DON'T THROW AWAY THE SOLUTION: TRAINING-FREE ACCELERATION OF FLOW-BASED ROBOT POLICIES

Anonymous authors
Paper under double-blind review

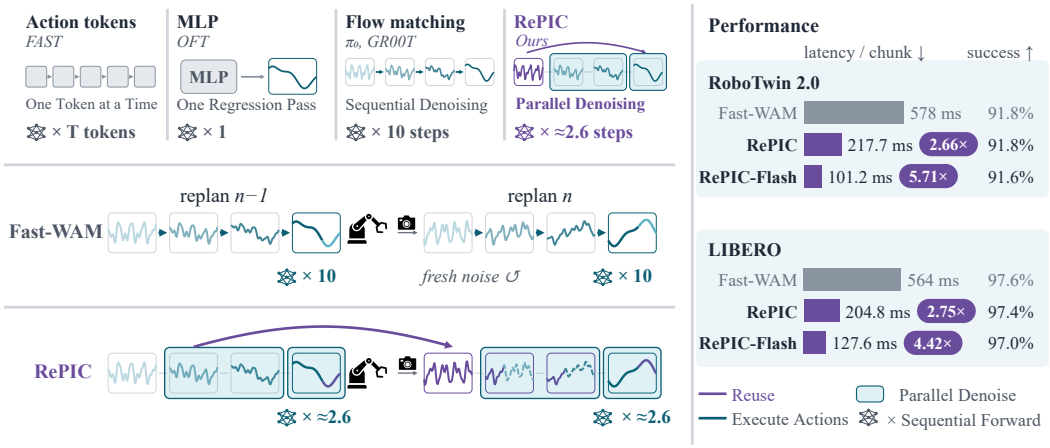


Figure 1: **Don't throw away the solution.** *Left:* RePIC combines trajectory reuse with parallel denoising, reducing sequential velocity network calls from 10 to 2.2–2.6 per chunk on Fast-WAM. *Right:* action-generation latency and task success of Fast-WAM and its RePIC variants on RoboTwin 2.0 and LIBERO. Latency excludes action execution and simulator stepping.

ABSTRACT

Standard samplers for flow-based robot policies generate each action chunk by sequential denoising and discard the solver trajectory at every replan, even though consecutive control cycles solve closely related problems. We introduce RePIC, a training-free sampler that turns this discarded trajectory into the initialization of the next solve. After an action prefix is executed, RePIC shifts the cached trajectory to the new horizon and re-aligns it to fresh noise with a closed-form correction that preserves the clean-action endpoint and needs no network evaluation. The pretrained velocity network then refines all transferred states under the new observation through batched fixed-point updates on the original integration grid, so an accurate initialization is converted into fewer sequential network calls. On Fast-WAM, RePIC reduces sequential calls per chunk from 10 to 2.2–2.6 and action-generation latency from 578 to 218 ms on RoboTwin 2.0 ($2.66\times$) and from 564 to 205 ms on LIBERO ($2.75\times$) at comparable success; with system optimizations, RePIC-Flash reaches $5.71\times$. The same sampler accelerates $\pi_{0.5}$ and GR00T N1.5 by $1.14\times$ and $1.68\times$ without retraining. Transferred drafts start $24\times$ closer to the serial solution on the carried action steps, and on a physical robot Fast-WAM with RePIC succeeds in 77.5% versus 66.3% for the original sampler. These results show that the numerical path of a previous solve is a reusable resource for fast robot control. Code will be made publicly available.

1 INTRODUCTION

Generalist robot policies increasingly generate actions with diffusion or flow models. Many recent vision-language-action models (VLAs) and world action models (WAMs) pair large multimodal backbones with generative action heads that produce entire action chunks through iterative denoising (Black et al., 2025a; Physical Intelligence et al., 2025; Ye et al., 2026; Yuan et al., 2026). This design

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.

captures multimodal and temporally coherent behavior, but its standard sampler is sequential: every chunk requires several evaluations of a large velocity network, one per integration step (Chi et al., 2023; Black et al., 2025a), and this latency sits directly inside the perception–action loop. In our evaluation, Fast-WAM spends 578 ms generating a single action chunk on RoboTwin 2.0 with an RTX 4090 (Tab. 1). Recent WAM accelerations remove test-time future imagination, shrink the action module, or decouple world reasoning from action prediction, yet they largely retain multi-step action denoising or shorten it only through additional distillation (Yuan et al., 2026; Ma et al., 2026; Cai et al., 2026).

Our Fast-WAM ablations reveal room to reduce this denoising budget: three Euler steps retain 96.60% success on LIBERO versus 97.55% with ten (Tab. 3). Yet fewer steps disproportionately hurt difficult RoboTwin 2.0 tasks (App. D), suggesting that not all refinement is dispensable. Beyond reducing computation *within a chunk*, we therefore examine computation repeated *across chunks*. Under receding-horizon control, a policy predicts H actions, executes a prefix of $R < H$ actions, and replans from a new observation (Zhao et al., 2023; Octo Model Team et al., 2024; Black et al., 2025a). Consecutive solves revisit $H - R$ action positions, yet standard sampling reconstructs their denoising paths from scratch (Fig. 1, left). This shared horizon suggests that the previous solve may remain informative despite the updated observation.

Prior work reuses partially denoised actions or previous predictions (Høeg et al., 2025; Chen et al., 2025; Duan et al., 2025; Li et al., 2026a), while C^{3}ache caches network residuals across chunks (Zhao et al., 2026b). Endpoint-only warm starts retain the final prediction but discard the intermediate states connecting noise to action. We ask: *can the entire denoising trajectory of the previous solve serve as solver state for the next one?* Our analyses support this premise. In a diagnostic RoboTwin 2.0 rollout, horizon- and noise-aligned drafts have $1/24$ the mean error of a cold-start linear draft on overlapping action positions, relative to the current serial solution (Fig. 4). On LIBERO, full-trajectory reuse needs 22.9% fewer sequential calls than endpoint-only reuse under the same parallel solver (Tab. 4), revealing useful information beyond the final prediction. Two obstacles remain: cached states must be aligned with the new horizon and initial noise, and serial Euler sampling cannot convert a better trajectory initialization into fewer calls.

We introduce RePIC, a training-free method addressing both obstacles. *Noise-aligned trajectory transfer* shifts cached states to the new horizon and aligns their noisy endpoint with the prescribed initial noise through a closed-form, noise-scaled correction. This map requires no network evaluation and preserves the mapped clean prediction and the path's deviation from endpoint interpolation. *Reuse-initialized parallel refinement* casts sampling as a fixed-point problem (Shih et al., 2023): the pretrained velocity network updates a window of denoising states in one batched call under the current observation. Residual-guided advancement accepts multiple integration steps per round when drafts are accurate, shortening the sequential path on the original integration grid. The transferred trajectory is an initialization to refine, not a committed action plan (Fig. 6). On RoboTwin 2.0, tail residuals guide the next replan's execution length without extra network evaluations.

We evaluate RePIC on Fast-WAM (Yuan et al., 2026) with RoboTwin 2.0 (Chen et al., 2026) and LIBERO (Liu et al., 2023). RePIC reduces sequential calls per chunk from 10 to 2.2–2.6 and accelerates action generation by $2.66\times$ and $2.75\times$ at comparable average success (91.8% vs. 91.8% and 97.35% vs. 97.55%). With system optimizations, RePIC-Flash reaches $5.71\times$ and $4.42\times$ (Fig. 1, right). The same sampler accelerates $\pi_{0.5}$ (Physical Intelligence et al., 2025) and GR00T N1.5 (NVIDIA et al., 2025) by $1.14\times$ and $1.68\times$. Across real-robot trials per method, Fast-WAM with RePIC achieves 77.5% success versus 66.3% for the original sampler. Our contributions are:

- **Analysis of cross-replan solver reuse.** We characterize cross-replan redundancy in Fast-WAM's denoising trajectories, showing that aligned intermediate states improve initialization over cold starts and reduce sequential calls beyond endpoint-only reuse.
- **Noise-aligned reuse and parallel refinement.** Closed-form trajectory transfer, batched fixed-point updates, and residual-guided advancement convert initialization quality into fewer sequential calls without changing the pretrained velocity field or integration grid.
- **Training-free acceleration across robot policies.** We demonstrate $2.66\times$ and $2.75\times$ action-generation speedups on Fast-WAM at comparable success, with further system-level gains, transfer to VLAs and real-robot deployment, without retraining or an draft model.

