Start Right, Arrive Right: Asynchronous Execution via Initial Noise Selection

Anonymous Author(s)
Affiliation
Address
email

Abstract: Action chunking enables robot policies to produce temporally coherent behavior, but generating multi-step action sequences with flow-based policies incurs latency that is incompatible with real-time control. Under asynchronous execution, the robot continues executing the current chunk while the next one is generated, causing even minor delays to create inconsistencies at chunk boundaries. Existing methods address this problem by steering generation toward the already executed action prefix. We instead show that prefix consistency can be achieved by selecting an appropriate initial noise before generation begins, allowing the unmodified flow ODE to produce a coherent next chunk. This reframes asynchronous inference as a noise selection problem rather than a trajectory steering problem. We introduce **PAINT**, a training-free method that finds this noise via backward Euler inversion and constructs the final chunk through a repainting rule. In summary, PAINT requires no gradients, retraining, or policy modification; yet it improves execution consistency and task performance across *12 simulated benchmarks* and *6 real-world manipulation tasks* spanning single-arm, bimanual, and humanoid embodiments. Website: https://paint-action-chunking.github.io.

Keywords: Asynchronous Inference, Action Chunking, Flow Matching

1 Introduction

Flow matching [1] and diffusion [2] policies have achieved remarkable dexterity by predicting *action chunks*, sequences of future actions generated in a single forward pass [3, 4, 5]. Action chunking improves temporal coherence, but each chunk requires multiple sequential denoising steps to generate. Under asynchronous inference, the robot cannot wait: it continues executing the previous chunk, and by the time the new one is ready, it has already advanced d steps. This creates the *prefix constraint*: those d actions must be consistent with what was just executed, or the robot experiences a discontinuous jump at the chunk boundary. Without enforcement, performance degrades substantially with delay, and this gap that widens as larger models push inference latency beyond the controller's sampling period.

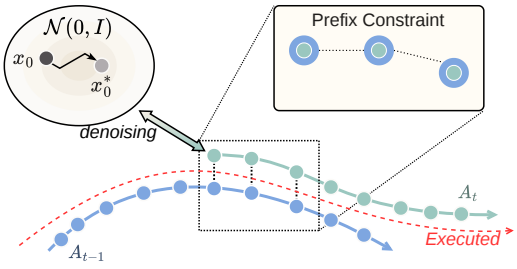


Figure 1: The *prefix constraint*: the first d actions of chunk A_t must approximate the last d actions of chunk A_{t-1}. A carefully chosen initial noise satisfies this constraint without modifying the policy.

Enforcing the prefix constraint is therefore critical, and the natural response is to steer generation toward the prefix target *during* denoising. Prior work does exactly this, using backpropagation through the policy [6], model retraining [7, 8], or additional sampling compute [9] to enforce continuity at generation time. Yet, this framing raises a deeper question: *"what if the trajectory never needed to*

be corrected in the first place?" "What if the right initial noise, chosen before *generation begins, would naturally produce a consistent prefix?"*

Fortunately, flow matching offers a cleaner path. Optimal-transport flow matching [1] (OT-FM) learns a transport map with an approximately *local* structure: each output position is governed largely by the noise at the corresponding input position, a property observed empirically in diffusion models by Mao et al. [10] and supported theoretically for OT-FM by Tong et al. [11]. This locality has a direct consequence: if the prefix of a generated chunk depends mainly on the prefix of the initial noise, then inverting the ODE from a desired prefix target recovers a noise that satisfies the prefix constraint under the unmodified forward pass, without velocity correction or gradients required. This reframes the prefix constraint as a *noise-selection problem* rather than a trajectory-steering problem.

This insight leads directly to our method named `PAINT` (**P**refix-**A**nchored **IN**i**T**ial noise), a training-free method that enforces the prefix constraint by inverting the flow ODE to find an initial noise x_0^*, then constructs the full chunk via the re-painting principle of Mao et al. [10]. Prefix and suffix of an action chunk are generated jointly from this re-painted noise, producing a continuation that is approximately consistent with the policy distribution under the executed prefix, unlike velocity-steering approaches, which do not explicitly enforce this property. `PAINT` requires (i) *no modification to the base policy, (ii) no training data, and (iii) no auto-differentiation framework at deployment*.

Our contributions are: (i) reframing asynchronous inference as a noise-selection problem, grounded in the locality structure of OT flow matching; (ii) `PAINT`, a training-free, backpropagation-free inference-time method that matches or improves over Real-Time Chunking (RTC) [6] on both task success and prefix consistency; and (iii) an empirical demonstration that `PAINT` transfers to any pretrained flow matching policy without retraining or modification, validated across 12 simulated benchmarks and six real-world tasks on two VLA architectures and three robot embodiments.

2 Related Work

`PAINT` sits at the intersection of two active research areas: real-time execution of action-chunking policies and structured manipulation of initial noise in generative models. We review each in turn.

Real-Time Execution of Action-Chunking Policies. Action chunking – predicting a fixed-length sequence of future actions in an inference call – has become the dominant paradigm for robot manipulation [3, 4], and large-scale VLAs [5, 12, 13, 14, 15] scale these architectures at the cost of inference latency. Black et al. [6] provide a detailed treatment of this latency problem; their RTC framework is our primary baseline. Existing methods address the gap between inference and execution at different stages. TT-RTC [7] and VLASH [8] fine-tune the policy to tolerate delay, while Streaming Diffusion and Streaming Flow Policies [16, 17] introduce training schemes for faster inference. At inference time, BID [9] uses candidate filtering through rejection sampling, while A2C2 [18], SAIL [19], and FASTER [20] modify the generated action trajectory through correction, guidance or compressing the sampling mechanisms. ABPolicy [21] enforces continuity with a B-spline action representation, and DiscreteRTC [22] uses discrete diffusion to align iterative generation with asynchronous execution better. Despite their diversity, all these methods intervene after the initial noise is sampled, either at training time, during denoising, or on the generated output, but never on the noise itself. `PAINT` acts at exactly that earlier point. A concurrent survey [23] compares several of these baselines but omits the noise-space perspective we adopt.

