)	94	1.82	**82.8**	**14.7 (1.3×)**	94	**1.39**
	GR00T N1.7	96.1	22.6 (1×)	115	2.59	**96.6**	**15.0 (1.5×)**	115	**1.72**
	FastWAM	97.4	15.7 (1×)	84	1.32	**97.6**	**13.4 (1.2×)**	84	**1.12**
LIBERO-Plus	$\pi_{0.5}$	83.7	38.5 (1×)	137	5.27	**86.5**	**23.9 (1.6×)**	137	**3.27**
	SmolVLA	31.8	29.4 (1×)	94	2.77	**33.3**	**24.1 (1.2×)**	94	**2.27**
	GR00T N1.7	82.1	36.4 (1×)	115	4.18	**82.3**	**22.7 (1.6×)**	115	**2.61**
	FastWAM	49.4	33.4 (1×)	84	2.79	**49.6**	**22.5 (1.5×)**	84	**1.88**
RoboTwin 2.0	$\pi_{0.5}$	59.7	40.9 (1×)	152	6.22	**60.5**	**29.8 (1.4×)**	152	**4.53**
	SmolVLA	34.8	50.6 (1×)	114	5.77	**47.4**	**29.7 (1.7×)**	114	**3.39**
	FastWAM	89.5	10.6 (1×)	214	2.27	**90.6**	**8.7 (1.2×)**	214	**1.86**

Table 2: **Comparison with adaptive execution horizon methods.** Results of $\pi_{0.5}$ on three benchmarks. We report success rate (%), policy calls per episode, latency per policy call (ms), and inference time per episode (s). Best in **bold**, second best underlined.

	LIBERO				LIBERO-Plus				RoboTwin 2.0			
Method	succ.	calls / ep	ms	s / ep	succ.	calls / ep	ms	s / ep	succ.	calls / ep	ms	s / ep
Baseline	96.9	32.4 (1×)	136	4.41	83.7	38.5 (1×)	137	5.27	59.7	40.9 (1×)	152	6.22
+ AAC (CVPR 2026)	97.3	23.5 (1.38×)	802	18.9	84.5	27.1 (1.42×)	802	21.7	53.1	46.1 (0.89×)	930	42.9
+ AutoHorizon (ECCV 2026)	97.4	22.3 (1.45×)	136	3.03	85.5	24.5 (1.57×)	136	3.33	56.8	**28.0 (1.46×)**	152	**4.26**
+ Action Upcycling	**97.9**	**21.9 (1.48×)**	137	**3.00**	**86.5**	**23.9 (1.61×)**	137	**3.27**	**60.5**	29.8 (1.37×)	152	4.53

number of calls. On $\pi_{0.5}$, for instance, Action Upcycling reduces policy calls by 1.4–1.6× and improves the success rate on all three benchmarks, from 96.9% to 97.9% on LIBERO, from 83.7% to 86.5% on LIBERO-Plus, and from 59.7% to 60.5% on RoboTwin 2.0. Action Upcycling also works on FastWAM without any modification, reducing its calls by 1.2–1.5× while improving the success rate, which demonstrates its effectiveness beyond VLAs.

Comparison with adaptive execution horizon methods. We compare Action Upcycling with two representative adaptive execution horizon methods on $\pi_{0.5}$, the attention-based AutoHorizon (Wang et al., 2026) and the sampling-based AAC (Liang et al., 2026). Table 2 reports the success rate, the number of policy calls per episode, the latency per policy call, and the inference time per episode on three benchmarks. On LIBERO and LIBERO-Plus, Action Upcycling achieves the highest success rate, the largest call reduction, and the shortest inference time per episode, without access to model internals. AAC reduces policy calls, but it draws multiple samples per call, which increases both the latency per call and the inference time per episode. On RoboTwin 2.0, AutoHorizon reduces policy calls slightly more but degrades the success rate, and AAC even increases policy calls, whereas Action Upcycling improves the success rate while still reducing policy calls.