2 Related Work

Action policies in robot learning. Autoregressive VLAs predict discrete action tokens, while OpenVLA-OFT uses parallel decoding, action chunking, and continuous regression for efficient action prediction (Zitkovich et al., 2023; Kim et al., 2024; 2025). Diffusion and flow-based policies model distributions over continuous action trajectories (Chi et al., 2023; Octo Model Team et al., 2024; Black et al., 2025a; Physical Intelligence et al., 2025), and WAMs such as Motus, DreamZero, and LingBot-VA couple action prediction with learned dynamics through video-action modeling (Bi et al., 2026; Ye et al., 2026; Li et al., 2026b). These works improve policy representation or training; RePIC leaves the learned flow policy unchanged and targets its inference.

Efficient robot inference. Fast-WAM removes explicit future generation at test time, AHA-WAM decouples low-frequency world reasoning from high-frequency action prediction (Yuan et al., 2026; Cai et al., 2026), and Faster-WAM variants use lightweight action modules or sparse reuse of future representations (Ma et al., 2026; Zhao et al., 2026a). ImageWAM, WAM4D, and Enfold change how predictive world representations are produced or consumed, by replacing video generation with image editing, distilling geometric priors only during training, or folding world-model imagination into predictive representations (Zhang et al., 2026; Li et al., 2026e; Zeng et al., 2026). Real-time chunking overlaps prediction with execution to tolerate policy latency (Black et al., 2025b), and bidirectional decoding improves the consistency and reactivity of chunked execution through test-time sampling (Liu et al., 2025b). RePIC instead targets the sequential dependency inside the flow-based action sampler and is complementary to these methods.

Accelerating iterative sampling. Faster solvers and one- or few-step generators shorten the sampling trajectory (Song et al., 2021; Lu et al., 2022; Song et al., 2023; Prasad et al., 2024; Frans et al., 2025; Geng et al., 2025), caching methods reuse intermediate network computation (Ma et al., 2024; Liu et al., 2025a), and ParaDiGMS and ParaTAA parallelize denoising through fixed-point iterations, with PCM learning to accelerate their convergence (Shih et al., 2023; Tang et al., 2024; So et al., 2025). For robot control, Streaming Diffusion Policy, Falcon, RTI-DP, and STEP reuse partially denoised actions or previous predictions to warm-start inference (Høeg et al., 2025; Chen et al., 2025; Duan et al., 2025; Li et al., 2026a); C^{3}ache caches network residuals across action chunks, and FBFM uses execution feedback to correct subsequent flow prediction (Zhao et al., 2026b; Li et al., 2026d). In contrast, RePIC transfers the *complete denoising trajectory* across replans to initialize a parallel fixed-point solve.

3 Method

3.1 Overview

At replan n, a chunked policy receives condition c_n (e.g., images, language, and proprioception), predicts an action chunk $\mathbf{a}_n \in \mathbb{R}^{H\times d}$, and executes a prefix of R_n actions before replanning. R_n is fixed before the current solve: it is constant under fixed-horizon execution, or selected from the preceding replan's residual under optional residual-guided commitment (Sec. 3.4); the first replan uses a controller-specified length.

We consider a pretrained flow policy with a deterministic Euler sampler for a fixed condition and noise draw, with noise schedule $1 = \sigma_0 > \sigma_1 > \cdots > \sigma_S = 0$. Starting from Gaussian noise $\mathbf{x}_0 = \boldsymbol{\epsilon}_n$, conventional sampling applies

$$\mathbf{x}_{s+1} = \mathbf{x}_s + \Delta_s v_\theta(\mathbf{x}_s, \sigma_s; c_n), \qquad \Delta_s = \sigma_{s+1} - \sigma_s, \tag{1}$$

where $\mathbf{x}_S$ is the generated action chunk. Because each state depends on its predecessor, Eq. 1 requires S sequential network evaluations. RePIC shortens this path with two complementary mechanisms: *within-replan parallelism*, which rewrites the serial trajectory as a fixed-point problem whose states are evaluated in batches, and *cross-replan reuse*, which transfers the complete denoising trajectory of the previous replan as the initialization of the next solve (Figs. 2 and 3).

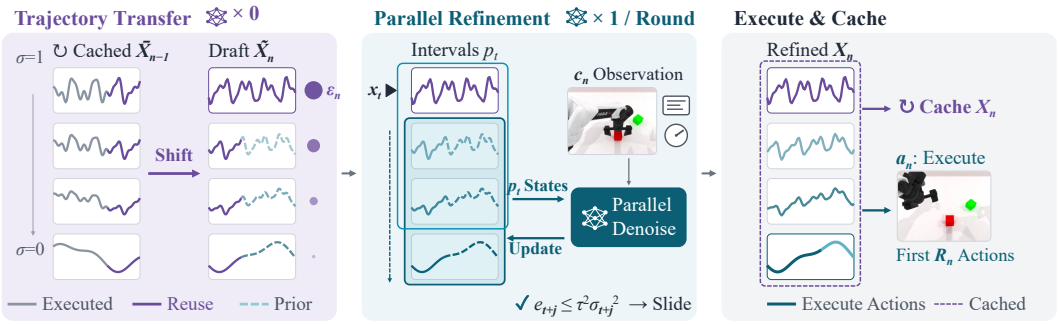


Figure 2: **One replan of RePIC.** Curves are action chunks, from noisy to clean states. *Left:* shift, fill, and align the cached trajectory to fresh noise without network calls. *Middle:* refine under c_n with one batched network call per round. *Right:* execute the R_n-action prefix and cache the trajectory.

3.2 Parallel Fixed-Point Refinement

Let $\mathbf{X} = (\mathbf{x}_0, \ldots, \mathbf{x}_S)$ denote the complete denoising trajectory. Unrolling Eq. 1 defines

$$[\mathcal{F}_{c_n}(\mathbf{X})]_j = \boldsymbol{\epsilon}_n + \sum_{i=0}^{j-1} \Delta_i v_\theta(\mathbf{x}_i, \sigma_i; c_n), \qquad j = 1, \ldots, S, \tag{2}$$

with $[\mathcal{F}_{c_n}(\mathbf{X})]_0 = \boldsymbol{\epsilon}_n$. The serial trajectory is a fixed point $\mathbf{X}^\star = \mathcal{F}_{c_n}(\mathbf{X}^\star)$, which refinement recovers at convergence (Shih et al., 2023) and approximates at finite tolerance.

Rather than evaluating Eq. 1 one state at a time, RePIC applies batched fixed-point (Picard) updates to a window of $p_t = \min(p, S - t)$ integration intervals. At refinement round k, the boundary state $\mathbf{x}_t$ is fixed, one batched call evaluates $\{v_\theta(\mathbf{x}_{t+i}^k, \sigma_{t+i}; c_n)\}_{i=0}^{p_t-1}$, and a prefix sum updates the window:

$$\mathbf{x}_{t+j}^{k+1} = \mathbf{x}_t + \sum_{i=t}^{t+j-1} \Delta_i v_\theta(\mathbf{x}_i^k, \sigma_i; c_n), \qquad j = 1, \ldots, p_t. \tag{3}$$

The shaded windows in Fig. 3 (middle) mark these updated states.

By default, window advancement uses the update residual on the action prefix scheduled for execution. With $\mathbf{x}_{s,1:R_n} \in \mathbb{R}^{R_n \times d}$ the first R_n action positions, the update residual of state $t + j$ is

$$e_{t+j}^{(n)} = \frac{1}{R_n d} \left\| \mathbf{x}_{t+j,1:R_n}^{k+1} - \mathbf{x}_{t+j,1:R_n}^{k} \right\|_F^2. \tag{4}$$

All H positions are updated, but only $1{:}R_n$ enters the default test; replacing R_n with H gives a full-chunk criterion. Prefix-wise acceptance is an advancement heuristic: it implies neither convergence of the unexecuted tail nor a bound on the distance to the serial trajectory; Tab. 5 compares both criteria. Following ParaDiGMS, the threshold scales with the squared noise level, $e_{t+j}^{(n)} \le \tau^2 \sigma_{t+j}^2$, and the window stride is

$$m = \min\left(\{ j \in \{1, \ldots, p_t - 1\} : e_{t+j}^{(n)} > \tau^2 \sigma_{t+j}^2 \} \cup \{p_t\} \right). \tag{5}$$

The next boundary is $\mathbf{x}_{t+m}^{k+1}$, and t advances by m. A more accurate initialization can therefore yield larger strides and a shorter sequential path: in the illustrative warm-started example of Fig. 3 (S=10, p=8), strides of 5, 3, and 2 complete the ten integration steps in three batched calls. At the first replan, one velocity evaluation at the noisy endpoint constructs a linear draft along the noise schedule, and states exposed by later windows are extrapolated from the last available velocity; once a trajectory is cached, cross-replan reuse replaces this cold start.

