ACCELERATING ROBOTIC POLICIES WITH CROSS-CYCLE BATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent generative robot policies produce action chunks through a sequence of dependent network forward passes. Each pass processes only a short action sequence, which can leave the GPU underutilized. Processing several updates together could use this spare capacity to reduce latency. However, successive updates of the same chunk must remain sequential. Overlapping prediction windows allow future chunks to begin sampling before their execution cycle. We use this overlap to develop *Cross-Cycle Batching*, a training-free inference schedule for pretrained flow policies. It staggers full-horizon chunks across control cycles, allowing their next updates to be batched under the latest observation. This reduces sequential calls per cycle in steady state while using the same per-chunk sampling budget as serial. Because each chunk retains the policy's standard input form, the weights, architecture, and update rule remain unchanged. Experiments show up to a $3.1\times$ denoising speedup while preserving task success, with benefits across policy architectures and in real-robot deployment.

1 INTRODUCTION

Recent generative robot policies span world-action models (WAMs) (Shen et al., 2026; Ye et al., 2026), video-action models (Li et al., 2025), and vision-language-action (VLA) models (Black et al., 2025b), many of which generate action chunks through iterative diffusion or flow sampling. At inference time, these policies start from noise and refine an action chunk through repeated forward passes (Lipman et al., 2023; Black et al., 2025b). Each step takes the result of the previous step as its input, making this denoising process sequential and difficult to accelerate. A straightforward way to reduce denoising latency is to use fewer forward passes, but doing so also reduces the sampling budget. Maintaining performance with fewer passes commonly requires retraining or distillation (Prasad et al., 2024; Wang et al., 2025b; Zhang et al., 2025). Given this trade-off, how can we reduce denoising latency while preserving a pretrained policy's full sampling budget?

We approach this problem by making better use of the GPU. An action chunk typically contains only a few dozen action tokens, so processing one chunk per forward pass can leave the GPU underutilized (Black et al., 2025b; Yuan et al., 2026; Wang et al., 2025a) (Figure 1b). This spare capacity offers an opportunity to reduce denoising latency by performing more computation in parallel. A straightforward way to do so is to batch multiple denoising updates into one model call, allowing more work to be completed with only a small increase in runtime (Kodaira et al., 2025; Wang et al., 2025a). However, the dependency between successive steps prevents us from batching denoising updates within the same chunk.

Although the denoising steps within an action chunk depend on one another and must be executed sequentially, action chunks for different control cycles do not share this dependency. This suggests that these chunks could be processed together in a larger batch. The challenge is to use the current observation to begin denoising an action chunk to be executed in a later control cycle. Receding-horizon control (Chi et al., 2023; Black et al., 2025c) naturally provides this opportunity because consecutive prediction windows overlap and actions executed in a later cycle already appear within the current prediction horizon (Figure 1a). Sampling for later cycles can therefore begin under the current observation and continue as new observations arrive, bringing the ready updates from different cycles into the same batch (Høeg et al., 2025; Chen et al., 2025; Kodaira et al., 2025).

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.

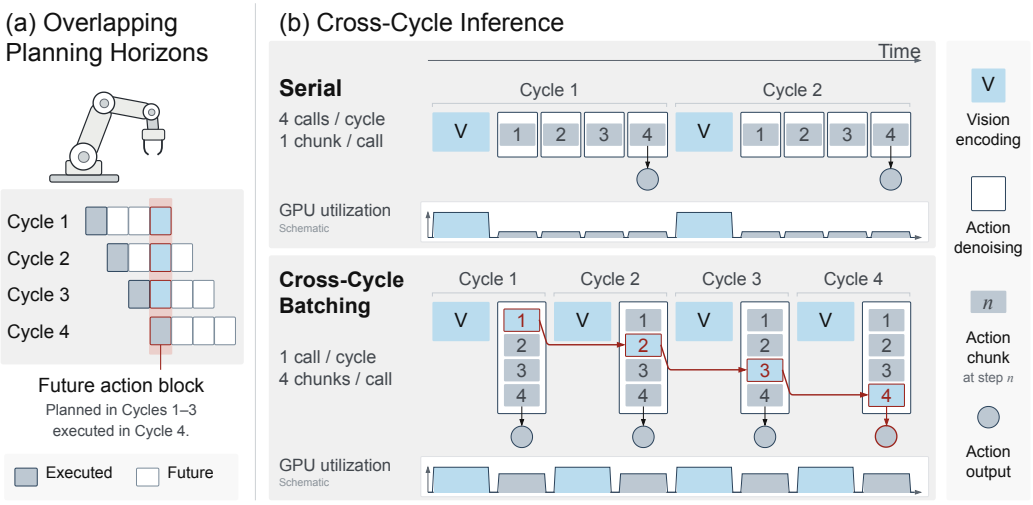


Figure 1: Cross-Cycle Batching at a glance. (a) The highlighted action block is planned across overlapping prediction windows before execution. (b) In steady state, Cross-Cycle Batching advances four chunks in one batched denoising call per cycle, while serial sampling requires four calls. Red traces one chunk's ordered updates. Both schedules encode observations once per cycle.

Building on this opportunity, we propose *Cross-Cycle Batching*, a training-free inference schedule for pretrained flow policies. Instead of completing one action chunk at a time, it keeps several full-horizon chunks at staggered sampling stages and batches their ready denoising updates in each model call. At every control cycle, the most advanced chunk supplies the next action prefix, while the remaining chunks continue sampling for later cycles. Each completed chunk uses the same sampling budget as serial, but the controller waits for fewer sequential model calls per cycle (Figure 1b). Because every batch element remains a standard full-horizon chunk at a single flow time, the method requires only lightweight tail padding as the prediction horizon shifts, with no change to the policy weights, architecture, training procedure, or denoising update rule.