Initial Noise Manipulation and Inversion in Generative Models. The initial noise vector, traditionally treated as unstructured randomness, in fact carries semantic structure. Mao et al. [10, 24] demonstrated spatially localized generation in vision: perturbing the noise at position i primarily affects the output at i. Patil et al. [25] extended this to robot policies, showing that a carefully chosen constant noise vector improves frozen-policy performance. Noise manipulation has also been approached through learning: Wagenmaker et al. [26] train a noise-space policy via RL, RTI-DP [27] warm-starts denoising from the previous prediction, A2A [28] replaces noise with proprioceptive embeddings, and UniSteer [29] inverts a flow decoder for RL-based adaptation.

Inversion methods recover the initial noise for a desired output: DDIM inversion [30] runs the generative process backward, with later work improving its accuracy [31, 32, 33, 34], and flow matching admits single-step inversion [35]. PAINT is the first to apply ODE inversion to the prefix constraint in asynchronous action-chunk inference, requiring no learning, retraining, or human feedback, and then combining backward Euler inversion [32] with the re-painting principle of Mao et al. [10] to produce temporally coherent chunks.

3 Preliminaries and Motivation

We adopt the notation and problem formulation of Black et al. [6]. Given an observation o_t and initial noise x_0, a deterministic policy $\pi_\theta(o_t, x_0)$ outputs an *action chunk* $A_t^{0:H-1}$ of *action horizon* H. Rather than waiting for the chunk to be fully consumed, the robot executes only the first s actions $A_t^{0:s-1}$ before issuing the next inference call; we refer to s as the *execution horizon*. Since each inference call requires wall-clock time δ to complete N denoising steps, and the controller operates at period Δt, the robot advances $d = \lfloor \delta / \Delta t \rfloor$ timesteps during chunk generation; we call d the *inference delay*. Real-time execution requires $d \leq H - s$, guaranteeing that a new chunk is ready before the current one is exhausted. Consequently, by the time chunk A_t becomes available, the robot has already consumed $s + d$ actions from the previous chunk $A_{t-1}^{0:s+d-1}$.

The Problem of Prefix Constraints in Asynchronous Inference. In synchronous inference, the policy π_θ computes the next chunk A_t only after finishing execution of A_{t-1}. In asynchronous inference, by contrast, the robot executes A_{t-1} while A_t is being computed, so the next chunk must satisfy the following *prefix constraint* to ensure motion continuity:

$$A_{t-1}[s+i] = A_t[i] \quad \text{for } i = 0, 1, \ldots, d-1\,. \tag{1}$$

Violating it causes a discontinuous jump at the chunk boundary, producing jerky or unsafe motion.

Initial Noise as a Control Variable. When the velocity field $v_\pi(\cdot, o, \tau)$ is Lipschitz continuous in its first argument and continuous in τ, the continuous-time flow ODE defines a unique flow map from x_0 to x_1 [1, 36]. Under these idealized conditions, the map is invertible. In practice, however, learned policies are evaluated with finite-step numerical solvers and the target prefix may not lie exactly on the learned trajectory manifold. We therefore treat PAINT as an approximate inverse procedure: it only needs to recover an initial noise whose forward rollout produces a prefix sufficiently close to the executed actions. This makes x_0 a natural control variable: some output constraints can be addressed by searching over the initial noise x_0 appropriately, without modifying the velocity field $v_\pi(x_\tau, o, \tau)$ at any denoising step [24]. Recent work corroborates this view: Wagenmaker et al. [26] and Patil et al. [25] demonstrate that structured noise choices steer frozen policies toward desired behaviors, and Jiang et al. [17] bias the initial noise toward a running estimate to encourage smooth inter-chunk transitions. This motivates our approach: the prefix constraint (Equation (1)) defines a set in noise space, and satisfying it approximately can be approached by searching for an appropriate point x_0^* within that set.

4 Asynchronous Inference as a Noise Selection Problem

Existing methods such as RTC [6] and BID [9] enforce the prefix constraint by steering the velocity field v_π at every denoising step (see Figure 2b), accepting backpropagation or heavy compute as the price. We take a different route: under OT-FM, there exists an initial noise x_0^* such that the *unmodified* ODE already satisfies the prefix constraint. The problem reduces to finding x_0^*. Once found, inference runs exactly as during training: no guidance, no backpropagation, no velocity correction. We call this method **PAINT** (**P**refix-**A**nchored **IN**i**T**ial noise) (see Figure 2c). The following subsections derive PAINT: we first explain why it should be noise selection, then show how ODE inversion can find x_0^* efficiently.