3.3 Noise-Aligned Trajectory Reuse

Endpoint-only reuse retains a clean action prediction but discards the intermediate states of the previous solve; RePIC transfers the full cached trajectory instead. Let $\bar{\mathbf{X}}_{n-1}$ denote the trajectory

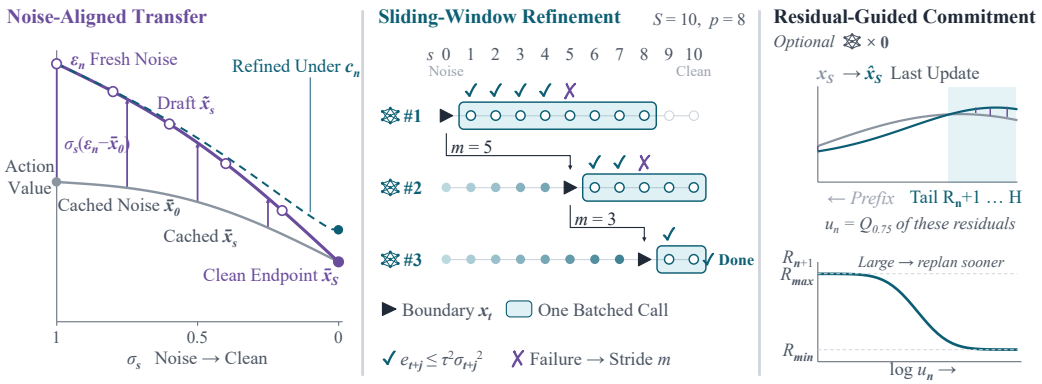


Figure 3: **Mechanisms of RePIC.** *Left:* noise alignment of one action position. *Middle:* a warm-started solve finishes in three batched calls (shaded: updated states). *Right:* residual-guided commitment maps tail residuals to the next execution length R_{n+1}.

cached after replan $n-1$, and let R_{n-1} be the prefix executed before replan n. For action position $h \in \{1, \dots, H\}$, define

$$q_{R_{n-1}}(h) = \begin{cases} h + R_{n-1}, & h \le H - R_{n-1}, \\ h, & h > H - R_{n-1}. \end{cases} \tag{6}$$

The overlapping suffix is shifted into the new prefix, and newly exposed positions take the same-position state of the previous trajectory as an initialization prior.

The shifted trajectory is tied to the previous noise realization, whereas the new replan starts from fresh noise $\boldsymbol{\epsilon}_n$. We therefore apply, at every noise level and action position, a correction that matches the new noisy endpoint and vanishes toward the clean endpoint:

$$\widetilde{\mathbf{x}}_{s,h} = \bar{\mathbf{x}}_{s,q_{R_{n-1}}(h)} + \sigma_s \left(\boldsymbol{\epsilon}_{n,h} - \bar{\mathbf{x}}_{0,q_{R_{n-1}}(h)} \right). \tag{7}$$

At $\sigma_0 = 1$, Eq. 7 recovers the fresh noise at every position; at $\sigma_S = 0$, the correction vanishes and the mapped clean actions are preserved; at intermediate levels, a noise-scaled offset adjusts the cached states (Fig. 3, left). Equivalently, the transfer keeps the cached trajectory's deviation from linear interpolation between its endpoints, which endpoint-only interpolation discards. The result is an initialization rather than a solution: all subsequent refinement evaluates the original velocity field under c_n, which corrects stale or inconsistent states, and the alignment itself needs only tensor operations.

3.4 Residual-Guided Action Commitment

When enabled, residual-guided commitment selects the execution length with a one-replan delay: R_n is fixed at replan n and defines the prefix in Eq. 4, while the final-update residuals on the unexecuted tail determine R_{n+1}. Let $\mathbf{x}_S$ and $\widehat{\mathbf{x}}_S$ denote the clean states before and after the final fixed-point update. For each tail position $h \in \{R_n{+}1, \dots, H\}$, we compute

$$r_{n,h} = \left(\frac{1}{d} \left\| \widehat{\mathbf{x}}_{S,h} - \mathbf{x}_{S,h} \right\|_2^2 \right)^{1/2}, \tag{8}$$

and the commitment score is their 0.75-quantile,

$$u_n = Q_{0.75}\left(\{ r_{n,h} \}_{h=R_n+1}^{H} \right). \tag{9}$$

A bounded decreasing map of $\log u_n$ selects $R_{n+1} \in \{R_{\min}, \dots, R_{\max}\}$, so smaller residuals select longer prefixes for the next replan (Fig. 3, right). The score reuses available iterates and needs no additional network evaluations.

Table 1: **Results on RoboTwin 2.0.** Success rates and action generation latency. Relative speed is normalized to Fast-WAM. Gray: training-free cache baseline; teal: RePIC.

Method	Size	Success rate (%) ↑			Latency ↓ (ms/chunk)	Rel. speed (×) ↑
		Clean	Random	Avg.		
LingBot-VA (Li et al., 2026c)	5.3B	92.9	91.6	92.2	9356.00	0.06
Motus (Bi et al., 2026)	8.0B	88.7	87.0	87.8	2177.00	0.27
Fast-WAM (Joint) (Yuan et al., 2026)	6.0B	90.8	90.3	90.6	1122.00	0.52
Enfold (Zeng et al., 2026)	3.0B	91.6	91.9	91.8	253.00	2.28
ImageWAM (Zhang et al., 2026)	8.6B	93.2	93.6	93.4	351.00	1.65
Faster-WAM (FC) (Zhao et al., 2026a)	∼6B	92.8	92.3	92.6	493.00	1.17
AHA-WAM (Cai et al., 2026)	7.23B	93.4	92.2	92.8	535.50	1.08
Faster-WAM (DoT) (Ma et al., 2026)	5.05B	89.7	88.6	89.2	–	–
Fast-WAM (Yuan et al., 2026)	6.0B	91.9	91.8	91.8	578.00	1.00
C^{3}ache (Zhao et al., 2026b)	6.0B	91.0	89.7	90.4	314.13†	1.84
RePIC	6.0B	91.9	91.7	91.8	**217.70**	**2.66**
RePIC-Flash	6.0B	92.4	90.7	91.6	**101.16**	**5.71**

† Latency and relative speed are inferred from the reported speedup and the reference latency.

3.5 Applicability and computational cost

The formulation applies directly to deterministic flow samplers whose states at different noise levels can be batched; other discretizations need their own fixed-point operator, and stochastic samplers need explicit treatment of their random increments. The condition c_n may come from a VLA, a WAM, or another conditional action model, and observation-dependent features can be computed once per replan and broadcast when the architecture allows. RePIC trades batched computation and memory for a shorter sequential path: a batched call over p_t states counts as one sequential call but p_t state-wise evaluations, and the cache stores $(S+1)Hd$ values. We therefore report wall-clock latency alongside sequential call counts.

4 Experiments

We evaluate RePIC on Fast-WAM (Yuan et al., 2026), which uses a Wan2.2-5B backbone (Wan Team et al., 2025), on RoboTwin 2.0 (Chen et al., 2026) and LIBERO (Liu et al., 2023), and on $\pi_{0.5}$ (Physical Intelligence et al., 2025) and GR00T N1.5 (NVIDIA et al., 2025) on LIBERO. On RoboTwin 2.0, Fast-WAM predicts $H = 32$ bimanual actions ($d = 14$) with $S = 10$ integration steps; the baseline executes $R = 24$ actions per replan, and RePIC uses residual-guided commitment with $R_n \in \{16, \dots, 24\}$ (Sec. 3.4). We evaluate all 50 tasks under clean and randomized conditions with 100 episodes each (10,000 per method) (Chen et al., 2026). On LIBERO, Fast-WAM covers all four suites with $d = 7$ and fixed $R = 10$ for all methods. All variants keep the pretrained weights, and RePIC-Flash adds implementation optimizations (App. C). Latency is the action-generation time per chunk on an NVIDIA RTX 4090 (a 48 GB variant), from available observation tensors to the output chunk, excluding observation acquisition, simulator stepping, and action execution. Relative speed is normalized to Fast-WAM for WAMs and to the original sampler for each VLA.