Our analysis and extensive experiments show that Cross-Cycle Batching accelerates inference while preserving task success. On Fast-WAM (Yuan et al., 2026), it achieves up to $2.9\times$ denoising and $1.8\times$ end-to-end speedups over full-budget serial. At matched sequential call counts, it avoids the success-rate drops observed with few-step serial and mixed-progress sampling. These benefits extend across tested policy families and model sizes. Integration with real-time chunking (Black et al., 2025c) enables faster real-robot inference while preserving success across four tasks.

Our contributions are threefold. (i) We introduce Cross-Cycle Batching, a training-free inference schedule that turns prediction-window overlap into batch parallelism within a single controller. It reduces sequential model calls per cycle in steady state while using the same per-chunk sampling budget as serial. (ii) We characterize the latency gains and their limits through analysis and profiling. We relate denoising speedup to batched model-call costs and explain how observation prefill limits the end-to-end gain. (iii) We validate the method in simulation and real-robot deployment. Evaluations across policy families and model sizes show inference speedups while preserving the pretrained policies' task performance.

2 Related Work

Efficient Generative Policy Inference. Diffusion and flow-based policies transform noise into action chunks through a sequence of dependent denoising steps, each requiring a model call (Chi et al., 2023; Black et al., 2025b). Their inference latency is therefore governed by the number and cost of these calls. Prior work reduces the number of calls using more efficient sampling methods and timestep schedules, or learns few-step policies through distillation and consistency objectives (Lu et al., 2022; Sabour et al., 2024; Prasad et al., 2024; Wang et al., 2025b; Zhang et al., 2025). FASTER (Lu et al., 2026) instead assigns fewer denoising steps to near-term actions and

streams them as soon as they are completed, reducing time to first action through schedule-aware fine-tuning. Other methods shorten the sampling path through warm starts or reduce each call's cost through caching, sparsity, pruning, and quantization (Duan et al., 2025; Ji et al., 2026a;b; Li et al., 2023). Together, these approaches reduce or approximate the computation devoted to each action chunk, often through additional training or computation reuse. Cross-Cycle Batching uses the same per-chunk sampling budget as serial and lowers latency by staggering chunks across control cycles and batching their ready denoising updates.

Rolling Sampling for Sequential Generation. Rolling samplers continuously extend a sequence by tying sampling progress to temporal position within an evolving window. Rolling Diffusion Models (Ruhe et al., 2024), Diffusion Forcing (Chen et al., 2024a), TEDi (Zhang et al., 2024), and FIFO-Diffusion (Kim et al., 2024) realize this idea through position-wise noise schedules or diagonal denoising queues. Robot policies have also adopted position-dependent sampling progress within an action sequence. Streaming Diffusion Policy (Høeg et al., 2025) and RNR-DP (Chen et al., 2025) maintain persistent rolling action buffers, while $\pi\mathbf{R}^2$ (Park & Tulsiani, 2026) uses a latency-adaptive staircase buffer that advances across policy calls. Cross-Cycle Batching retains the first-in, first-out execution pattern of rolling buffers, but moves sampling-progress variation from the action dimension to the batch dimension. This separation exposes the ready updates of independent full-horizon chunks as parallel work, while each chunk remains a standard single-time input to the pretrained policy and requires no retraining.

Parallel and Batched Generative Inference. Parallel generative inference requires several model evaluations to be ready at the same time. Existing methods obtain these evaluations in different ways. Distributed inference systems expose concurrency by partitioning tokens or model layers within a denoiser call, preserving the original denoising update rule (Li et al., 2024; Fang et al., 2025). AsyncDiff (Chen et al., 2024b) combines model partitioning with asynchronous denoising, allowing different partitions to process different denoising steps in parallel by reusing hidden states from adjacent steps. Parallel-in-time samplers instead batch tentative states from one trajectory and handle their dependencies through fixed-point correction or prediction reuse (Shih et al., 2023; Wang et al., 2025a). More closely related, StreamDiffusion (Kodaira et al., 2025) pipelines staggered stream elements, while Armory (Bansal et al., 2026) batches independent robot-policy requests. Cross-Cycle Batching draws its batch from successive full-horizon plans of a single receding-horizon controller, without model partitioning or relaxing dependencies between each chunk's denoising updates.

3 Preliminaries

3.1 Flow-based action policies

Given an observation $\mathbf{o}$, a flow policy samples a full-horizon action chunk $\mathbf{a}$ from the conditional distribution $\pi(\mathbf{a} \mid \mathbf{o})$ (Black et al., 2025b). Sampling starts from a noise chunk $\mathbf{x}_0 = \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim p_{\text{noise}}$, and transforms it into $\mathbf{x}_1 = \mathbf{a}$. A scalar flow time $\tau \in [0, 1]$ tracks this transformation and is shared by all action positions in the chunk. Flow matching learns an observation-conditioned vector field $v_\theta(\mathbf{x}_\tau, \tau \mid \mathbf{o})$ that defines the transformation (Lipman et al., 2023):

$$\frac{\mathrm{d}\mathbf{x}_\tau}{\mathrm{d}\tau} = v_\theta(\mathbf{x}_\tau, \tau \mid \mathbf{o}), \qquad \mathbf{x}_0 = \boldsymbol{\epsilon}, \quad \mathbf{x}_1 = \mathbf{a}. \tag{1}$$

During inference, we approximate the flow ODE with K Euler updates on the time grid

$$0 = \tau_0 < \tau_1 < \cdots < \tau_K = 1, \qquad \Delta\tau_k = \tau_{k+1} - \tau_k > 0. \tag{2}$$

The initial state is $\mathbf{x}_{\tau_0} = \boldsymbol{\epsilon}$. At denoising step k, the policy updates the current chunk as

$$\mathbf{x}_{\tau_{k+1}} = \mathbf{x}_{\tau_k} + \Delta\tau_k v_\theta(\mathbf{x}_{\tau_k}, \tau_k \mid \mathbf{o}), \qquad k = 0, \ldots, K-1. \tag{3}$$

Each update requires one model call and depends on the preceding state. Serial sampling therefore takes K sequential calls to produce $\mathbf{a} = \mathbf{x}_{\tau_K} = \mathbf{x}_1$ before its near-term prefix can be executed.