Combination with other policy acceleration methods. Most policy acceleration methods reduce the cost of each policy call, whereas Action Upcycling reduces the number of calls, so the two directions are complementary. We combine Action Upcycling with two such methods (few-step sampling and streaming action decoding with FlashVLA (Li et al., 2026)) on $\pi_{0.5}$. Table 3 shows that, even with fewer denoising steps (N=5 and N=2), Action Upcycling keeps the call reduction at about 1.5× and improves the success rate over the baseline with the same N. As a result, combining Action Upcycling with few-step sampling raises the total speed-up to 2.8× at N=2. Table 4 shows that FlashVLA reduces the latency per policy call from 136 ms to 32 ms. Building on this, Action Upcycling reduces the number of policy calls by 1.51× while improving the success rate. As a result, combining Action Upcycling with FlashVLA yields a total speed-up of 6.7×. Implementation details of all compared and combined methods are provided in Section A.3.

Table 3: **Action Upcycling stacks with denoising-step reduction.** Results of $\pi_{0.5}$ with N denoising steps on LIBERO. We report success rate (%), policy calls per episode, latency per policy call (ms), inference time per episode (s), and speed-up over the N=10 baseline. Better value at each N in **bold**.

Setting	succ. ↑	calls / ep ↓	ms	s / ep ↓	speed-up ↑
Baseline (N=10)	96.9	32.4 (1×)	136	4.41	1.0×
+ Action Upcycling	**97.9**	**21.9 (1.48×)**	137	**3.00**	**1.5×**
N=5	96.9	32.4 (1×)	97	3.14	1.4×
+ Action Upcycling	**97.7**	**21.9 (1.48×)**	98	**2.15**	**2.1×**
N=2	96.8	32.0 (1×)	69	2.21	2.0×
+ Action Upcycling	**97.8**	**21.8 (1.47×)**	71	**1.55**	**2.8×**

Table 4: **Action Upcycling stacks with streaming action decoding.** Results of $\pi_{0.5}$ with FlashVLA on LIBERO. FlashVLA denoises a rolling buffer of chunks at different noise levels for faster decoding. We report success rate (%), policy calls per episode, latency per policy call (ms), inference time per episode (s), and speed-up over $\pi_{0.5}$. Best in **bold**.

Setting	succ. ↑	calls / ep ↓	ms	s / ep ↓	speed-up ↑
Baseline	96.90	32.4 (1×)	136	4.41	1.0×
+ FlashVLA	98.65	31.4 (1.03×)	32	1.01	4.4×
+ FlashVLA & Action Upcycling	**98.70**	**21.5 (1.51×)**	31	**0.66**	**6.7×**

4.1.2 Ablation Study

One may attribute the gain of Action Upcycling simply to executing more actions per call. To examine this possibility, we replace our adaptive execution with fixed-length execution, where the length is set close to the mean execution length $\bar{h}$ of Action Upcycling. Table 5 shows that fixed-length execution degrades the success rate by 0.7–1.1 points compared to the baseline for three out of four policies. In contrast, with a similar number of policy calls, our adaptive execution consistently improves the success rate by 0.9–1.3 points over fixed-length execution across all four policies. Furthermore, Figure 3 shows that the execution lengths selected by Action Upcycling do not concentrate on a single value but spread over a range. This indicates that Action Upcycling dynamically adapts the number of tail actions to execute for each chunk based on the velocity fluctuation signal. Further analysis of how each policy behaves under different upcycling ratios is provided in Section B.2.

Table 5: **Adaptive vs. fixed-length execution.** Results of four policies on LIBERO. Fixed-length execution uses a length close to the mean execution length $\bar{h}$ of Action Upcycling. We report success rate (%) and $\bar{h}$. Better value in **bold**.