4.1 Main results

On RoboTwin 2.0, RePIC reduces Fast-WAM latency from 578.00 ms to 217.70 ms ($2.66\times$) at the same average success (91.8%; binomial standard error 0.3 points), and $2.25\times$ with fixed $R = 24$ (Tab. 5); on LIBERO, it reduces latency from 563.85 ms to 204.77 ms ($2.75\times$), with success changing from 97.55% to 97.35% (Tabs. 1 and 2). RePIC-Flash reaches $5.71\times$ and $4.42\times$, with average success 0.2 and 0.55 points below the respective baselines. On the VLAs, RePIC accelerates $\pi_{0.5}$ by $1.14\times$ ($96.05\% \to 95.75\%$) and GR00T N1.5 by $1.68\times$ ($86.90\% \to 88.15\%$), so the gain carries over to both backbones; its magnitude depends on the sampler's share of latency and on the cost of a batched call relative to a single one.

Table 2: **Results on LIBERO.** Success rates on all four suites; Avg. is their arithmetic mean. Relative speed is normalized to the corresponding VLA backbone or to Fast-WAM for WAMs. Gray: training-free cache baseline; teal: RePIC.

Method	Size	Success rate (%) ↑					Latency ↓ (ms/chunk)	Rel. speed (×) ↑
		Spatial	Object	Goal	Long	Avg.		
Vision-language-action models (VLA)								
SmolVLA (Shukor et al., 2025)	0.45B	90.0	96.0	92.0	71.0	87.25	239.00	–
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	∼3.35B	98.0	98.6	96.8	90.8	96.05	70.24	1.00
+ RePIC	∼3.35B	97.6	98.2	96.0	91.2	95.75	**61.35**	**1.14**
GR00T N1.5 (NVIDIA et al., 2025)	3.0B	89.0	91.4	88.8	78.4	86.90	120.62	1.00
+ RePIC	3.0B	90.0	93.2	90.4	79.0	88.15	**71.60**	**1.68**
World action models (WAM)								
LingBot-VA (Li et al., 2026c)	5.3B	98.5	99.6	97.2	98.5	98.45	8623.00	0.07
Fast-WAM (Joint) (Yuan et al., 2026)	6.0B	99.6	99.4	98.2	96.8	98.50	1146.00	0.49
Enfold (Zeng et al., 2026)	3.0B	97.0	100.0	96.8	97.4	97.80	268.00	2.10
ImageWAM (Zhang et al., 2026)	8.6B	97.0	99.4	98.2	96.8	97.85	373.00	1.51
Faster-WAM (FC) (Zhao et al., 2026a)	∼6B	98.8	100.0	98.4	97.2	98.60	488.00	1.16
Faster-WAM (DoT) (Ma et al., 2026)	5.05B	98.4	100.0	97.0	97.8	98.30	176.20†	3.20
Fast-WAM (Yuan et al., 2026)	6.0B	98.0	99.6	97.2	95.4	97.55	563.85	1.00
C^{3}ache (Zhao et al., 2026b)	6.0B	97.6	99.6	96.0	95.2	97.10	224.64†	2.51
RePIC	6.0B	97.4	99.6	96.8	95.6	97.35	**204.77**	**2.75**
RePIC-Flash	6.0B	97.4	99.4	96.4	94.8	97.00	**127.63**	**4.42**

† Latency and relative speed are inferred from the reported speedup and the reference latency.

Table 3: **Parallelism and reuse.** Baselines and parallel refinement.

Method	S	SR (%) ↑	Calls	Latency (ms) ↓
Baseline	10	97.55	10.00	563.85
Baseline	5	96.90	5.00	333.00
Baseline	3	96.60	3.00	236.30
Baseline	1	95.35	1.00	151.80
Parallel Only	10	97.10	4.00	298.10
RePIC	10	97.35	2.22	204.77

Table 4: **Trajectory initialization.** Endpoint and full-path reuse.

Draft	Shift	Align	SR (%) ↑	Calls	Latency (ms) ↓
No reuse	–	–	97.10	4.00	298.10
Endpoint only	✓	–	97.05	2.88	234.70
Full trajectory	✗	✗	96.35	3.03	245.80
Full trajectory	✗	✓	96.40	3.01	243.90
Full trajectory	✓	✗	97.15	4.87	331.40
RePIC	✓	✓	97.35	2.22	204.77

✓: enabled; ✗: disabled; –: not applicable. Latency and Calls are per chunk.

4.2 Ablation Studies

Unless noted, ablations use Fast-WAM on LIBERO. *Baseline* is Fast-WAM's sequential Euler sampler and *Parallel only* applies fixed-point refinement without reuse; S is the number of integration intervals, SR the average success rate, and *Calls* the number of sequential network invocations per chunk, where a batched call counts once and the cold-start draft call is included. Sensitivity to τ and p is reported in App. E.

Parallelism and reuse. At $S = 10$, parallel refinement alone reduces calls from 10 to 4 and latency from 563.85 ms to 298.10 ms (1.89×); adding reuse reduces calls to 2.22 and latency by a further 31.3% to 204.77 ms, with success changing from 97.10% to 97.35% (Tab. 3). RePIC is also faster than the three-step baseline while keeping all ten intervals and 0.75 points higher success; only the one-step baseline is faster, at 95.35% success. On RoboTwin 2.0, the three-step baseline is 8% faster than RePIC but loses 1.0 point overall and 25.5 points on the hardest task (App. D).

What to reuse. *Endpoint only* interpolates between fresh noise and the shifted clean actions, mirroring warm starts that reuse only the previous prediction (Duan et al., 2025; Li et al., 2026a); full-

Table 5: **Residual scope and execution length** on RoboTwin 2.0. Shaded: default RePIC.

Residual	Execution	SR (%) ↑	Calls	Latency (ms) ↓
Full 1:H	Fixed R=24	**92.2**	3.23	256.2
Full 1:H	Residual-guided	91.0	2.95	241.4
Prefix 1:R_n	Fixed R=24	92.1	3.19	257.0
Prefix 1:R_n	Residual-guided	91.8	**2.63**	**217.7**

Table 6: **Sampler and system optimization on LIBERO.**

Sampler	System opt.	SR (%) ↑	Latency (ms) ↓
Baseline	✗	97.55	563.85
Baseline	✓	97.35	204.70
RePIC	✗	97.35	204.77
RePIC-Flash	✓	97.00	**127.63**

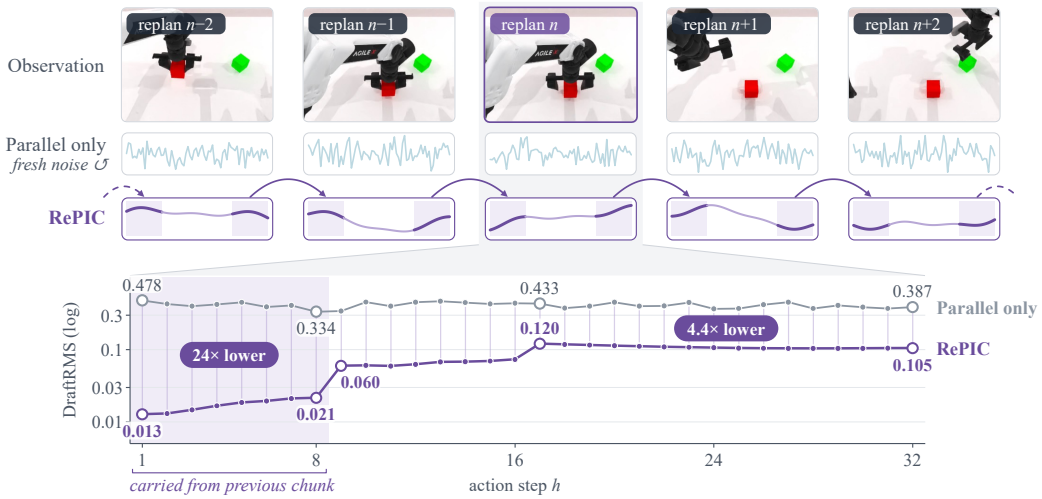


Figure 4: **Trajectory reuse lowers draft error.** *Top:* five consecutive RoboTwin 2.0 replans; RePIC carries each unexecuted suffix into the next chunk (shaded), whereas *Parallel only* starts from a linear draft of fresh noise. *Bottom:* DraftRMS per action step h (log scale), fixed $R = 24$.