3.2 Receding-horizon action chunking

Receding-horizon execution proceeds in control cycles (Chi et al., 2023; Black et al., 2025c). At cycle n, let $\mathbf{a}_n$ denote the completed H-step action chunk. Under serial sampling, it is generated

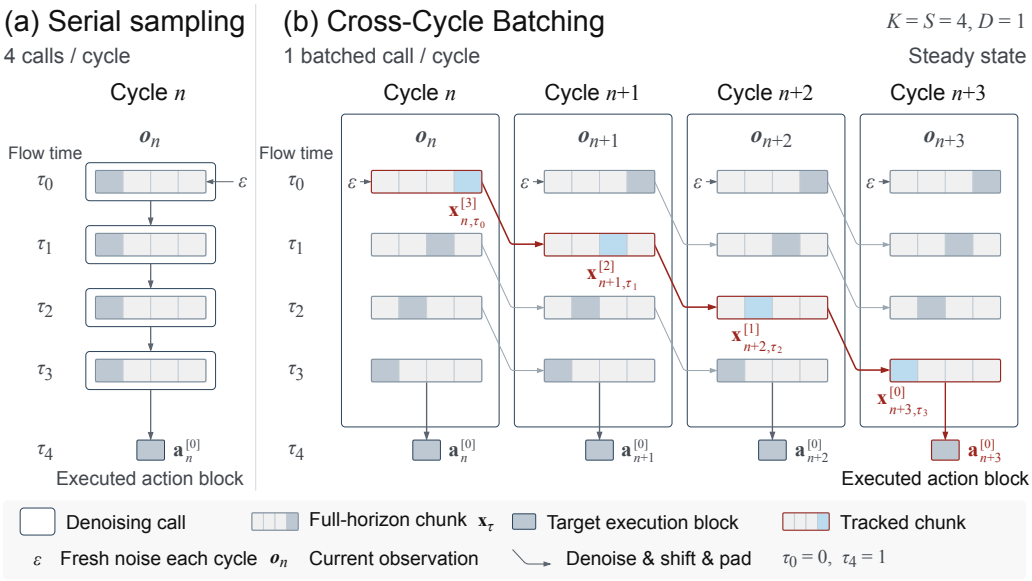


Figure 2: Serial sampling versus Cross-Cycle Batching ($K = S = 4$, $D = 1$). (a) Four sequential calls complete one chunk within a cycle. (b) One call updates four full-horizon chunks under the current observation, with input flow times ordered from τ_0 to τ_3. Red traces the chunk executed in cycle $n + 3$. For this trajectory, $\mathbf{x}^{[j]}_{m,\tau}$ denotes block j of the input chunk in cycle m at flow time τ. Shading marks the block eventually executed from each chunk. Cross-cycle arrows combine one denoising update with shift-and-pad (Figure 3). In steady state, fresh noise $\boldsymbol{\epsilon}$ is sampled independently at $\tau_0 = 0$ each cycle; the executed action block $\mathbf{a}_n^{[0]}$ exits at $\tau_4 = 1$ in cycle n.

from the current observation $\mathbf{o}_n$. We call H the *prediction horizon*. The controller executes the first $b \leq H$ actions and replans from an updated observation. We call b the *execution horizon*.

When $b < H$, the prediction windows at cycles n and $n+1$ overlap by $H-b$ action steps. Assuming b divides H, we partition each chunk into $S = H/b$ execution blocks of b actions:

$$\mathbf{a}_n = \left(\mathbf{a}_n^{[0]}, \mathbf{a}_n^{[1]}, \ldots, \mathbf{a}_n^{[S-1]}\right). \tag{4}$$

After executing $\mathbf{a}_n^{[0]}$, the prediction window advances by one block. For $j = 0, \ldots, S-2$, $\mathbf{a}_n^{[j+1]}$ and $\mathbf{a}_{n+1}^{[j]}$ correspond to the same future b-step interval. This overlap means that actions for later control cycles already appear in the current prediction window. Meanwhile, the dependency in equation 3 fixes the order of a chunk's denoising updates, but not the control cycles in which they are computed.

4 Cross-Cycle Batching

Cross-Cycle Batching uses receding-horizon overlap to distribute each chunk's K ordered denoising updates across control cycles. Updates within a chunk remain sequential, while ready updates from different chunks share one batched model call. Figure 2 illustrates the steady-state schedule.

4.1 Staggered Batched Sampling

Each cycle's execution block is covered by the prediction horizons of the preceding $S - 1$ cycles. Instead of performing all K denoising updates sequentially in the execution cycle, we distribute them evenly over S control cycles before executing the block. Assuming that S divides K, we set $D = K/S$. At every cycle, S chunks associated with S successive control cycles remain in flight, and each chunk receives D denoising updates.

Specifically, at the start of cycle n, these chunks are scheduled to supply one execution block per cycle from n through $n+S-1$. Ordered from the farthest target cycle $n+S-1$ to the current cycle