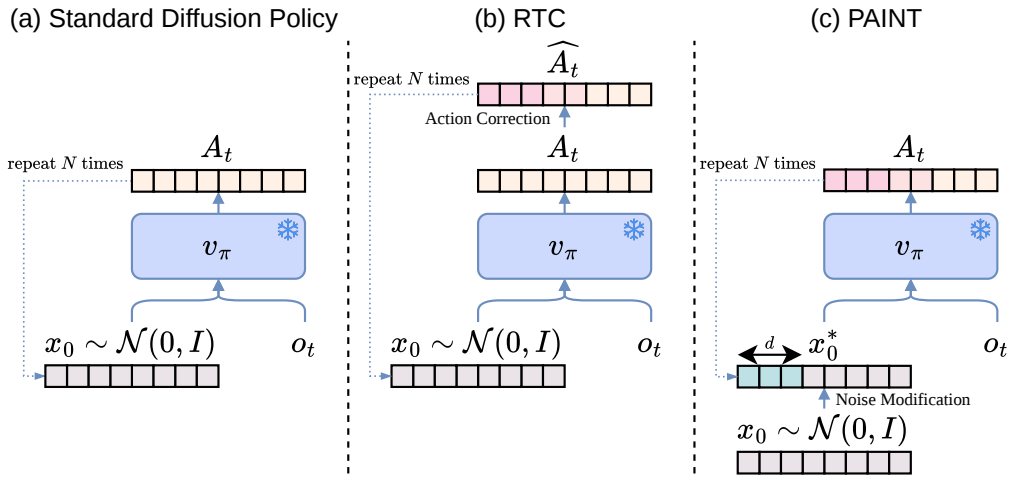


Figure 2: Overview of standard diffusion policy [4, 5, 12] (a), RTC [6] (b), and our proposed method PAINT (c), which leverages modified noise. Given a frozen, pretrained diffusion or flow-matching policy π_θ, instead of modifying the ODE at each denoising step, we modify the initial noise x_0 to get x_0^*, which satisfies the prefix constraint (Equation (1)). We show that x_0^* can improve policy performance while satisfying the prefix constraint across model architectures and embodiments.

4.1 From Velocity Steering to Noise Selection

Asynchronous inference can be addressed in two directions: modifying the ODE *during* generation to steer toward the prefix, or selecting the initial noise *before* generation so that the ODE automatically satisfies the constraint. Existing inference-time methods, including RTC [6], operate in the first direction. We argue that the second direction is both simpler and less biased toward the learned distribution. Simpler, because it requires no change to the velocity field v_π at any ODE step, the standard solver runs unaltered, with complexity absorbed entirely into a selection of the initial noise x_0. Less biased, because the trajectory is generated without modifying the learned flow field: since v_π is never corrected, the generated chunk more closely matches samples induced by the learned flow rather than the endpoint of a steered path that may lie outside the support of the learned distribution.

Velocity Steering May Deviate from Learned Flow. RTC applies a pseudoinverse correction to the velocity field at each ODE step:

$$v_\pi^{\text{RTC}}(A_t^\tau, o_t, \tau) \;=\; v_\pi(A_t^\tau, o_t, \tau) \;+\; w \cdot J^\dagger \cdot \left(Y - \widehat{A_t^1}\right). \tag{2}$$

where $J = \partial \widehat{A_t^1} / \partial A_t^\tau$ is the Jacobian of the denoiser prediction and $J^\dagger$ is its pseudoinverse. The correction pushes the predicted output $\hat{x}_1$ toward the prefix target. However, we decompose the correction into components parallel and orthogonal to the natural velocity field.

$$J^\dagger \cdot e \;=\; \delta v_\parallel \;+\; \delta v_\perp\,, \qquad e = Y - \widehat{A_t^1}\,. \tag{3}$$

the orthogonal component $\delta v_\perp \neq 0$ in general, steering x_t away from the natural ODE trajectory. The correction can move the trajectory away from the dynamics induced by the learned vector field, potentially producing chunks that differ from those generated by the unmodified policy.

Noise Selection Leaves the Learned Flow Field Unchanged. There exists an initial noise $x_0^* \in \mathbb{R}^{H \times D_{\text{action}}}$ such that the standard unmodified ODE already satisfies the prefix constraint. Running the ODE from x_0^* requires no correction to v_π at any step. Since `PAINT` changes only the initial noise and leaves the learned flow field unchanged, the generated chunk is obtained from the same forward dynamics as the base policy, conditioned on the observation o_t. The prefix constraint is satisfied not by distorting the distribution but by choosing the right starting point within it. The remainder of this section derives an efficient procedure for finding x_0^* as shown in Algorithm 1 below.

Algorithm 1 PAINT (**P**refix-**A**nchored **INi**t**T**ial noise)

Require: Observation o_t, executed prefix $A_{t-1}^{s:s+d-1}$, delay d, execution s, ODE steps N
Output: Action chunk A_t

$x_0^{\text{free}} \sim \mathcal{N}(0, I)$
$x_1^{\text{naive}} \leftarrow \pi_\theta(x_0^{\text{free}},\ o_t)$ ▷ naive forward pass [N calls]
$x_1^{\text{target}} \leftarrow \left[A_{t-1}^{s:s+d-1},\ x_1^{\text{naive}}[d{:}]\right]$ ▷ construct inversion target
$x_\tau \leftarrow x_1^{\text{target}}$
for $\tau = 1,\ 1-\frac{1}{N},\ \ldots,\ \frac{1}{N}$ **do**
  $x_\tau \leftarrow x_\tau - \frac{1}{N} \cdot v_\pi(x_\tau,\ o_t,\ \tau)$ ▷ backward Euler [N calls]
end for
$x_0^{\text{inv}} \leftarrow x_\tau$
$x_0^* \leftarrow \left[x_0^{\text{inv}}[{:}d],\ x_0^{\text{free}}[d{:}]\right]$ ▷ Mao re-painting rule
$A_t \leftarrow \pi_\theta(x_0^*,\ o_t)$ ▷ final forward pass [N calls]
return A_t

4.2 PAINT: Prefix-Anchored Initial Noise

To find x_0^*, we run the flow ODE in reverse. Starting from x_1^{target} at $\tau{=}1$ and applying backward Euler for N steps,

$$x_{\tau-\Delta\tau} = x_\tau - \Delta\tau\, v_\pi(x_\tau, o, \tau), \qquad \Delta\tau = \frac{1}{N}\,. \tag{4}$$

recovers an inverted noise x_0^{inv} that the model associates with x_1^{target}. The full procedure is in Algorithm 1. Two implementation details matter:

Constructing the target. We build x_1^{target} by fixing its prefix positions to $A_{t-1}^{s:s+d-1}$ and filling the remaining positions with the tail $x_1^{\text{naive}}[d\ {:}]$ of a standard forward pass from fresh noise, keeping x_1^{target} on the data manifold. Substituting zeros or random values for the free region moves it off-manifold, destabilizing backward Euler integration, and corrupting x_0^{inv}.