trajectory variants toggle *Shift* (horizon mapping) and *Align* (noise alignment), and all rows reset the noise endpoint. Relative to endpoint-only reuse, the complete RePIC initialization reduces calls from 2.88 to 2.22 (22.9%) and latency from 234.70 ms to 204.77 ms (12.8%), with success from 97.05% to 97.35% (Tab. 4), supporting the reuse of intermediate solver states. Both components matter: shifting without alignment requires 4.87 calls (331.40 ms). Without shifting, the aligned and unaligned rows coincide: Fast-WAM draws $\boldsymbol{\epsilon}_n$ from a fixed seed at every replan, so the correction in Eq. 7 vanishes without a positional shift, and alignment corrects the mismatch $\boldsymbol{\epsilon}_{n,h} - \boldsymbol{\epsilon}_{n-1,h+R_{n-1}}$.

Residual scope and execution length. The unexecuted tail does not enter the default advancement test, yet it becomes the cached prior for the next replan. On RoboTwin 2.0 (Tab. 5), restricting the test to the prefix barely changes calls (3.23 vs. 3.19) or success (92.2% vs. 92.1%) at fixed $R = 24$. Residual-guided commitment lowers calls under both scopes but reduces success to 91.0% with full-chunk residuals; with prefix residuals, it gives the fewest calls (2.63) and 15.0% lower latency than the full-chunk, fixed-length setting, at 91.8% success, 0.3–0.4 points below the fixed-length settings.

System optimization. With CUDA Graph replay, the baseline takes 204.70 ms and RePIC-Flash 127.63 ms (1.60×; Tab. 6), so the sampler's gain persists on a compiled path, although it is smaller than with the standard backend (2.75×); RoboTwin 2.0 uses a TensorRT backend (App. C).

4.3 Why trajectory reuse helps

To see why reuse reduces calls, we measure how close the initialization is to the serial solution with $\mathrm{DraftRMS}(h)$, the distance $\|\widetilde{\mathbf{x}}_{s,h} - \mathbf{x}^{\star}_{s,h}\|_2/\sqrt{d}$ averaged over noise levels s, where $\widetilde{\mathbf{x}}$ is the draft and $\mathbf{X}^{\star}$ the serial Euler trajectory (the fixed point of Eq. 2) for the same c_n and $\boldsymbol{\epsilon}_n$; a closer draft yields smaller update residuals (Eq. 4) and thus larger strides (Eq. 5). On a RoboTwin 2.0 rollout with fixed $R = 24$ (Fig. 4), the $H - R = 8$ steps carried over from the previous chunk have a draft



Figure 5: **Real-world deployment.** *Left:* a rollout of *Stack Three Bowls* on an Agilex Piper arm. *Right:* success rates over 20 trials per task for Fast-WAM and Fast-WAM with RePIC.

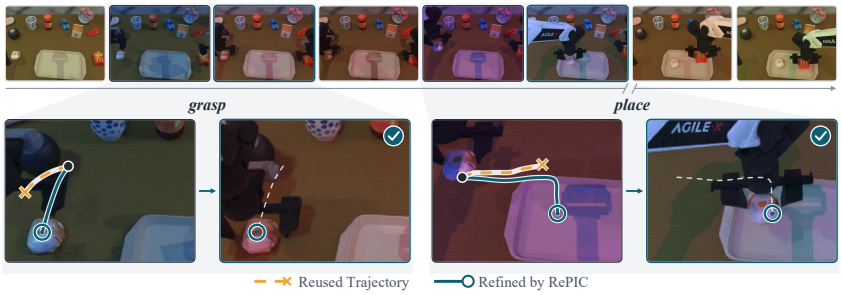


Figure 6: **Refinement corrects the reused trajectory (illustration).** Two replans of a RoboTwin 2.0 rollout, before (left) and after (right) execution: refinement (teal) redirects the reused trajectory (amber) toward the burger and down into the tray.

error of 0.013–0.021, 24× lower on average than the cold-start linear draft (0.33–0.48), and they open the next execution prefix on which the advancement test is evaluated. The remaining positions, initialized from same-position states, are coarser (0.06–0.12) but still 4.4× lower.

Refinement corrects reused trajectories. Reuse supplies only an initialization, since every refinement round evaluates the velocity field under the current condition c_n. Fig. 6 shows two replans of a `place_burger_fries` rollout in which refinement redirects the reused trajectory, toward the burger when grasping and down into the tray when placing, rather than committing to a stale plan.

4.4 Real-world deployment

We deploy Fast-WAM with and without RePIC on an Agilex Piper arm (same GPU), using the RoboTwin 2.0 checkpoint fine-tuned on 400 real demonstrations mixed with simulation data, on four tasks with 20 trials each (Fig. 5). Fast-WAM with RePIC succeeds in 62 of 80 trials (77.5%) versus 53 (66.3%) for the original sampler, with higher success on three of the four tasks. On hardware the control cycle is dominated by executing the R-action prefix, so faster generation shortens the pause at each replan rather than the task duration; at 20 trials per task the difference is within trial-to-trial variation, and the result establishes that the shorter sampling path holds up in closed loop.

5 Conclusion

We presented RePIC, a training-free sampler that couples noise-aligned trajectory reuse with parallel fixed-point refinement for flow-based robot policies. By transferring intermediate solver states across replans and correcting them under the current observation, RePIC shortens the sequential inference path on the integration grid, accelerating Fast-WAM by 2.66× and 2.75× at comparable success, transferring to $\pi_{0.5}$ and GR00T N1.5, and holding up on a real robot. It trades batched computation for latency and is evaluated under the standard receding-horizon protocol of each benchmark; abrupt observation changes are outside this scope. More broadly, a reusable resource in generative control is not just the previous prediction, but the numerical path that produced it.

AI USE STATEMENT

LLMs provided feedback on methodology and experimental design and presentation, assisted coding, and improved manuscript readability. They did not develop the method or generate ideas, hypotheses, claims, data, or results. All results derive from the authors' experiments. The authors reviewed all AI-assisted work and take full responsibility for the final text, claims, and artifacts.

REFERENCES

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A unified latent action world model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Kevin Black, Noah Brown, Danny Driess, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems*, 2025a.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, 2025b.

Jisong Cai, Long Ling, Shiwei Chu, Zhongshan Liu, Jiayue Kang, Zhixuan Liang, Wenjie Xu, Yinan Mao, Weinan Zhang, Xiaokang Yang, Ru Ying, Ran Zheng, and Yao Mu. AHA-WAM: Asynchronous horizon-adaptive world-action modeling with observation-guided context routing. arXiv preprint arXiv:2606.09811, 2026.

Haojun Chen, Minghao Liu, Chengdong Ma, Xiaojian Ma, Zailin Ma, Huimin Wu, Yuanpei Chen, Yifan Zhong, Mingzhi Wang, Qing Li, and Yaodong Yang. Falcon: Fast visuomotor policies via partial denoising. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8552–8573. PMLR, 2025.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *International Conference on Machine Learning*, 2026.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

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 (IROS)*, pp. 11758–11764. IEEE, 2025.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *International Conference on Learning Representations*, 2025.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *Advances in Neural Information Processing Systems*, 2025.

Sigmund H. Høeg, Yilun Du, and Olav Egeland. Fast policy synthesis with variable noise diffusion models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4821–4828. IEEE, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Robotics: Science and Systems*, 2025.

Jinhao Li, Yuxuan Cong, Yingqiao Wang, Hao Xia, Shan Huang, Yijia Zhang, Ningyi Xu, and Guohao Dai. STEP: Warm-started visuomotor policies with spatiotemporal consistency prediction. *arXiv preprint arXiv:2602.08245*, 2026a.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control. arXiv preprint arXiv:2601.21998, 2026b.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026c.

Peize Li, Ruimeng Zhang, Ru Zhang, Cong Huang, Kai Chen, and Shanghang Zhang. FBFM: A training-free asynchronous feedback mechanism for flow-matching in world-action models execution. *arXiv preprint arXiv:2607.29235*, 2026d.