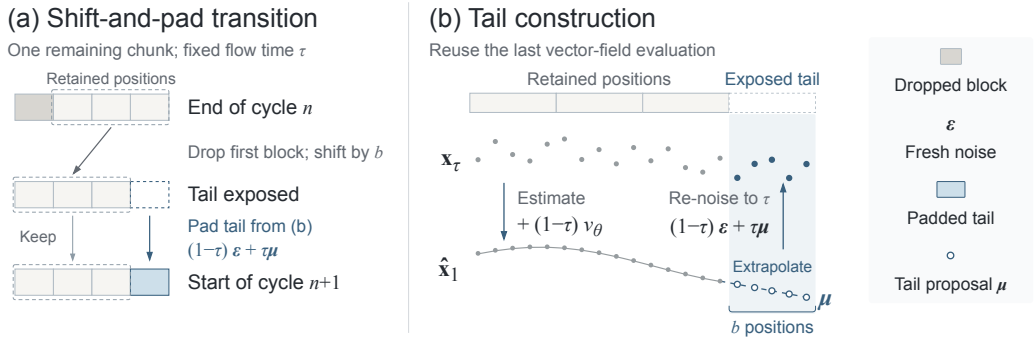


Figure 3: Transition of one remaining chunk between cycles n and $n+1$. (a) Its first block is dropped, the remaining positions shift forward by b, and the exposed tail is padded at the chunk's current flow time. (b) The retained positions at flow time τ (top lane) and their clean-endpoint estimate obtained from the last vector-field evaluation at no extra cost (bottom lane). A proposal $\boldsymbol{\mu}$ extrapolated from the clean estimate over the exposed b positions is re-noised to flow time τ and inserted as the tail (Section 4.3).

n, they have completed $0, D, \ldots, (S-1)D$ updates. We index the chunks by their current denoising stage $s \in \{0, \ldots, S-1\}$. The stage-s chunk has completed sD updates and lies at flow time τ_{sD}. We denote its current state by $\mathbf{x}^{(s)}_{\tau_{sD}}$. The stage-$s$ chunk is scheduled to supply the execution block for cycle $n + S - 1 - s$. Thus, stage $S-1$ will complete during the current cycle, while the fresh stage-zero noise chunk will complete in cycle $n + S - 1$.

During cycle n, every in-flight chunk receives its next D denoising updates. Let $d \in \{0, \ldots, D-1\}$ index these updates within the cycle. For the chunk that began the cycle at stage s, update d corresponds to denoising step $k = sD + d$:

$$\mathbf{x}^{(s)}_{\tau_{sD+d+1}} = \mathbf{x}^{(s)}_{\tau_{sD+d}} + \Delta\tau_{sD+d}\, v_\theta\left(\mathbf{x}^{(s)}_{\tau_{sD+d}}, \tau_{sD+d} \mid \mathbf{o}_n\right), \qquad s = 0, \ldots, S-1. \tag{5}$$

For any fixed chunk, the D updates remain sequential because update $d+1$ uses the state produced by update d. After update $D-1$, the stage-$(S-1)$ chunk completes as $\mathbf{a}_n = \mathbf{x}^{(S-1)}_{\tau_K}$. Its first block $\mathbf{a}^{[0]}_n$ supplies the next b actions to execute.

For a fixed d, all S updates use the latest observation $\mathbf{o}_n$, but each update depends only on its own chunk state. We stack the S states, pair each with its own flow time τ_{sD+d}, and compute the updates in one batched model call. Although the flow times differ across batch elements, each element remains a full-horizon chunk at one scalar flow time and matches the input form of serial sampling. In steady state, applying this batched update for $d = 0, \ldots, D-1$ gives D model calls of batch size S per control cycle. Serial sampling instead uses K model calls with batch size one per cycle. Each completed chunk uses the same sampling budget of K denoising steps as serial. Cross-Cycle Batching therefore requires no change to the policy weights, architecture, training procedure, or denoising update rule.

4.2 Receding-Horizon Chunk Transitions

At the end of cycle n, the stage-$(S-1)$ chunk has completed all K denoising updates. The controller takes its first block for execution, and we remove the completed chunk from the batch. Each of the other $S-1$ chunks has also received D updates and advanced by one denoising stage. We then add a fresh stage-zero noise chunk $\mathbf{x}^{(0)}_{\tau_0} = \boldsymbol{\epsilon}_{n+1}$, which is scheduled to supply the execution block for cycle $n+S$. Together, the remaining $S-1$ chunks and the new noise chunk form the staggered batch for cycle $n+1$.

The remaining chunks, however, still correspond to the prediction window of cycle n. After the selected block is executed, the prediction window advances by b actions and the controller receives the next observation $\mathbf{o}_{n+1}$. For each remaining chunk, we remove its first block and shift the other

$H-b$ action positions forward. This shift exposes a new b-action interval at the end of the prediction window, for which the chunk has no existing state. We fill this interval with a tail block at the chunk's current flow time (Figure 3). For a chunk at stage $s \in \{1, \ldots, S-1\}$, let $\boldsymbol{\mu}^{(s)}$ denote its tail proposal, and let $\boldsymbol{\epsilon}^{(s)}_{\text{tail}}$ be fresh tail noise. We initialize the tail as

$$\mathbf{x}^{(s)}_{\text{tail},\tau_{sD}} = (1-\tau_{sD})\boldsymbol{\epsilon}^{(s)}_{\text{tail}} + \tau_{sD}\boldsymbol{\mu}^{(s)}, \qquad s = 1, \ldots, S-1. \tag{6}$$

After this shift-and-pad operation, each remaining chunk again contains H action positions at one scalar flow time. The new stage-zero chunk is already full-horizon noise and requires no tail padding. The resulting batch is then ready for the D batched updates of cycle $n+1$, conditioned on $\mathbf{o}_{n+1}$.

4.3 Tail proposal construction

The newly exposed tail is unknown at inference time, but its content can affect vector-field predictions on retained positions. We therefore seek a proposal that keeps these predictions close to those obtained with the true tail. We extrapolate the chunk's predicted actions using the clean-endpoint estimate from its last vector-field evaluation, at no extra model cost:

$$\widehat{\mathbf{x}}_1 = \mathbf{x}_\tau + (1-\tau)v_\theta(\mathbf{x}_\tau, \tau \mid \mathbf{o}_n). \tag{7}$$

We drop the first block of this estimate, fit a degree-r polynomial to its last $r+1$ positions, and extrapolate over the next b positions to obtain $\boldsymbol{\mu}^{(s)}$. For $r=0$ this repeats the last action, and for $r=1$ it continues the motion at constant velocity. Equation 6 then re-noises the proposal to the chunk's current flow time (Figure 3b).