Re-painting the free region. Inversion yields $x_0^{\text{inv}}[{:}d]$, encoding the prefix, and $x_0^{\text{inv}}[d{:}]$, which we discard. Replacing the free region with fresh noise $\varepsilon \sim \mathcal{N}(0, I)$ is tempting but problematic: token mixing in v_π spreads the mismatch between ε and $x_0^{\text{inv}}[{:}d]$ across all positions during the final forward pass. Instead, we retain $x_0^{\text{free}}[d{:}]$ from the naive pass, the same noise used to generate $x_1^{\text{naive}}[d{:}]$ and hence the free region of x_1^{target}. Since $x_0^{\text{free}}[d{:}]$ already encodes the same free output as $x_0^{\text{inv}}[d{:}]$, substituting it introduces minimal disruption, following the re-painting principle in [10].

The resulting chunk $A_t = \pi_\theta(x_0^*, o_t)$ satisfies the prefix constraint and produces a continuation that approximates the prefix-conditioned policy distribution:

$$A_t^{0:d-1} \approx A_{t-1}^{s:s+d-1} \quad \text{(prefix)};\ A_t^{d:} \approx p_{\tau=1}\big(\cdot \mid A_{t-1}^{s:s+d-1}, o_t\big) \quad \text{(suffix)}\,. \tag{5}$$

5 Experiments

We design our experiments to answer two questions. First, *"how does PAINT compare to existing inference methods in terms of both success rate and execution time?"* Second, *"how does the choice of the inversion method affect PAINT's performance?"*

Metrics. We evaluate two sets of metrics. The first measures task performance and efficiency: **s**uccess **r**ate (SR↑) and **a**verage **t**ime for successful **r**ollouts (ATR↓). The second assesses how well the generated chunk respects the prefix constraint via the **con**sistency score (CON↓). Formally, let $\mathcal{S}$ be a set of successful rollouts, T_j the completion time of rollout j, and d is the inference delay:

$$\text{SR} = \frac{\#\text{successful trials}}{\#\text{total trials}}\,, \quad \text{ATR} = \frac{1}{|\mathcal{S}|}\sum_{j\in\mathcal{S}} T_j\,, \quad \text{CON} = \frac{1}{d}\sum_{i=0}^{d-1} \|A_{t-1}[s+i] - A_t[i]\|_2 \tag{6}$$

We provide more information about these metrics in the Appendix.

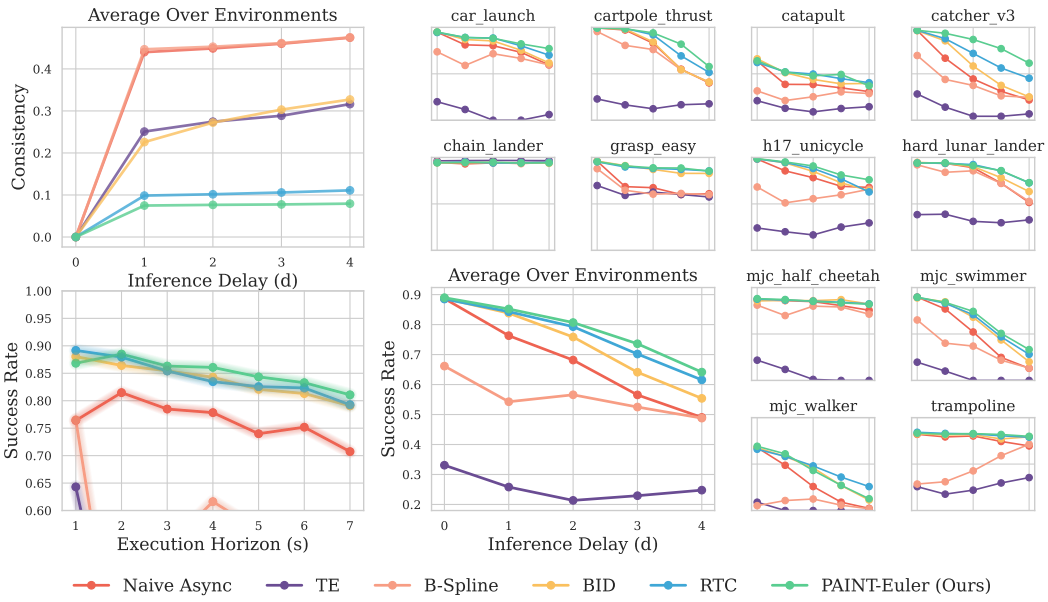


Figure 3: **Top left**: Inference delay (d) vs. prefix consistency (CON↓) across all environments. **Bottom left**: Execution horizon vs. success rate (SR↑) at d=1. **Right**: Inference delay (d) vs. success rate (SR↑) across simulated $d \in \{0, 1, 2, 3, 4\}$. PAINT-Euler achieves the strongest overall performance across all delay values. Each data point aggregates 2048 trials.

5.1 Simulated Benchmark

Setup. We follow the exact setup of RTC [6] on Kinetix [37], using 12 force-control environments and 4-layer MLP-Mixer [38] flow policies with $H = 8$. We report success rates (SR↑), and consistency scores (CON↓) with delays $d \in \{0, 1, 2, 3, 4\}$ and execution horizon $s \in \{d, \dots, H - d\}$.