	Baseline		Fixed-length		Adaptive (ours)	
Model	succ. ↑	$\bar{h}$	succ. ↑	$\bar{h}$	succ. ↑	$\bar{h}$
$\pi_{0.5}$	96.90	5.0	97.05	7.0	**97.90**	7.3
SmolVLA	82.60	10.0	81.55	13.0	**82.80**	13.3
GR00T N1.7	96.05	8.0	95.40	12.0	**96.55**	11.9
FastWAM	97.35	10.0	96.45	12.0	**97.55**	11.9

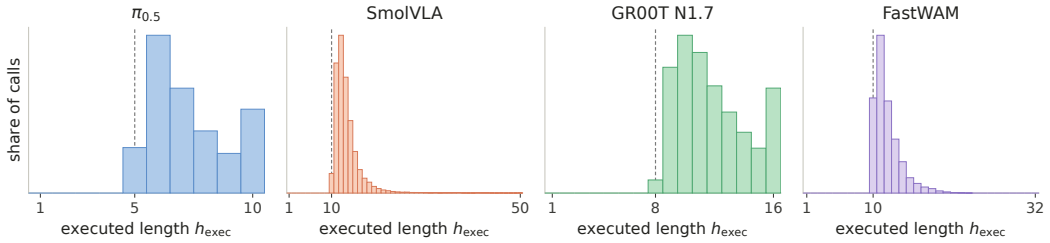


Figure 3: **Distribution of execution lengths.** The execution lengths selected by Action Upcycling spread over a range rather than concentrating on a single value.

4.2 REAL-WORLD EXPERIMENTS

We evaluate Action Upcycling on real-world robot manipulation tasks to verify that its effectiveness carries over to physical execution. We use $\pi_{0.5}$ and GR00T N1.6 as base policies and deploy them on the YAM arm, a 6-DoF single-arm manipulator with a linear gripper. We describe the data collection, policy training, and evaluation protocol below.

Data collection. We collect 424 teleoperated episodes on the YAM arm with a leader arm, covering eight tasks with about 53 episodes each. The tasks consist of four pick-and-place tasks and four long-horizon drawer tasks that require opening a drawer, moving an object, and closing the drawer. Each episode records a top-down camera view, a wrist camera view, the joint state, and the action, represented as absolute joint positions, at 30 Hz.

Policy training. We fine-tune both policies on the collected data with their official training recipes, using 20,000 steps with a batch size of 64 for $\pi_{0.5}$ and 10,000 steps with a batch size of 32 for GR00T N1.6, while keeping all other settings at their defaults. Both policies are trained to predict absolute joint positions as actions, with a prediction horizon of H=50 actions for $\pi_{0.5}$ and H=16 actions for GR00T N1.6.

Evaluation. We evaluate $\pi_{0.5}$ on all eight tasks and GR00T N1.6 on the four pick-and-place tasks, with 10 episodes per task. The default execution horizon is h=15 for $\pi_{0.5}$ and h=9 for GR00T N1.6, which gave the most stable execution in our preliminary trials. Since actions are absolute joint positions, we compute v_k as the difference of consecutive joint positions, excluding the gripper. We set τ using r=1.25 for $\pi_{0.5}$ and r=1.1 for GR00T N1.6. The baseline and Action Upcycling share the same checkpoint and control settings. An episode counts as a success only if every step of the instruction is completed, with no partial credit. Further details on the robot setup and computing resources are provided in Sections A.4 and A.5.

4.2.1 QUANTITATIVE RESULTS

Tables 6 and 7 report success rate, policy calls per episode, and time per episode for each task with $\pi_{0.5}$ and GR00T N1.6. For both policies, Action Upcycling matches or improves the success rate on every task while reducing policy calls.

For $\pi_{0.5}$, the overall success rate increases from 72/80 to 77/80, and policy calls decrease from 48.3 to 34.0 per episode. On the pick-and-place tasks, where the baseline already succeeds in every episode, Action Upcycling maintains the perfect success rate while reducing policy calls by 1.30$\times$ on average. On the long-horizon drawer tasks, Action Upcycling even improves the success rate from 32/40 to 37/40 and reduces policy calls by 1.48$\times$.