To evaluate a proposal, we construct two inputs that share the same observation, flow time, retained action positions, and noise. One uses the noised ground-truth tail and the other the proposed tail. Let $\mathbf{v}_{\text{real}}$ and $\mathbf{v}^{(r)}_{\text{pad}}$ denote their predicted vector fields on the block that will eventually be executed. We measure the relative change of this prediction, $\Delta^{(r)}(\tau) = \sqrt{\mathbb{E}\|\mathbf{v}^{(r)}_{\text{pad}} - \mathbf{v}_{\text{real}}\|_2^2 / \mathbb{E}\|\mathbf{v}_{\text{real}}\|_2^2}$, averaged over the padding flow times τ_{sD}, $s = 1, \ldots, S-1$.

Table 1 reports this change in percent for absolute and delta action representations. Appendix A provides the evaluation details. For $r \leq 1$, the padded tail changes the vector-field prediction on the executed block by less than 0.3% in both representations. The constructed tail therefore has little effect on executed-block vector-field predictions. The best order differs between the two representations. Absolute actions prefer $r=1$ and delta actions prefer $r=0$. With absolute actions, continuing the last velocity ($r=1$) follows the motion. A delta action is itself a velocity, so repeating the last delta ($r=0$) is the constant-velocity continuation. Both choices extend the motion at constant velocity in position space. This suggests choosing r by the action representation. Higher orders extrapolate acceleration from a noisy estimate and degrade in both representations.

Table 1: Relative change in the executed-block vector-field prediction caused by tail padding, for absolute and delta action representations.

	Vector-field change ↓	
Fitting order	Absolute	Delta
$r=0$	0.186%	0.283%
$r=1$	0.179%	0.293%
$r=2$	0.262%	0.432%
$r=3$	0.667%	1.542%

5 Analysis and Experiments

5.1 Latency Analysis

Cross-Cycle Batching reduces sequential calls by processing more chunks together. Its latency gain therefore depends on how call cost grows with batch size. Let $T(x)$ be the latency of one denoising call on $x = BH$ action tokens, batched as B separate chunks of fixed length H. The model, hardware, and inference settings are fixed. Cross-Cycle Batching sets $B = S$ and reduces calls per cycle from K to $D = K/S$. Ignoring scheduling overhead, its denoising speedup over serial is

$$G_{\text{denoise}}(B) \approx \frac{B \times T(H)}{T(BH)}. \tag{8}$$

The gain therefore depends on the reduction in average cost per token, $T(x)/x$. Once this cost plateaus, increasing S offers little further speedup. Its onset suggests a candidate value for SH.

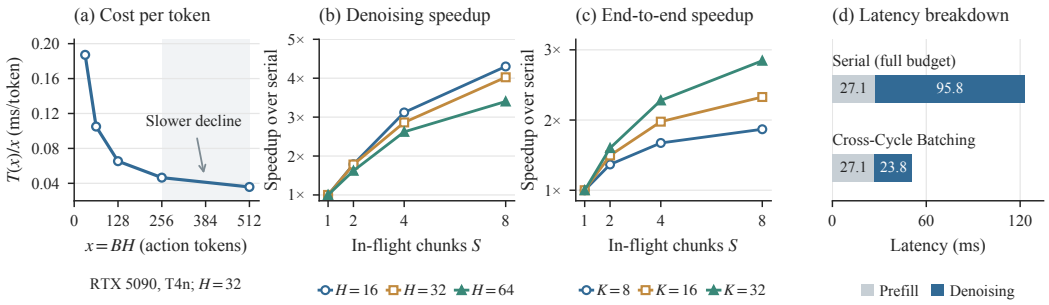


Figure 4: Latency scaling on Fast-WAM. (a) Average cost per action token at $H = 32$, with a slower decline at larger batches. (b) Predicted denoising gains from measured call costs at $K = 16$. (c) Measured speedup of prefill plus model calls at $H = 32$. (d) Prefill and denoising at $H = 32$, $K = 16$, and $S = 8$. Both sweeps use $S \in \{1, 2, 4, 8\}$ and $D = K/S$. Timings report medians over 200 repetitions.

We examine this scaling on Fast-WAM. At $H = 32$, the per-token cost decreases more slowly at larger token counts and continues to fall through 512 tokens (Figure 4a). To examine how chunk length affects the predicted gain, we fix $K = 16$ and vary $H \in \{16, 32, 64\}$. Figure 4b applies the speedup approximation to measured call costs. Longer chunks yield smaller predicted gains at the same S. Inference also includes observation prefill. Its cost is not reduced by batching, which limits the end-to-end gain. We therefore fix $H = 32$ and vary $K \in \{8, 16, 32\}$ to change the share of inference time spent on denoising. We time prefill followed by K serial or K/S batched model calls. At a fixed $S > 1$, larger K yields greater end-to-end speedup (Figure 4c). With more denoising work per cycle, prefill accounts for a smaller fraction of serial latency, so more of the denoising gain carries through. Figure 4d shows the component breakdown at $K = 16$ and $S = 8$. Batching reduces denoising time, leaving prefill as a larger share of the total.

5.2 Main Results

Simulation Evaluation. We first evaluate the success–latency trade-off of Cross-Cycle Batching on Fast-WAM (Yuan et al., 2026). Specifically, we compare it with full-budget serial, few-step serial, and mixed-progress sampling. Our mixed-progress baseline adapts the pipelined rolling-buffer pattern of prior work (Kim et al., 2024; Ruhe et al., 2024; Høeg et al., 2025). It advances action blocks at different flow times within a single chunk. All schedules are evaluated in a training-free setting using the same frozen checkpoint. Task initializations and execution settings are matched across schedules. We evaluate all schedules on all 50 RoboTwin 2.0 tasks under clean and randomized settings. Each configuration uses 100 episodes per task, totaling 5,000 evaluation trajectories per setting. The prediction horizon is $H = 32$ actions, with $b = 8$ actions executed per cycle. The accelerated schedules use $D \in \{1, 2, 3\}$ sequential calls per cycle. Full-budget serial uses $K = 4D$ denoising steps per chunk; few-step serial uses $K = D$. Detailed experimental settings are provided in Appendix B.1.