We compare PAINT against five approaches, including (i) NAIVE ASYNC (independent chunks), (ii) TEMPORAL ENSEMBLING (TE; overlap averaging) [3], (iii) B-SPLINE REFITTING (post-hoc smoothing) [21], (iv) BID (CANDIDATE FILTERING) [9], and (v) RTC (GRADIENT-BASED GUIDANCE) [6]. Additional details about these baselines are in Appendix A.1.

Results. Figure 3 summarizes the simulated results. Under increasing delay, Naive Async degrades sharply, while TE and B-spline refitting provide limited robustness because averaging or smoothing generated actions does not explicitly condition on the executed prefix. B-spline refitting, in particular, leaves the prefix mismatch close to Naive Async, indicating that post-hoc smoothing alone is insufficient for chunk-boundary consistency. BID partially mitigates degradation, but remains below RTC and PAINT despite higher compute. PAINT-Euler achieves the strongest delay robustness without gradient computation and the lowest prefix mismatch, consistently improving over RTC. The execution-horizon sweep shows that PAINT and RTC benefit from shorter horizons because they can use more frequent feedback without introducing large chunk-boundary mismatches.

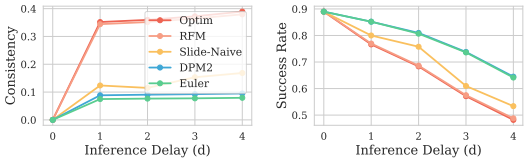


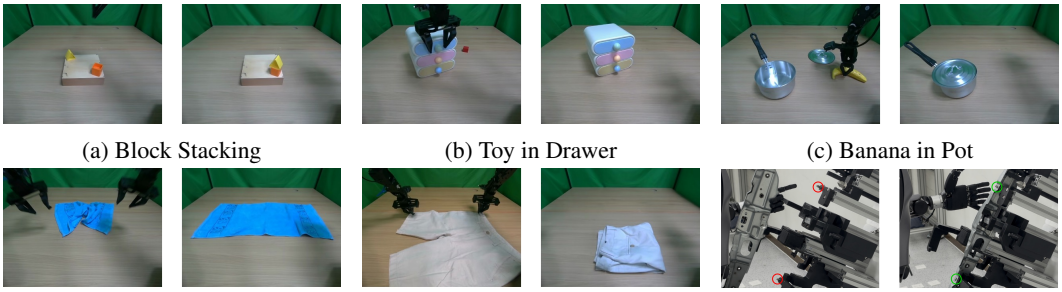
Figure 4: Average consistency scores and success rates over environments. Among various inversion methods, our chosen method (Euler) offers the best balance between quality and complexity.

Choices of Inversion Methods. We ablate the inversion step against four alternatives and a no-inversion baseline. Backward Euler (Euler) [32] runs v_θ in reverse for N steps; DPM-Solver (DPM2) [34] adds a midpoint correction for lower error at twice the cost; single-step reverse flow matching (RFM) [35] exploits the linear interpolant at t=1 to recover noise in one call ($x_0^{\text{inv}} =$

$x_1 - v_\theta(x_1^{\text{target}}, o, 1)$); and optimization-based (Optim) inversion minimizes $\|\pi_\theta(x_0^{\text{inv}} \mid o_t) - x_1^{\text{target}}\|^2$ via gradient descent. Slide-Naive is a no-inversion baseline, which shifts the stored noise from the previous inference forward by d positions and resamples the free region. Figure 4 shows that inversion quality correlates strongly with robustness under asynchronous execution. Euler achieves the best balance of consistency, stability, and efficiency – on par with the costlier DPM2 – while RFM, Optim, and Slide-Naive exhibits a larger prefix mismatch and degrades more sharply with increasing delay.

5.2 Real-World Evaluation

We next test whether PAINT transfers from controlled simulation to physical hardware across six real-world manipulation tasks spanning single-arm, bimanual (ALOHA [39]) and humanoid settings (see Figure 5) and two VLA architectures (GR00T-N1.5 [12] and π_0 [5]).



(a) Block Stacking (b) Toy in Drawer (c) Banana in Pot

(d) Towel Flinging (e) Shorts Folding (f) Part Placing

Figure 5: The environments for real-world evaluation. Each sub-figure's left image shows the initial state of environment; the right displays the goal state (more details are provided in the Appendix.

Setup. Our tasks: **(i) Block Stacking**, a precision pick-and-place task; **(ii) Toy in Drawer** and **(iii) Banana in Pot**, single-arm multi-stage contact tasks; **(iv) Towel Flinging** and **(v) Shorts Folding**, bimanual deformable-object tasks; and **(vi) Part Placing**, a humanoid precision-alignment task. All tasks except Towel Flinging use binary success scores; Towel Flinging is scored post-hoc based on towel flatness and spread. We use GR00T-N1.5 [12] (H=16, N=4) and π_0 [5] (H=50, N=10) under remote LAN inference, with 20 trials per method–task pair, comparing against TE and RTC under identical checkpoints and inference infrastructure. We focus on training-free inference-time baselines. Training-based approaches such as TT-RTC [7] and A2C2 [18] are orthogonal to PAINT as discussed in Appendix A.2 and left for future work. More details about setups are provided in Appendix B.

Results. Table 1 summarizes the real-world evaluation across all tasks using GR00T-N1.5 and π_0. Across both architectures, PAINT generally matches or improves over RTC while requiring no back-propagation through the policy. On GR00T-N1.5, PAINT improves the success rate on continuity-sensitive tasks such as Toy in Drawer, Towel Flinging, and Shorts Folding, and consistently improves CON when measured. This suggests that selecting the initial noise before generation can produce smoother chunk transitions than steering the denoising trajectory after generation begins. RTC enforces the prefix using gradient-based guidance [40], i.e., deployment-time backward operations, which can perturb the suffix after the prefix; on contact-rich or deformable-object tasks, such perturbations may produce small over-corrections or less stable contact behavior. On π_0, PAINT remains competitive on single-arm tasks and improves performance on bimanual deformable-object manipulation, indicating transferability across flow-based VLA architectures.