For GR00T N1.6, the overall success rate slightly increases from 34/40 to 35/40, and policy calls decrease from 90.6 to 71.0 per episode. The time per episode also decreases for both policies, from 24.9 s to 21.5 s for $\pi_{0.5}$ and from 28.0 s to 24.9 s for GR00T N1.6, indicating that the policy call reduction from Action Upcycling translates into faster task completion in real-world deployment.

Table 6: **Quantitative real-world results with $\pi_{0.5}$.** Each task is evaluated over 10 episodes. We report success rate, policy calls per episode, and time per episode (s). Better value in **bold**.

	Success rate ↑		Calls / ep ↓		Time / ep (s) ↓	
Task	Baseline	Action Upcycling	Baseline	Action Upcycling	Baseline	Action Upcycling
Soccer ball → plate (layout 1)	10/10	10/10	26.9	**26.8**	**14.4**	17.4
Soccer ball → plate (layout 2)	10/10	10/10	36.5	**20.7**	19.0	**12.9**
Basketball → basket	10/10	10/10	31.3	**24.9**	16.5	**15.8**
Baseball → basket	10/10	10/10	32.1	**25.0**	17.1	**16.1**
Drawer: black ball → drawer	8/10	**10/10**	73.8	**49.2**	37.3	**30.9**
Drawer: basketball → drawer	8/10	8/10	61.2	**40.7**	31.6	**25.5**
Drawer: baseball → basket	7/10	**9/10**	67.9	**40.6**	34.3	**25.5**
Drawer: basketball → pot	9/10	**10/10**	56.9	**44.5**	28.7	**27.8**
Total / mean	72/80	**77/80**	48.3	**34.0**	24.9	**21.5**

Table 7: **Quantitative real-world results with GR00T N1.6.** Each task is evaluated over 10 episodes. We report success rate, policy calls per episode, and time per episode (s). Better value in **bold**.

	Success rate ↑		Calls / ep ↓		Time / ep (s) ↓	
Task	Baseline	Action Upcycling	Baseline	Action Upcycling	Baseline	Action Upcycling
Soccer ball → plate (layout 1)	10/10	10/10	56.8	**50.3**	17.8	**17.7**
Soccer ball → plate (layout 2)	8/10	**9/10**	102.2	**81.7**	31.5	**28.8**
Basketball → basket	8/10	8/10	95.6	**69.9**	29.6	**24.3**
Baseball → basket	8/10	8/10	107.9	**82.3**	33.1	**28.9**
Total / mean	34/40	**35/40**	90.6	**71.0**	28.0	**24.9**