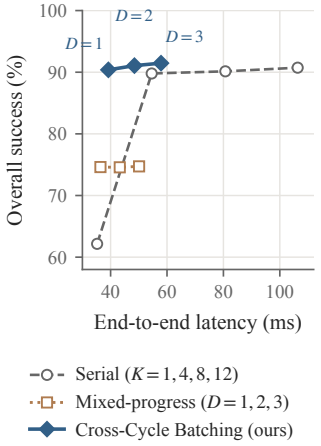


Figure 5: Success versus end-to-end latency on Fast-WAM.

Table 2 reports success rates and latency across sampling schedules and budgets. Figure 5 summarizes the trade-off between overall success and end-to-end latency. Cross-Cycle Batching maintains full-budget serial performance with fewer sequential calls and lower latency. Few-step serial and mixed-progress sampling have lower latency, but also lower success at the same call count. The success advantage holds across all tested budgets in both clean and randomized settings. We also observe slightly higher overall success than full-budget serial at every tested budget. This may reflect a benefit of refining plans with updated observations across cycles.

Table 2: Fast-WAM on all 50 RoboTwin 2.0 tasks. Cross-Cycle Batching achieves success rates comparable to full-budget serial with lower latency. K denotes the sampling budget; Calls and Batch denote sequential policy calls per cycle and chunks per controller per call. Parentheses report changes in overall success (percentage points) and latency speedups relative to the serial baseline in each group. End-to-end latency includes observation prefill and sampling.

Sampling configuration					Success (%)			Latency (ms) and speedup	
D	Schedule	K	Calls	Batch	Clean	Random	Overall	Denoising	End-to-end
	Serial (full budget)	4	4	1	89.6	90.0	89.8	26.2	54.6
1	Serial (few-step)	1	1	1	61.7	62.6	62.2 (−27.6)	6.5 (4.0×)	35.2 (1.5×)
	Mixed-progress	4	1	1	72.2	77.1	74.6 (−15.2)	6.8 (3.9×)	36.4 (1.5×)
	Cross-Cycle Batching (ours)	4	1	4	90.5	90.3	90.4 (+0.6)	9.0 (2.9×)	39.2 (1.4×)
	Serial (full budget)	8	8	1	90.3	90.0	90.1	52.4	80.7
2	Serial (few-step)	2	2	1	87.6	88.9	88.2 (−1.9)	13.0 (4.0×)	41.7 (1.9×)
	Mixed-progress	8	2	1	72.3	76.8	74.6 (−15.6)	13.7 (3.8×)	43.3 (1.9×)
	Cross-Cycle Batching (ours)	8	2	4	91.0	91.2	91.1 (+0.9)	17.8 (2.9×)	48.5 (1.7×)
	Serial (full budget)	12	12	1	91.0	90.5	90.7	78.7	106.3
3	Serial (few-step)	3	3	1	89.2	89.0	89.1 (−1.6)	19.6 (4.0×)	48.0 (2.2×)
	Mixed-progress	12	3	1	71.9	77.5	74.7 (−16.0)	20.5 (3.8×)	50.1 (2.1×)
	Cross-Cycle Batching (ours)	12	3	4	91.2	91.7	91.5 (+0.7)	27.5 (2.9×)	57.9 (1.8×)



(a) Produce placement (b) Block stacking (c) Bracket flipping (d) Cylinder insertion

Figure 6: Four real-robot tasks, with wrist views inset at the upper left.

Table 3: Real-robot evaluation with RTC. Cross-Cycle Batching lowers inference latency while maintaining comparable task success. Overall averages success across the four tasks. Parentheses report changes in overall success (percentage points) and latency speedups relative to Serial.

Schedule	Task success (%)					Latency (ms) and speedup	
	Placement	Stacking	Flipping	Insertion	Overall	Denoising	End-to-end
Serial	84	80	64	24	63	136.3	177.1
Cross-Cycle Batching	88	80	64	32	66 (+3)	58.2 (2.3×)	99.0 (1.8×)

Real-Robot Evaluation. We next ask whether the inference gains carry over to real-robot control while preserving task performance. We evaluate four manipulation tasks on a Franka arm: produce placement, block stacking, bracket flipping, and cylinder insertion (Figure 6). We use Fast-WAM as the base model and train for 30,000 steps using 1,000 demonstration trajectories, including 400 for produce placement and 200 for each of the other three tasks. We evaluate each method for 25 episodes per task.

For deployment, we integrate Cross-Cycle Batching with real-time chunking (RTC) (Black et al., 2025c). The policy predicts an action chunk with horizon $H = 64$ and executes $b = 16$ actions per control cycle, with actions executed at 30 Hz. Within this pipeline, the $2.3\times$ denoising speedup translates into a $1.8\times$ end-to-end inference speedup (Table 3). These gains come with comparable task success and a slightly higher average across the four tasks. Appendix B.2 provides the evaluation and profiling details.

Table 4: Comparison with Serial using the same checkpoint and sampling budget per chunk.