Compared with TE, PAINT substantially reduces rollout time while preserving prefix consistency. TE averages overlapping chunks, which can smooth motion but may also blend conflicting commands. This issue worsens with larger H in retry cases, where older actions may continue forward while the new chunk corrects backward, pulling the final command toward the middle. Finally, PAINT is faster than RTC on GR00T-N1.5 (86 ± 2 vs. 113 ± 3 ms) but slower on π_0 (311 ± 7

Table 1: Real-world evaluation over 20 trials per method-task pair. π_0 results are omitted for Part Placing because the public checkpoint targets manipulator embodiments rather than the humanoid arm-hand platform. TE [3] is used as a synchronous inference and therefore does not have a CON score; we mark this entry as "–" in the table.

Method	Block Stacking			Toy in Drawer			Banana in Pot		
	SR↑	ATR↓	CON↓	SR↑	ATR↓	CON↓	SR↑	ATR↓	CON↓
GR00T-N1.5 [12]									
TE [3]	0.55	28.27	–	0.60	23.19	–	0.60	29.57	–
RTC [6]	**0.75**	16.04	0.030	0.75	17.85	0.025	**0.70**	30.15	0.031
PAINT (Ours)	**0.75**	**15.32**	**0.023**	**0.85**	**17.08**	**0.023**	**0.70**	**29.79**	**0.026**
π_0 [5]									
TE [3]	0.20	58.35	–	0.15	53.27	–	0.10	63.20	–
RTC [6]	**0.50**	15.30	0.036	**0.65**	36.35	0.032	**0.55**	**37.41**	0.028
PAINT (Ours)	**0.50**	**14.90**	0.034	**0.65**	**30.39**	**0.028**	**0.55**	39.64	**0.026**

Method	Towel Flinging			Short Folding			Part Placing		
	SR↑	ATR↓	CON↓	SR↑	ATR↓	CON↓	SR↑	ATR↓	CON↓
GR00T-N1.5 [12]									
TE [3]	0.51	17.13	–	0.90	33.82	–	0.50	68.51	–
RTC [6]	0.76	**6.98**	0.028	0.90	**18.79**	0.027	**0.70**	18.28	0.030
PAINT (Ours)	**0.79**	7.44	**0.023**	**0.95**	19.77	**0.025**	**0.70**	**17.32**	**0.021**
π_0 [5]									
TE [3]	0.30	25.07	–	0.40	52.32	–			
RTC [6]	0.54	17.04	0.039	0.65	**29.64**	0.028		N/A	
PAINT (Ours)	**0.80**	**15.16**	**0.027**	**0.70**	30.54	**0.027**			

vs. 213 ± 4 ms), reflecting its core trade-off: replacing gradient-based correction with additional forward-only evaluations.

6 Conclusion

PAINT shows that asynchronous action-chunk execution can be addressed before generation begins, by selecting an initial noise that anchors the next chunk to the actions already executed. This avoids retraining and deployment-time gradient correction while preserving the base policy dynamics unchanged. Our experiments suggest that noise-space adaptation is a practical mechanism for improving chunk-boundary consistency in flow-based robot policies. PAINT is most relevant in regimes where inference latency cannot be fully eliminated, a common setting in practice, arising whenever policies are served over a local network, run on shared GPU infrastructure, or require many denoising steps as in large VLA models. Because PAINT uses only forward model evaluations, it is also more naturally compatible with graph-compiled deployment pipelines such as TensorRT than methods that require deployment-time vector-Jacobian products. More broadly, our results highlight the initial noise as a useful control interface [25, 26, 41, 42] rather than a passive random input, enabling real-time adaptation of generative robot policies without modifying the learned generative process.

7 Limitations

PAINT relies on two practical assumptions, each suggesting a direction for future work. First, it benefits from approximate locality between noise and action positions: changing the prefix region of the initial noise should mostly affect the prefix of the generated chunk. This locality is encouraged by optimal-transport flow matching, but may weaken for architectures with strong cross-position mixing or highly multimodal action distributions. Future work could quantify locality in deployed VLA backbones via noise-perturbation probes and, where it is weak, augment PAINT with a token-attention mask or a brief locality-preserving fine-tuning stage. Second, PAINT uses backward Euler inversion, which works well when the sampling trajectory is close to straight. This is reasonable for OT flow matching with a linear interpolant, but may be less accurate for variance-preserving diffusion models, whose probability paths are more curved. Extending PAINT to such models would likely require a dedicated inversion procedure (e.g., DDIM inversion or higher-order solvers such as DPM-Solver), possibly with a learned correction term for residual discretization error. Finally, our real-world evaluation uses a single natural inference-delay setting ($d \approx 3$); evaluating a broader range of delays on physical hardware would further characterize robustness.

References

[1] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *11th International Conference on Learning Representations, ICLR 2023*, 2023.

[2] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[3] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[4] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[5] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[6] K. Black, M. Y. Galliker, and S. Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025.

[7] K. Black, A. Z. Ren, M. Equi, and S. Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

[8] J. Tang, Y. Sun, Y. Zhao, S. Yang, Y. Lin, Z. Zhang, J. Hou, Y. Lu, Z. Liu, and S. Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

[9] Y. Liu, J. I. Hamid, A. Xie, Y. Lee, M. Du, and C. Finn. Bidirectional decoding: Improving action chunking via closed-loop resampling. *arXiv preprint arXiv:2408.17355*, 2024.