Ying Li, Xiaobao Wei, Jiajun Cao, Hao Wang, Xiaowei Chi, Chengyu Bai, Qianpu Sun, Jiajun Li, Xiaojie Zhang, Peidong Jia, Jian Tang, Sirui Han, and Shanghang Zhang. WAM4D: Fast 4D world action model via spatial register tokens. arXiv preprint arXiv:2606.14048, 2026e.

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 (Datasets and Benchmarks Track)*, 2023.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025a.

Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via guided test-time sampling. In *International Conference on Learning Representations*, 2025b.

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*, 2022.

Liheng Ma, Rui Heng Yang, Zhanguang Zhang, Mateo Clemente, Ziwen Hu, Tongtong Cao, and Yingxue Zhang. Faster-WAM: Do world action models need deep action modules? arXiv preprint arXiv:2608.02365, 2026.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, et al. GR00T N1: An open foundation model for generalist humanoid robots. arXiv preprint arXiv:2503.14734, 2025.

Octo Model Team, Dibya Ghosh, Homer Walke, et al. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems*, 2024.

Physical Intelligence, Kevin Black, Noah Brown, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Conference on Robot Learning*, 2025.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. In *Robotics: Science and Systems*, 2024.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. In *Advances in Neural Information Processing Systems*, 2023.

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.

Junhyuk So, Jiwoong Shin, Chaeyeon Jang, and Eunhyeok Park. PCM: Picard consistency model for fast parallel sampling of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23313–23322, 2025.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning*, 2023.

Zhiwei Tang, Jiasheng Tang, Hao Luo, Fan Wang, and Tsung-Hui Chang. Accelerating parallel sampling of diffusion models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47800–47818. PMLR, 2024.

Wan Team, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, et al. Wan: Open and advanced large-scale video generative models. arXiv preprint arXiv:2503.20314, 2025.

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 Fan, and Joel Jang. World action models are zero-shot policies. arXiv preprint arXiv:2602.15922, 2026.

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.

Weili Zeng, Yitong Xing, Fulong Liu, Chengqun Yang, Antao Xiang, Feng Tian, Jingnan Gao, Jisong Cai, Xin Wang, Xiaomin Wu, Yao Mu, Xiaokang Yang, and Yichao Yan. Enfold: Folding world model imagination into predictive representations for ultra-efficient embodied control. arXiv preprint arXiv:2607.26657, 2026.

Yuyang Zhang, Wenyao Zhang, Zekun Qi, He Zhang, Haitao Lin, Jingbo Zhang, Yao Mu, Xiaokang Yang, Wenjun Zeng, and Xin Jin. ImageWAM: Do world action models really need video generation, or just image editing? arXiv preprint arXiv:2606.19531, 2026.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems*, 2023.

Weiheng Zhao, Haoyi Jiang, Xin Shi, Liu Liu, Fan Huang, Zhizhong Su, Wei Sui, and Xinggang Wang. Faster-WAM: Efficient inference-time future conditioning for robust world action models. arXiv preprint arXiv:2608.04404, 2026a.

Weisen Zhao, Lam Nguyen, Zhicong Lu, and Yuzhang Shang. C^{3}ache: Accelerating world action models with cross inference chunk cache. arXiv preprint arXiv:2606.08962, 2026b.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183, 2023.

A REPIC ALGORITHM

Algorithm 1 lists one replan of RePIC. The pretrained velocity network v_θ is never modified, and the execution length R_n is fixed before the solve begins. At the start of an episode the cache is empty and R_n is the controller-specified length $R_{\max}$. We write $\mathbf{x}'_{t+j}$ for the states produced by one window update (Eq. 3). The next replan receives $(\mathbf{X}_n, R_n)$ as its cache and R_{n+1} as its execution

Algorithm 1: RePIC, one replan

Input : condition c_n, noise $\boldsymbol{\epsilon}_n$, execution length R_n; optional cache $(\bar{\mathbf{X}}_{n-1}, R_{n-1})$; $R_{\max}$, $R_{\min}$, S, p, τ, μ, ς
Output : action chunk $\mathbf{a}_n = \mathbf{x}_S$, cache $(\mathbf{X}_n, R_n)$, next length R_{n+1}

if *a cache* $(\bar{\mathbf{X}}_{n-1}, R_{n-1})$ *is available* **then**
  define $q_{R_{n-1}}$ by Eq. 6
  set $\widetilde{\mathbf{x}}_{s,h}$ by Eq. 7 for all s, h
  $\mathbf{x}_{s,h} \leftarrow \widetilde{\mathbf{x}}_{s,h}$ for all s, h
else
  $\mathbf{x}_0 \leftarrow \boldsymbol{\epsilon}_n$
$t \leftarrow 0$
while $t < S$ **do**
  $p_t \leftarrow \min(p, S - t)$
  if *any of* $\mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+p_t}$ *is unfilled* **then**
    if *no velocity has been accepted yet* **then** evaluate $v_\theta(\mathbf{x}_t, \sigma_t; c_n)$
    else reuse the last available velocity
    linear-fill only the unfilled states
  evaluate $\{v_\theta(\mathbf{x}_{t+i}, \sigma_{t+i}; c_n)\}_{i=0}^{p_t-1}$ in one batched call
  $\mathbf{x}'_{t+j} \leftarrow \mathbf{x}_t + \sum_{i=t}^{t+j-1} \Delta_i \, v_\theta(\mathbf{x}_i, \sigma_i; c_n)$ for $j = 1, \ldots, p_t$ `// Eq. 3; x_t fixed`
  if $t + p_t = S$ **then**
    $\widehat{\mathbf{x}}_S \leftarrow \mathbf{x}'_S$
    $r_{n,h}$ by Eq. 8 for $h = R_n{+}1, \ldots, H$
    $u_n \leftarrow Q_{0.75}\big(\{r_{n,h}\}_{h=R_n+1}^{H}\big)$
  $m \leftarrow p_t$
  for $j = 1$ **to** $p_t - 1$ **do**
    $e^{(n)}_{t+j}$ by Eq. 4 with $\mathbf{x}^{k+1}_{t+j} \leftarrow \mathbf{x}'_{t+j}$ and $\mathbf{x}^{k}_{t+j} \leftarrow \mathbf{x}_{t+j}$
    if $e^{(n)}_{t+j} > \tau^2 \sigma^2_{t+j}$ **then**
      $m \leftarrow j$; **break**
  $\mathbf{x}_{t+1:t+p_t} \leftarrow \mathbf{x}'_{t+1:t+p_t}$
  $t \leftarrow t + m$ `// Eq. 5`
execute the first R_n actions of $\mathbf{a}_n = \mathbf{x}_S$
cache $(\mathbf{X}_n, R_n) \leftarrow \big((\mathbf{x}_0, \ldots, \mathbf{x}_S), R_n\big)$
if $R_{\max} > R_{\min}$ **then**
  $z \leftarrow (\log u_n - \mu)/\varsigma$
  $R_{n+1} \leftarrow \mathrm{round}\big(R_{\max} - (R_{\max} - R_{\min})\,\Phi(z)\big)$
else
  $R_{n+1} \leftarrow R_{\max}$

length, so the shift of Eq. 6 always equals the prefix that was just executed. Two residuals appear in the algorithm, and they are not interchangeable: the stride test (Eq. 4) is evaluated on the execution prefix $1{:}R_n$ of every window state, whereas the commitment score u_n uses only the final update of the clean endpoint on the unexecuted tail $h = R_n{+}1, \ldots, H$. Φ is the standard normal CDF, so R_{n+1} is a bounded, decreasing function of $\log u_n$; u_n is clamped away from zero before the logarithm. We set $R_{\min} = \lceil H/2 \rceil$. Commitment is disabled whenever $R_{\max} \le R_{\min}$, in which case $R_n = R_{\max}$ at every replan; otherwise $R_{\max}$ is only the upper bound of R_{n+1}.

B Evaluation Details

This appendix collects the evaluation hyperparameters and protocol details that the main text does not state. The hardware, the latency window (from available observation tensors to the output action chunk, excluding observation acquisition, simulator stepping, and action execution), and the definition of relative speed are given in Sec. 4. All RePIC variants keep the pretrained policy weights, and RePIC-Flash runs the same sampler as the corresponding RePIC row on an optimized backend (Appendix C).