Model	Type	Params.	Benchmark	Schedule	Success (%)	Latency (ms) and speedup	
						Denoising	End-to-end
$\pi_{0.5}$	VLA	4B	LIBERO	Serial	97.6	38.8	87.9
				Cross-Cycle Batching	97.3 (−0.3)	12.4 (3.1×)	60.4 (1.5×)
LiLaWAM	WAM	0.5B	RoboTwin 2.0	Serial	88.8	16.5	20.6
				Cross-Cycle Batching	89.5 (+0.8)	6.0 (2.8×)	10.3 (2.0×)
WLA-0	WLA	3.4B	RoboTwin 2.0	Serial	89.8	131.0	160.6
				Cross-Cycle Batching	90.8 (+1.0)	44.3 (3.0×)	75.6 (2.1×)

5.3 Generality and Ablation Studies

Evaluation across Policy Models. To test whether Cross-Cycle Batching generalizes beyond Fast-WAM, we apply it to three additional policies. We evaluate $\pi_{0.5}$ (Black et al., 2025a) on LIBERO (Liu et al., 2023), and LiLaWAM (Yang et al., 2026a) and WLA-0 (Yang et al., 2026b) on RoboTwin 2.0. They represent vision-language-action (VLA), world-action (WAM), and world-language-action (WLA) policies, respectively. For each model, we compare serial sampling and Cross-Cycle Batching using the same frozen checkpoint and sampling budget per chunk. RoboTwin success averages the clean and randomized settings. Appendix B.4 provides the configurations and evaluation details.

The results in Table 4 show that Cross-Cycle Batching achieves success rates comparable to serial sampling while reducing inference latency on all three models. Together with the Fast-WAM results, these observations suggest that the benefits extend across the tested policy families and model sizes.

Tail Padding Ablation. We use the main Fast-WAM evaluation settings with $D = 3$ denoising updates per chunk per cycle. We compare polynomial fitting orders $r = 0, 1, 2$, following Section 4.3. All padded variants retain the full prediction horizon. The no-padding variant removes one execution block at each shift without refilling the tail, so its prediction horizon shrinks.

Table 5 shows higher success with all three padding variants than with no padding. Our earlier analysis of the absolute action representation used here showed the smallest vector-field change at $r = 1$ (Table 1). However, $r = 0$ has the highest observed overall success in this ablation. The success rates are close across fitting orders. Appendix B.3 provides the detailed evaluation protocol.

Table 5: Tail-padding ablation for Fast-WAM on RoboTwin 2.0. We observe higher success with padding than without padding. $r = 0$ has the highest overall success in these runs. Gray marks the default $r = 1$.

Tail setting	RoboTwin 2.0 success (%)		
	Clean	Random	Overall
No padding	89.3	90.5	89.9
$r = 0$	91.5	91.8	91.7
$r = 1$	91.2	91.7	91.5
$r = 2$	91.6	91.2	91.4

6 Conclusion, Limitation and Future Work

We presented Cross-Cycle Batching, a training-free schedule that turns overlap between control cycles into batch parallelism. It batches ready denoising updates to reduce sequential model calls while using the same per-chunk sampling budget as serial. Experiments across policy architectures and real-robot tasks show faster inference while maintaining task success. These results support temporal overlap as a source of parallelism within a single controller.

The method relies on receding-horizon control with overlapping prediction windows. Tighter integration with real-time chunking could better coordinate denoising, incoming observations, and action execution. Future work could also train policies for cross-cycle inference. Combining causal attention with variable-length action chunks could support shrinking horizons without tail padding. Such training could improve efficiency and task performance by teaching policies to refine partially denoised actions as new observations arrive.

AI USE STATEMENT

We used generative AI tools to assist with conceptual development, feedback on experimental design, data processing, and interpretation of results. Additional uses included literature search, translation, manuscript drafting and editing, development of analysis and plotting scripts, figure and table preparation, and LaTeX formatting. We reviewed and revised AI-assisted material during preparation, checking references against source publications and reported values against experimental records, and inspecting the resulting figures and tables. The authors take responsibility for the final text, claims, results, and artifacts in this work.

ETHICS STATEMENT

Our experiments evaluate object manipulation in simulation and under human supervision in a laboratory. Real-robot execution uses joint-impedance control and checks for abrupt joint-target changes, with trials manually terminated and scored. These safeguards and the trial-exclusion rules are described in Appendix B.2. The reported inference speedups do not establish safe operation beyond the evaluated conditions. Deployment in other settings requires validation of the policy and controller, together with appropriate safety mechanisms.

REPRODUCIBILITY STATEMENT

Section 4 specifies the sampling schedule, chunk transitions, and tail-proposal construction. Section 5 describes the evaluation settings and latency analysis. Appendix A details the paired vector-field diagnostic, including data selection and input construction. Appendix B records the sampling budgets, evaluation protocols, and latency profiling settings for the reported results. For real-robot experiments, Appendix B.2 additionally documents the training configuration, policy inputs and outputs, RTC integration, asynchronous execution, and trial protocol.

REFERENCES

Rohan Bansal, David He, Nadun Ranawaka Arachchige, Zhenyang Chen, Soobum Kim, Kexin Rong, and Danfei Xu. Action chunk scheduling for batched robot policy serving, 2026. URL https://arxiv.org/abs/2608.00337.

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 *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

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

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025c. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. doi: 10.52202/

079017-0759. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/2aee1c4159e48407d68fe16ae8e6e49e-Abstract-Conference.html.

Zhuoqun Chen, Xiu Yuan, Tongzhou Mu, and Hao Su. Responsive noise-relaying diffusion policy: Responsive and efficient visuomotor control. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=LLWJkR6gaI.

Zigeng Chen, Xinyin Ma, Gongfan Fang, Zhenxiong Tan, and Xinchao Wang. AsyncDiff: Parallelizing diffusion models by asynchronous denoising. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95170–95197, 2024b. doi: 10.52202/079017-3015. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad15848baa3932c0d2deabf0e11d1dcd-Abstract-Conference.html.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin C. M. Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://www.roboticsproceedings.org/rss19/p026.html.

Yufei Duan, Hang Yin, and Danica Kragic. Real-time iteration scheme for diffusion policy. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 11758–11764, 2025. doi: 10.1109/IROS60139.2025.11247391. URL https://arxiv.org/abs/2508.05396.