[10] J. Mao, X. Wang, and K. Aizawa. Guided image synthesis via initial image editing in diffusion model. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 5321–5329, 2023.

[11] A. Tong, K. Fatras, N. Malkin, G. Huguet, Y. Zhang, J. Rector-Brooks, G. Wolf, and Y. Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *arXiv preprint arXiv:2302.00482*, 2023.

[12] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[13] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walke, A. Walling, H. Wang, L. Yu, and U. Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

[14] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[15] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[16] S. H. Høeg, Y. Du, and O. Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.

[17] S. Jiang, X. Fang, N. Roy, T. Lozano-Pérez, L. P. Kaelbling, and S. Ancha. Streaming flow policy: Simplifying diffusion/flow policies by treating action trajectories as flow trajectories. In *ICRA 2025 Workshop: Beyond Pick and Place*, 2025.

[18] K. Sendai, M. Alvarez, T. Matsushima, Y. Matsuo, and Y. Iwasawa. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224*, 2025.

[19] N. R. Arachchige, Z. Chen, W. Jung, W. C. Shin, R. Bansal, P. Barroso, Y. H. He, Y. C. Lin, B. Joffe, S. Kousik, et al. Sail: Faster-than-demonstration execution of imitation learning policies. *arXiv preprint arXiv:2506.11948*, 2025.

[20] Y. Lu, Z. Liu, X. Fan, Z. Yang, J. Hou, J. Li, K. Ding, and H. Zhao. Faster: Rethinking real-time flow vlas. *arXiv preprint arXiv:2603.19199*, 2026.

[21] F. Yang, P. Jing, K. Qu, N. Zhao, and Y. Su. Abpolicy: Asynchronous b-spline flow policy for real-time and smooth robotic manipulation. *International Conference on Robotics and Automation (ICRA)*, 2026.

[22] P. Wang, K. Hong, C. Peng, K. Driggs-Campbell, M. Tomizuka, C. Xu, and C. Tang. Discretertc: Discrete diffusion policies are natural asynchronous executors. *arXiv preprint arXiv:2604.25050*, 2026.

[23] A. Agouzoul. Understanding asynchronous inference methods for vision-language-action models. *arXiv preprint arXiv:2605.08168*, 2026.

[24] J. Mao, X. Wang, and K. Aizawa. The lottery ticket hypothesis in denoising: Towards semantic-driven initialization. In *European Conference on Computer Vision*, pages 93–109. Springer, 2024.

[25] O. Patil, O. Biza, T. Weng, K. Schmeckpeper, W. Thomason, X. Zhang, R. Walters, N. Gopalan, S. Castro, and E. Rosen. You've got a golden ticket: Improving generative robot policies with a single noise vector. *arXiv preprint arXiv:2603.15757*, 2026.

[26] A. Wagenmaker, Y. Zhang, M. Nakamoto, S. Park, W. Yagoub, A. Nagabandi, A. Gupta, and S. Levine. Steering your diffusion policy with latent space reinforcement learning. In *9th Annual Conference on Robot Learning*, 2025.

[27] Y. Duan, H. Yin, and D. Kragic. Real-time iteration scheme for diffusion policy. *arXiv preprint arXiv:2508.05396*, 2025.

[28] J. Jia, G. Li, X. Chen, T. An, Y. Hu, J. Li, X. Guo, and J. Yang. Action-to-action flow matching. *arXiv preprint arXiv:2602.07322*, 2026.

[29] J. Lu, X. Qin, Y. Jiang, K. Wang, C. Zhang, B. Liang, J. Yang, M. Xu, and L. Zhao. Unified noise steering for efficient human-guided VLA adaptation. *arXiv preprint arXiv:2605.10821*, 2026.

[30] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

[31] B. Meiri, D. Samuel, N. Darshan, G. Chechik, S. Avidan, and R. Ben-Ari. Fixed-point inversion for text-to-image diffusion models. *arXiv e-prints*, pages arXiv–2312, 2023.

[32] Z. Pan, R. Gherardi, X. Xie, and S. Huang. Effective real image editing with accelerated iterative diffusion inversion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15912–15921, 2023.

[33] G. Zhang, J. P. Lewis, and W. B. Kleijn. Exact diffusion inversion via bidirectional integration approximation. In *European Conference on Computer Vision*, pages 19–36. Springer, 2024.

[34] S. Hong, K. Lee, S. Y. Jeon, H. Bae, and S. Y. Chun. On exact inversion of dpm-solvers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7069–7078, 2024.

[35] Z. Li, S. Tang, and N. Azizan. Reverse flow matching: A unified framework for online reinforcement learning with diffusion and flow policies. *arXiv preprint arXiv:2601.08136*, 2026.

[36] Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

[37] M. Matthews, M. Beukman, C. Lu, and J. N. Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. In *The Thirteenth International Conference on Learning Representations*, 2025.

[38] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

[39] Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

[40] J. Song, A. Vahdat, M. Mardani, and J. Kautz. Pseudoinverse-guided diffusion models for inverse problems. In *International conference on learning representations*, 2023.

[41] C. Pan, G. Anantharaman, N.-C. Huang, C. Jin, D. Pfrommer, C. Yuan, F. Permenter, G. Qu, N. Boffi, G. Shi, et al. Much ado about noising: Dispelling the myths of generative robotic control. *arXiv preprint arXiv:2512.01809*, 2025.

[42] X. Ye, R. H. Yang, J. Jin, Y. Li, and A. Rasouli. Ra-dp: Rapid adaptive diffusion policy for training-free high-frequency robotics replanning. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6227–6234. IEEE, 2025.