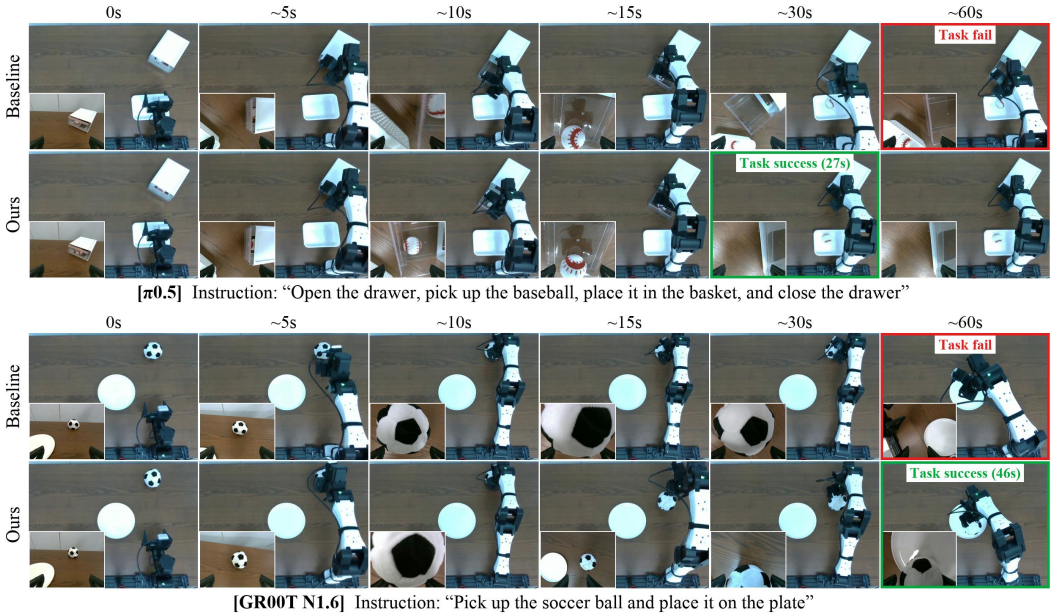


Figure 4: **Qualitative real-world results.** A drawer task with $\pi_{0.5}$ (top) and a pick-and-place task with GR00T N1.6 (bottom), each shown with the baseline policy and with Action Upcycling. Red and green boxes mark failure and success, respectively. In both tasks, the baseline policy times out before completion, while the same policy with Action Upcycling succeeds. Videos are available on the project page.

4.2.2 Qualitative Results

Figure 4 shows real-world rollouts of $\pi_{0.5}$ on a drawer task and of GR00T N1.6 on a pick-and-place task. In both tasks, the baseline policy times out at 60 s before completion, whereas the same policy with Action Upcycling succeeds in 27 s and 46 s, respectively. In episodes where the baseline policy also succeeds, the same policy with Action Upcycling tends to complete the task faster, as shown in Section C. Videos of the real-world rollouts are available on the project page.

5 Conclusion

We proposed Action Upcycling, a simple but effective algorithm that reuses the tail of each action chunk, which chunked policies usually discard. We showed that discarded actions stay close to replanned versions as long as their motion remains smooth. We used the accumulated velocity fluctuation as a signal for how far the tail can be trusted. Since the signal is computed from a single predicted chunk, Action Upcycling requires no access to model internals, no extra samples, and no additional training. Across three VLAs and a World Action Model, three benchmarks, and real-world experiments, Action Upcycling demonstrates its effectiveness and robustness in policy acceleration and composes with orthogonal acceleration methods, opening a new axis for faster policy inference.

AI USE STATEMENT

In this work, we used generative AI tools to assist in implementing methods and writing experiment code. We have not used generative AI tools for other tasks with required disclosure. Additionally, we used generative AI tools to edit the manuscript for readability and to assist in creating figures. We have reviewed all AI-assisted work. AI-assisted code, text, and figures were all verified for correctness by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects research. Real-robot demonstrations were collected by the authors in a controlled laboratory environment and contain no personally identifiable information. The collected real-robot data and fine-tuned weights are not publicly released. Our method accelerates inference of existing policies without additional training, so it does not introduce new capabilities beyond those of the underlying models.

REPRODUCIBILITY STATEMENT

We describe the proposed method in Section 3 and provide further implementation details in Appendix A. Since our method is training-free, all simulation results can be reproduced using publicly available policy checkpoints and benchmarks without additional training. We will publicly release the source code upon acceptance. Real-robot data and fine-tuned weights are not publicly released, but the data collection setup and fine-tuning procedure are described in Appendix A.

REFERENCES

Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, et al. Motus: A unified latent action world model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 35101–35113, 2026.

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, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025a. URL https://openreview.net/forum?id=vlhoswksBO.

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

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2025c.

Feng Chen, Xianghui Wang, Yuxuan Chen, Boying Li, Yefei He, Zeyu Zhang, and Yicheng Wu. Dynamic execution commitment of vision-language-action models. *arXiv preprint arXiv:2605.11567*, 2026.

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*, 2025.

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.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Filippo Lazzati, Kyle Stachowicz, William Chen, Alberto Maria Metelli, Andrew Wagenmaker, and Sergey Levine. Why does action chunking improve behavioral cloning performance in robotic control? *arXiv preprint arXiv:2608.02547*, 2026.

Zekai Li, Jiaming Tang, and Zhijian Liu. Flashvla: Streaming action decoding for fast and asynchronous vla inference. *arXiv preprint arXiv:2608.27384*, 2026.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20802–20811, June 2026.

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.