Jiarui Fang, Jinzhe Pan, Aoyu Li, Xibo Sun, and Jiannan Wang. PipeFusion: Patch-level pipeline parallelism for diffusion transformers inference. In *Advances in Neural Information Processing Systems*, volume 38, pp. 109089–109110, 2025. doi: 10.52202/085713-3289. URL https://papers.nips.cc/paper_files/paper/2025/hash/8da04a60948be713dc766f0c7e3a5b1f-Abstract-Conference.html.

Sigmund H. Høeg, Yilun Du, and Olav Egeland. Fast policy synthesis with variable noise diffusion models. In *IEEE International Conference on Robotics and Automation*, pp. 4821–4828, 2025. doi: 10.1109/ICRA55743.2025.11127858. URL https://arxiv.org/abs/2406.04806.

Kangye Ji, Yuan Meng, Hanyun Cui, Ye Li, Jianbo Zhou, Shengjia Hua, Lei Chen, and Zhi Wang. Block-wise adaptive caching for accelerating diffusion policy. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=c6ZWfQLOWD.

Kangye Ji, Yuan Meng, Jianbo Zhou, Ye Li, Chen Tang, and Zhi Wang. Test-time sparsity for extreme fast action diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9414–9423, 2026b. URL https://openaccess.thecvf.com/content/CVPR2026/html/Ji_Test-time_Sparsity_for_Extreme_Fast_Action_Diffusion_CVPR_2026_paper.html.

Jihwan Kim, Junoh Kang, Jinyoung Choi, and Bohyung Han. FIFO-Diffusion: Generating infinite videos from text without training. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2853. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/a397986e0f34d4b1f0b640686ceaeff7-Abstract-Conference.html.

Akio Kodaira, Chenfeng Xu, Toshiki Hazama, Takanori Yoshimoto, Kohei Ohno, Shogo Mitsuhori, Soichi Sugano, Hanying Cho, Zhijian Liu, Masayoshi Tomizuka, and Kurt Keutzer. StreamDiffusion: A pipeline-level solution for real-time interactive generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12371–12380, 2025. doi: 10.1109/ICCV51701.2025.01150. URL https://openaccess.thecvf.com/content/ICCV2025/html/Kodaira_StreamDiffusion_A_Pipeline-level_Solution_for_Real-Time_Interactive_Generation_ICCV_2025_paper.html.

Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Kai Li, and Song Han. DistriFusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7183–7193, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Li_DistriFusion_Distributed_Parallel_Inference_for_High-Resolution_Diffusion_Models_CVPR_2024_paper.html.

Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified video action model. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.074. URL https://www.roboticsproceedings.org/rss21/p074.html.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-Diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17535–17545, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Li_Q-Diffusion_Quantizing_Diffusion_Models_ICCV_2023_paper.html.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-0418. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/260a14acce2a89dad36adc8eefe7c59e-Abstract-Conference.html.

Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. FASTER: Rethinking real-time flow VLAs, 2026. URL https://arxiv.org/abs/2603.19199.

Sungjae Park and Shubham Tulsiani. $\pi\mathbf{R}^2$: Reactive real-time flow policies, 2026. URL https://arxiv.org/abs/2607.26055.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.071. URL https://www.roboticsproceedings.org/rss20/p071.html.

David Ruhe, Jonathan Heek, Tim Salimans, and Emiel Hoogeboom. Rolling diffusion models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42818–42835, 2024. URL https://proceedings.mlr.press/v235/ruhe24a.html.

Amirmojtaba Sabour, Sanja Fidler, and Karsten Kreis. Align your steps: Optimizing sampling schedules in diffusion models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42947–42975. PMLR, 2024. URL https://proceedings.mlr.press/v235/sabour24a.html.

Qiuhong Shen, Shihua Zhang, Yue Liao, Qi Li, Zhenxiong Tan, Shizun Wang, Shuicheng Yan, and Xinchao Wang. World action models: A survey, 2026. URL https://arxiv.org/abs/2606.20781.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0188. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/0d1986a61e30e5fa408c81216a616e20-Abstract-Conference.html.

Kunyun Wang, Bohan Li, Kai Yu, Minyi Guo, and Jieru Zhao. Communication-efficient diffusion denoising parallelization via reuse-then-predict mechanism. In *Advances in Neural Information Processing Systems*, volume 38, pp. 186183–186207, 2025a. doi: 10.52202/085713-5601. URL https://papers.nips.cc/paper_files/paper/2025/hash/f588cca6003651c4d4cb75b8554d55ad-Abstract-Conference.html.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63399–63416. PMLR, 2025b. URL https://proceedings.mlr.press/v267/wang25ba.html.

Fan Yang, Yuting Su, Xiaobo Wang, Yuncheng You, Fugui Fan, Yuting Wu, Minghui Wu, Chenxu Zhao, JiaHong Ning, and Peiguang Jing. LiLa-WAM: Lightweight latent reasoning world-action model for robotic manipulation, 2026a. URL https://arxiv.org/abs/2608.03701.

Yi Yang, Zhihong Liu, Siqi Kou, Yiyang Chen, Yanzhe Hu, Jianbo Zhou, Boyuan Zhao, Zhijie Wei, Xiao Xia, Xueqi Li, Pengfei Liu, and Zhijie Deng. World-language-action model for unified world modeling, language reasoning, and action synthesis, 2026b. URL https://arxiv.org/abs/2606.05979.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi Jim Fan, and Joel Jang. World action models are zero-shot policies, 2026. URL https://arxiv.org/abs/2602.15922.

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

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. FlowPolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14754–14762, 2025. doi: 10.1609/aaai.v39i14.33617. URL https://ojs.aaai.org/index.php/AAAI/article/view/33617.

Zihan Zhang, Richard Liu, Kfir Aberman, and Rana Hanocka. TEDi: Temporally-entangled diffusion for long-term motion synthesis. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. doi: 10.1145/3641519.3657515. URL https://doi.org/10.1145/3641519.3657515.