Shared sampler. All methods use deterministic Euler sampling. The Fast-WAM noise schedule follows the original configuration: u is spaced uniformly from 1 to 0 over $S{+}1$ points and

$$\sigma = \frac{\text{shift} \cdot u}{1 + (\text{shift} - 1)u}, \qquad \text{shift} = 5,$$

with no additional σ-shift and no stochastic sampling. When no cache is available (the first replan of an episode), a single velocity-network evaluation at the noisy endpoint constructs a linear draft along the schedule; every later replan starts from noise-aligned full-trajectory reuse. In all main-table evaluations the window stride compares the residual on the current execution prefix $1{:}R_n$ with $\tau^2\sigma^2$ (Eq. 5); Tab. 5 ablates the full-chunk criterion. All residuals are computed in the normalized action space of each policy. Environments are stepped one at a time (environment batch size 1); the batch dimension of a network call is the window of p_t denoising states, padded to $B{=}10$ on the RoboTwin 2.0 Flash backend (Appendix C.3).

Fast-WAM. We use the unconditional pretrained checkpoint of each benchmark together with its normalization statistics; future video is not generated at test time. Tab. 7 lists the default settings of all Fast-WAM experiments. For the optimized backends, latency excludes the first episode of each task, which absorbs CUDA Graph capture and TensorRT engine build; the standard backend has no such one-time cost.

Table 7: **Fast-WAM evaluation settings.** Defaults of all Fast-WAM experiments; an ablation that varies one of them (e.g., S in Tabs. 3 and 8) states the change in its table. The commitment parameters are given in the text.

Symbol	Meaning	RoboTwin 2.0	LIBERO
H	action-chunk length: actions predicted per replan	32	32
d	action dimension of one step	14 (bimanual)	7
S	integration intervals of the noise schedule (sequential steps of the serial sampler)	10	10
$R_{\max}$	maximum execution length: actions executed per replan by the baseline, and by RePIC at the first replan	24	10
p	window size: maximum number of denoising states refined in one batched call	10	10
τ	stride tolerance: a window state is accepted when its prefix residual satisfies $e \le \tau^2\sigma^2$ (Eq. 5)	0.05	0.05
env. seed	simulator seed that fixes the scene initialization; identical for all methods	per-episode table	42
noise seed	seed of the initial noise $\boldsymbol{\epsilon}_n$, redrawn from the same seed at every replan (Fast-WAM default)	42	42

Execution length. The serial Fast-WAM baseline executes $R{=}R_{\max}$ actions at every replan on both benchmarks. Residual-guided commitment is enabled for the main RePIC and RePIC-Flash results on RoboTwin 2.0 with $R_{\min}{=}\lceil H/2 \rceil{=}16$, $R_{\max}{=}24$, $\mu{=}{-}3.5$, and $\varsigma{=}0.5$, so $R_{n+1} \in \{16, \dots, 24\}$ and the first replan uses $R_{\max}$; Tab. 5 reports the fixed-length alternative. On LIBERO, $R_{\max}{=}10 \le R_{\min}{=}16$, so commitment is disabled and $R_n{=}10$ at every replan for all methods.

$\pi_{0.5}$. We use the official JAX $\pi_{0.5}$ LIBERO checkpoint with H=10, S=10, $R_{\max}$=5, p=8, and τ=0.1, with the prefix criterion and noise-aligned full-trajectory reuse; the seed is 7. Because $R_{\max} \leq R_{\min}$=5, commitment is disabled and R_n=5 at every replan.

GR00T N1.5. We use a separate LIBERO fine-tune for each of the four suites with H=16, S=4, $R_{\max}$=1, p=4, and τ=0.1, with the prefix criterion; the environment seed is 0. Because $R_{\max} \leq R_{\min}$=8, commitment is disabled and a single action is executed per replan.

Other methods in the tables. For Enfold, ImageWAM, Faster-WAM (FC), and AHA-WAM, success rates are taken from the original papers and latencies were measured by us on the same RTX 4090 with the same latency window. C^{3}ache and Faster-WAM (DoT) are not open-sourced: their success rates are taken from the original papers, and their latencies are inferred from the speedups over Fast-WAM reported there, applied to the Fast-WAM reference latency in our tables (marked † in Tabs. 1 and 2). The DoT paper reports its speedup (3.2×) only on LIBERO and no RoboTwin latency, so that entry is left blank.

C System Optimizations for RePIC-Flash

RePIC-Flash keeps the pretrained velocity network, the noise schedule, the tolerance, the trajectory reuse, and the update rule of RePIC, and changes only how the action expert is executed. Video tokens and text context are computed once per replan; the refinement loop then evaluates the action expert repeatedly on those features, with a batch size equal to the current window. Flash caches the quantities that are constant within a replan and runs the action-expert calls on a compiled path: CUDA Graph replay on LIBERO, and a TensorRT FP16 core with CUDA Graph replay on RoboTwin 2.0. Both paths perturb the numerics of the velocity evaluation, so residual-guided strides and success differ slightly from the standard backend (e.g., 3.08 vs. 2.63 calls on RoboTwin 2.0, Tab. 10).

C.1 Feature reuse within a replan

Fast-WAM's Mixture-of-Transformers block attends jointly over video and action tokens. The video branch does not depend on the current denoising state, so we run the video expert once per replan and store its per-layer keys and values. Every subsequent action-expert evaluation concatenates this cache with the current action keys and values and attends only from the action queries.

The joint attention mask and the flow-matching schedule depend only on the sequence lengths and the number of integration steps, so they are reused across replans of the same shape; prompt embeddings for a fixed instruction are computed once per episode. Both backends use these caches, which do not change the numerical trajectory.

C.2 CUDA Graph replay

Each refinement round launches one batched action-expert call whose batch size equals the current window. After a stride m the next window may have any length in $\{1, \ldots, p\}$, whereas a CUDA Graph has a fixed shape. On LIBERO (no TensorRT), the graph key includes the window batch size, so we capture one graph per feasible length $1, \ldots, \min(p, S)$, and the serial baseline replays its own b=1 graph. On RoboTwin 2.0, every request is padded to B=10 and a single graph serves all windows (Appendix C.3).

Capture uses graph-owned buffers for the action latents, timesteps, text context, attention mask, and video keys and values, and the current tensors are copied into these buffers before every replay. Copying is necessary because the video cache is constant within a replan but changes at the next observation: pointer identity is unreliable under allocator recycling, and keeping one graph per replan would exhaust memory. Each graph is captured once, after a short eager warm-up, and capture time is excluded from the reported latency. If capture fails, the call falls back to the eager path with the same caches.

C.3 TensorRT action core

The optimized RoboTwin 2.0 backend replaces the body of the action Transformer with a TensorRT engine. Only `pre_dit` remains in PyTorch; the residual blocks and the output head are compiled into the engine. Complex-valued rotary embeddings and the original scaled-dot-product attention do not lower reliably, so the core uses real $\sin$ / $\cos$ RoPE and explicit masked attention with the same weights. The host model stays in BF16 and the core holds an FP16 copy of the weights.

We build a single static-shape engine with batch capacity $B{=}10$ for our $S{=}10$, $p{=}10$ setting. A request of batch $b \leq 10$ is padded to B by repeating its first row, and the first b outputs are kept, so one engine and one CUDA Graph serve every window. The optimized serial baseline on RoboTwin 2.0 runs on the same padded engine with $b{=}1$, so its 405.7 ms (Tab. 10) is an upper bound on the latency of an optimized serial sampler; the LIBERO comparison, whose baseline is not padded, is the cleaner estimate of the sampler gain on a compiled path ($1.60\times$). The engine is built once per process, and its build time is excluded from the latency. TensorRT is used only together with CUDA Graph replay: the graph records `pre_dit`, the engine, and the write-back.

C.4 What is and is not included in the Flash latency

Flash is timed over the same action-generation interval as RePIC and the serial baseline: from available observation tensors to the returned action chunk, with CUDA synchronization. The interval includes the cached video prefill, the refinement rounds on the compiled path, and action decoding; it excludes observation capture, simulator stepping, action execution, graph capture, and TensorRT compilation. Flash changes neither the checkpoint, nor the number of integration steps, nor the tolerance. Running the serial sampler on the same backend separates the sampler gain from the implementation gain (Tabs. 6 and 10).