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

Ziyan Liu, Yeqiu Chen, Hongyi Cai, Tao Lin, Shuo Yang, Zheng Liu, and Bo Zhao. Bridging the semantic-action gap in visual token pruning for efficient vla inference. *arXiv preprint arXiv:2511.16449*, 2025b.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. Vla knows its limits: Adaptive execution horizons for robot policies. In *European Conference on Computer Vision*, pp. 607–623. Springer, 2026.

Zhendong Wang, Max Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, Ming-Yu Liu, and Yu Zeng. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=E2VsqgKNlr.

Runze Xu, Xiaolong Shan, Shuang Dai, Yu Wang, and Jincheng Yu. Knowing when to stop: Adaptive action chunking via internal cross-attention dynamics in vlas. *arXiv preprint arXiv:2609.00908*, 2026.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *Advances in Neural Information Processing Systems*, 38:164448–164473, 2025.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40891–40914, 2025.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

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

Michael Zeng, Abhinav Agarwal, Ajay Bati, Brian Lee, Siddharth Ancha, and Russ Tedrake. Revisiting open-loop execution in robotics: Toward reactive, higher-performing policies. *arXiv preprint arXiv:2608.15938*, 2026.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016.

A Further Implementation Details

A.1 Policies

We apply Action Upcycling to the following policies using their released checkpoints without any fine-tuning for simulation experiments. The execution and prediction horizons of each policy are listed in Table 8.

- $\pi_{0.5}$ (Black et al., 2025a) is a VLA with a flow-matching action expert that attends to vision-language tokens at every layer through shared attention. We use N=10 denoising steps. For RoboTwin 2.0, we use the public checkpoint released by Motus (Bi et al., 2026).
- **SmolVLA** (Shukor et al., 2025) is a compact VLA with a flow-matching action expert that consists of alternating cross-attention and self-attention blocks and attends to a frozen vision-language model. We use N=10 denoising steps.
- **GR00T N1.7** (Bjorck et al., 2025) is a VLA with a flow-matching action expert, which receives features from a frozen vision-language model through cross-attention. We use N=4 denoising steps.
- **FastWAM** (Yuan et al., 2026) is a World Action Model that generates actions jointly with future video latents. We use N=10 denoising steps.

Table 8: **Horizon settings.** Execution horizon h and prediction horizon H of each model and benchmark.

Model	LIBERO	LIBERO-Plus	RoboTwin 2.0
$\pi_{0.5}$	5 / 10	5 / 10	10 / 32
SmolVLA	10 / 50	10 / 50	10 / 50
GR00T N1.7	8 / 16	8 / 16	—
FastWAM	10 / 32	10 / 32	24 / 32

A.2 Benchmarks

We follow the official evaluation protocol of each benchmark unless otherwise specified.

- **LIBERO** (Liu et al., 2023) is a simulated single-arm manipulation benchmark consisting of four suites that focus on spatial layouts, objects, task goals, and long-horizon tasks, with ten tasks each. The action is a relative end-effector displacement.
- **LIBERO-Plus** (Fei et al., 2025) extends LIBERO to evaluate robustness under seven perturbation dimensions, namely camera viewpoint, robot initial state, language, lighting, background, sensor noise, and object layout. It shares the action space of LIBERO.
- **RoboTwin 2.0** (Chen et al., 2025) is a bimanual manipulation benchmark consisting of 50 tasks, which we evaluate in both clean and randomized settings. The action is an absolute joint position.

A.3 Baselines

We compare or combine Action Upcycling with the following methods, using their official implementations and default hyperparameters unless otherwise specified.

- **AAC** (Liang et al., 2026) samples multiple chunks per policy call and executes actions up to the point where the entropy across the samples increases most rapidly. We use K=20 samples and the default hyperparameters.
- **AutoHorizon** (Wang et al., 2026) determines the execution horizon from the attention of the action expert. We use the official soft pointer implementation.
- **FlashVLA** (Li et al., 2026) advances a rolling buffer of four action chunks by one denoising step per call, thereby reducing the latency of each call. We use the $\pi_{0.5}$ checkpoint fine-tuned by the authors and execute the same number of actions per call as in the baseline.

A.4 Real-World Experimental Setup

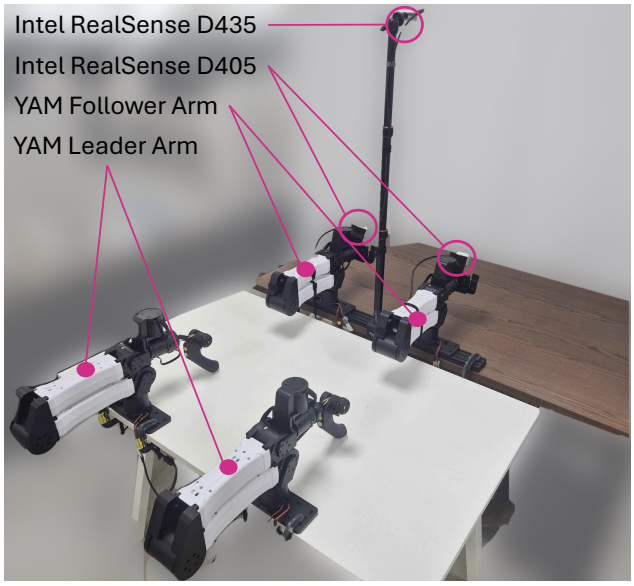


Figure 5: **Real-robot setup.** Two YAM follower arms with wrist-mounted Intel RealSense D405 cameras, a top-mounted Intel RealSense D435, and two YAM leader arms used for teleoperated data collection. All experiments use only the right follower arm.

Robot setup. Figure 5 shows the robot station used in our real-world experiments. It consists of two YAM follower arms, driven over a CAN bus with joint-position PD control, and a matching pair of YAM leader arms for teleoperation. All experiments in this paper use only the right follower arm. An Intel RealSense D435 mounted on a pole above the table provides the top-down view, and an Intel RealSense D405 on the wrist of each arm provides the wrist view. The policy receives the RGB streams from the two cameras at a resolution of 640×480, along with the joint state, consisting of six joint positions and the gripper opening, and outputs targets for the same seven dimensions. The joint velocity is limited to 0.75 rad/s. All episodes are stored in the LeRobot format.

Fine-tuning $\pi_{0.5}$. We fine-tune the released $\pi_{0.5}$ base checkpoint with the official post-training recipe of openpi. The YAM state and action are 7-dimensional, whereas $\pi_{0.5}$ operates on a 32-dimensional state and action space. Following the recipe, both are zero-padded to 32 dimensions and normalized with per-dimension statistics computed on our dataset, and the padded dimensions are ignored at deployment. All parameters are updated during fine-tuning. We train for 20,000 steps with a batch size of 64, keeping the other settings of the recipe unchanged.

Fine-tuning GR00T N1.6. We fine-tune GR00T N1.6 with the official recipe of Isaac-GR00T. The YAM is registered as a new embodiment, with a modality configuration that declares the state and action fields (six arm joints and the gripper) and the two camera views. Following the default recipe for new embodiments, the vision-language backbone is kept frozen, and only the embodiment-specific projectors and the diffusion transformer are fine-tuned. Arm actions are predicted relative to the current joint state, while the gripper action is predicted as an absolute value. State and action are normalized with per-dimension statistics computed on our dataset. We train for 10,000 steps with a batch size of 32, keeping the other settings of the recipe unchanged.

A.5 Computing Resources

All simulation experiments are run on NVIDIA RTX 3090 GPUs, with one policy per GPU. The latency per policy call and the inference time per episode reported in Tables 1 to 4 are measured on a single NVIDIA RTX 4090 GPU. For the real-world experiments in Section 4.2, $\pi_{0.5}$ and GR00T N1.6 are fine-tuned on a single NVIDIA B200 GPU and deployed on a single NVIDIA RTX 4090 GPU.