D RoboTwin Ablations

Sec. 4.2 reports the parallelism, initialization, and backend comparisons on LIBERO; this appendix repeats the same three comparisons for Fast-WAM on RoboTwin 2.0. Tabs. 8, 9, and 10 are the counterparts of Tabs. 3, 4, and 6, with the same row names. Success rate is the average over all 50 tasks and both conditions; latency and calls are per chunk under the protocol of Sec. 4. The RePIC rows use the main-table configuration (residual-guided commitment, $R_n \in \{16, \ldots, 24\}$), and the serial baselines execute $R{=}24$.

Table 8: **Parallelism and reuse.** Counterpart of Tab. 3 on RoboTwin 2.0.

Method	S	SR (%) ↑	Calls	Latency (ms) ↓
Baseline	10	91.8	10.00	578.0
Baseline	5	91.0	5.00	300.6
Baseline	3	90.8	3.00	200.2
Parallel only	10	91.9	4.00	341.2
RePIC	10	91.8	2.63	217.7

Table 9: **Trajectory initialization.** Counterpart of Tab. 4 on RoboTwin 2.0.

Draft	Shift	Align	SR (%) ↑	Calls	Latency (ms) ↓
No reuse	–	–	91.9	4.00	341.2
Endpoint only	✓	–	90.9	3.19	233.3
Full trajectory	✗	✗	91.6	3.36	233.4
Full trajectory	✗	✓	91.6	3.18	290.9
Full trajectory	✓	✗	91.6	4.73	295.9
RePIC	✓	✓	91.8	2.63	217.7

✓: enabled; ✗: disabled; –: not applicable. Latency and Calls are per chunk.

Parallelism and reuse. At $S = 10$, parallel refinement alone reduces latency from 578.0 ms to 341.2 ms and calls from 10 to 4.00 at 91.9% success; adding reuse reduces calls to 2.63 and latency to 217.7 ms at 91.8% (Tab. 8). Shortening the schedule instead gives 300.6 ms at $S = 5$ (91.0%) and 200.2 ms at $S = 3$ (90.8%): the three-step baseline is 8% faster than RePIC but loses 1.0 point overall, and the loss concentrates on difficult tasks. On `open_microwave`, the task with the lowest success under the original sampler, RePIC, $S = 5$, and $S = 3$ reach 50.5%, 33.5%, and 25.0%.

What to reuse. Noise-aligned full-trajectory reuse reaches 2.63 calls and 217.7 ms at 91.8% success, against 3.19 calls and 233.3 ms at 90.9% for endpoint-only reuse (Tab. 9). Both components of

the transfer matter: shift without alignment needs 4.73 calls (295.9 ms) and alignment without shift 3.18 calls (290.9 ms), and full-trajectory reuse with neither operation needs 3.36 calls (233.4 ms); all three retain 91.6% success.

Table 10: **Sampler and system optimization (RoboTwin 2.0).** Optimized: TensorRT FP16 core with CUDA Graph replay, both samplers padded to B=10 (Appendix C). Relative speed uses the standard baseline.

Sampler	Backend	SR (%) ↑	Calls	Latency (ms) ↓	Rel. speed (×) ↑
Baseline	Standard	91.8	10.00	578.0	1.00
Baseline	Optimized	92.0	10.00	405.7	1.42
RePIC	Standard	91.8	2.63	217.7	2.66
RePIC-Flash	Optimized	91.6	3.08	101.2	5.71

System optimization. With the standard backend, RePIC reduces latency from 578.0 ms to 217.7 ms (2.66×) at 2.63 calls and unchanged 91.8% success (Tab. 10). The optimized baseline, padded to B=10, takes 405.7 ms (1.42×) at 92.0%; RePIC-Flash takes 101.2 ms (5.71×) at 3.08 calls and 91.6%, where the FP16 core changes the strides relative to RePIC (2.63 calls).

E Sensitivity to the Tolerance and the Window Size

The sampler of RePIC has two hyperparameters: the stride tolerance τ (Eq. 5) and the window size p (Eq. 3). Fig. 7 varies each of them while holding the other at its default value, for Fast-WAM on LIBERO averaged over the four suites. The default point is the RePIC row of Tab. 3: success (97.35%) and calls (2.22) coincide with that table, and latency was re-timed together with the other sweep points (209.5 ms, vs. 204.77 ms in Tab. 3).

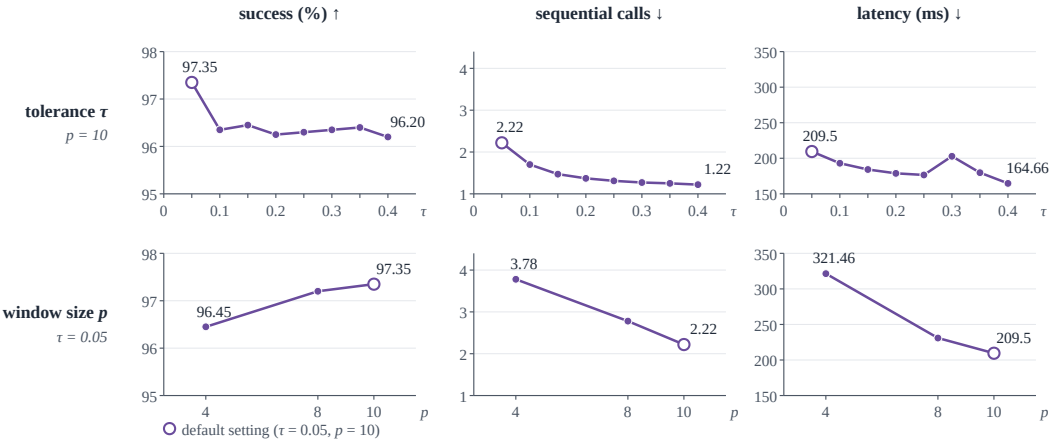


Figure 7: **Sensitivity to the tolerance and the window size.** Fast-WAM on LIBERO (average over the four suites), one hyperparameter varied at a time from the default setting (open marker; τ=0.05, p=10). *Top:* stride tolerance τ at p=10. *Bottom:* window size p at τ=0.05. Calls and latency are per chunk. The default setting is the RePIC row of Tab. 3; latencies were re-timed within the sweep.

Tolerance. The default τ=0.05 gives the highest success (97.35%); any looser tolerance costs about one point (96.20–96.45% for $\tau \in [0.10, 0.40]$), with no further trend beyond that step. Calls and latency fall steeply at first and then saturate: raising τ from 0.05 to 0.10 reduces calls from 2.22 to 1.70 and latency from 209.5 ms to 193.01 ms, and τ=0.40 reaches 1.22 calls and 164.66 ms per chunk, approaching a single refinement round per replan. Latency follows calls except at τ=0.30 (202.82 ms), where calls continue to decrease. The default is therefore the accuracy-first choice: a looser tolerance trades roughly one point of success for 0.5–1.0 fewer calls per chunk (8–21% lower latency).

Window size. A larger window refines more denoising states per batched call. From $p{=}4$ to $p{=}8$ and $p{=}10$, calls decrease from 3.78 to 2.78 and 2.22, latency from 321.46 ms to 230.89 ms and 209.5 ms, and success rises from 96.45% to 97.20% and 97.35%. Because each round advances by at most p states, $p < S$ forces at least $\lceil S/p \rceil$ rounds; with $p{=}S{=}10$, the setting used throughout the paper, the whole trajectory fits in one window and the number of rounds is governed by the tolerance alone. The price of a larger window is the batched computation and memory of a p_t-fold evaluation (Sec. 3), which on the RTX 4090 is outweighed by the shorter sequential path.

F LIMITATIONS

RePIC targets the sampling stage of chunked flow policies under receding-horizon control, so its gains scale with the share of latency spent in sequential velocity evaluations and with the temporal coherence of consecutive replans; policies whose latency is dominated by condition encoding, or control settings with abrupt changes between replans, benefit less. The formulation covers deterministic samplers on a fixed integration grid; extending trajectory transfer to stochastic samplers and other discretizations is a natural next step. Our evaluation centers on Fast-WAM in simulation with a physical-robot validation on one platform, and the tolerance and window size are set per policy; broader deployment across policies, tasks, and hardware, together with tolerances that adapt to task difficulty, would further characterize the accuracy–latency trade-off. Finally, RePIC-Flash shows that the sampler composes with compiled inference paths, and co-designing the batched window with serving systems is a promising direction for further gains.