B Further Analysis

B.1 Pool Construction

The pool $\mathcal{C}$ of velocity fluctuation signals can be built from rollouts of the policy itself, without demonstrations or success labels (Section 3.3). Table 9 shows that a small pool suffices for selecting τ. Using the signals of only a random 2 % of the chunks yields nearly the same τ and resulting mean execution length $\bar{h}$ as using all of them, across all four policies.

Furthermore, the pool need not be collected in advance. While the results above use an offline pool collected before deployment, we can also build an online pool of signals collected during deployment and update τ accordingly. As shown in Table 10, the online pool achieves a comparable success rate and call reduction to the offline pool. In practice, we use the offline pool in all experiments, as it provides a slightly higher success rate and a shorter episode time.

Table 9: **A small pool suffices for selecting τ.** Selected τ and resulting mean execution length $\bar{h}$ on LIBERO, using pools built from all chunks and from a random 2 % of them.

	All chunks		2 % of chunks	
Model	τ	$\bar{h}$	τ	$\bar{h}$
$\pi_{0.5}$	0.15	7.3	0.15	7.3
SmolVLA	0.65	13.3	0.64	13.2
GR00T N1.7	0.22	11.9	0.22	11.9
FastWAM	0.09	11.9	0.09	11.9

Table 10: **Offline vs. online pool construction.** Results of $\pi_{0.5}$ on LIBERO. Best in **bold**.

Method	succ. ↑	calls / ep ↓	ms / call ↓	s / ep ↓
Baseline	96.9	32.4 (1×)	**136**	4.41
+ Action Upcycling (offline pool $\mathcal{C}$)	**97.9**	21.9 (1.48×)	137	**3.00**
+ Action Upcycling (online pool $\mathcal{C}$)	97.7	**21.0 (1.54×)**	150	3.15

B.2 Effect of the Upcycling Ratio

Figure 6 shows how the success rate of each policy on LIBERO behaves as the upcycling ratio r increases from the default setting (r=1). For every policy, there is an upcycling ratio between 1.2 and 1.7 that slightly improves the success rate while reducing policy calls, in line with the main results in Table 1. Beyond this ratio, the success rate gradually drops, likely because a larger r admits tail actions with larger velocity fluctuation.

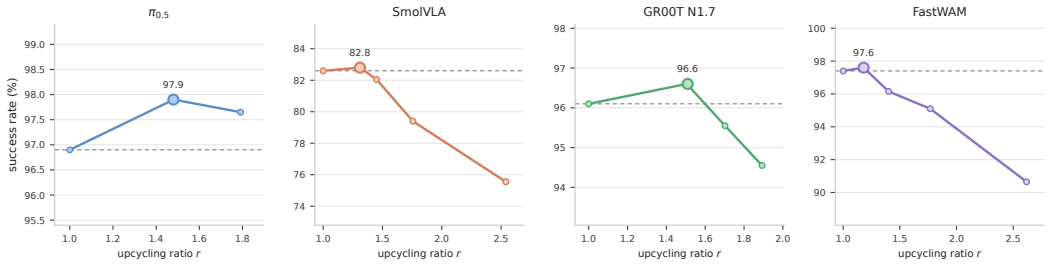


Figure 6: **Effect of the upcycling ratio.** Success rate (%) of each policy on LIBERO as r increases from the default setting (r=1).

C More Qualitative Real-World Results



Figure 7: **Additional qualitative real-world results with $\pi_{0.5}$.** Four drawer tasks, each shown with the baseline policy (top) and with Action Upcycling (bottom). Red and green boxes mark failure and success, respectively. In two tasks, the baseline policy times out before completion, while the same policy with Action Upcycling succeeds. In the other two tasks, both succeed, but the policy with Action Upcycling completes the task faster.



Figure 8: **Additional qualitative real-world results with GR00T N1.6.** Four pick-and-place tasks, each shown with the baseline policy (top) and with Action Upcycling (bottom). Red and green boxes mark failure and success, respectively. In two tasks, the baseline policy times out before completion, while the same policy with Action Upcycling succeeds. In the other two tasks, both succeed, but the policy with Action Upcycling completes the task faster.