Appendix Contents

A Kinetix Benchmark

Kinetix [37] is an open-ended, two-dimensional rigid-body physics benchmark designed for evaluating generalist agents across a broad range of control problems. Each Kinetix environment is procedurally defined by a configuration of polygons, circles, joints, and thrusters governed by a 2D physics simulator, yielding a unified action and observation space across drastically different tasks. Each environment also defines a binary success criterion specified through colour-coded entities (e.g. green target circles, red obstacles to avoid) and goal joints that must be activated, so success is determined automatically by the simulator. This makes Kinetix particularly well-suited for evaluating asynchronous inference methods: the same policy architecture and learning pipeline must operate over locomotion, manipulation, control, projectile dynamics without task-specific modifications, and a single success metric is directly comparable across the entire benchmark. See Figure 6.

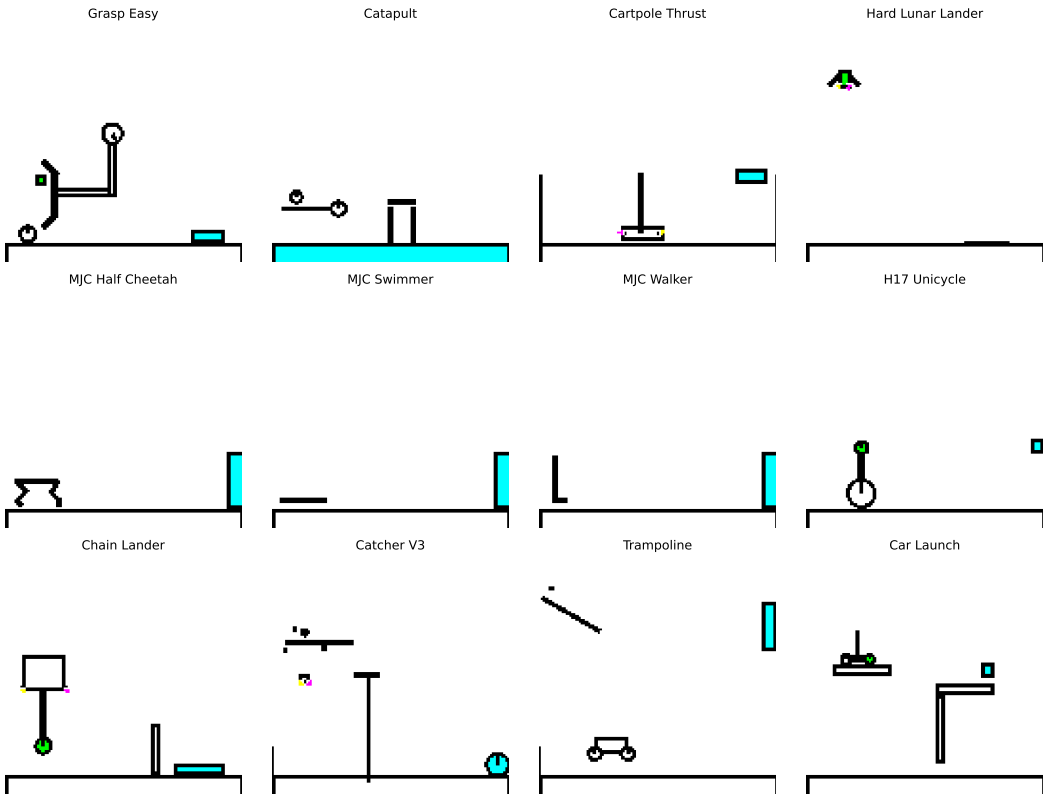


Figure 6: The visualizer of 12 environments in Kinetix [37].

A.1 Baseline Details

We compare `PAINT` against representative baselines for sync/asynchronous action-chunk execution. Unless otherwise stated, all methods use the same pretrained policy checkpoint and differ only in how action chunks are executed, filtered, or modified at chunk boundaries.

Naive Async. The robot executes the current chunk while the next chunk is generated in parallel, and then switches directly to the new chunk as soon as it is fully available. Each query uses independently sampled initial noise, and no constraint is imposed between the executed actions and the prefix of the new chunk. This method is cheap and requires no retraining or gradients, but can create discontinuities at chunk boundaries.

Temporal Ensembling (TE) [3] averages overlapping action predictions across chunks, following the aggregation strategy of ACT. In recovery scenarios, older chunks may perpetuate the previous motion while newer ones attempt correction; averaging these conflicting predictions can bias the action toward an intermediate trajectory, slowing task completion.

B-spline Refitting. We use only the post-hoc trajectory-refitting component inspired by ABPolicy [21], without retraining the base policy or changing its action representation. The generated chunk is refit with a smooth B-spline to reduce abrupt transitions, testing whether smoothness alone is sufficient for asynchronous execution. Since the refitting does not condition generation on the executed prefix, it can reduce visual jerkiness while still creating a prefix mismatch.

Bidirectional Decoding (BID) [9] samples multiple candidate chunks and selects one according to a consistency or scoring criterion. It requires no gradients or retraining, but increases inference cost because multiple chunks are generated per query. Unlike our method, BID addresses asynchronous mismatch by filtering generated outputs rather than modifying the denoising process or selecting the initial noise directly.

Real-Time Chunking (RTC) [6] steers the denoising trajectory during generation to satisfy the prefix constraint. At each denoising step, it uses vector-Jacobian products (VJPs) to move the predicted final chunk toward the executed prefix. RTC directly targets chunk-boundary consistency, but requires deployment-time backward operations through the policy, which may be expensive or unsupported in graph-compiled runtimes.

Training-time RTC (TT-RTC) [7] modifies the policy during training so that the learned model becomes more tolerant to inference delay. It requires fine-tuning but does not require VJPs at deployment. We include TT-RTC to distinguish training-time adaptation from `PAINT`'s inference-time noise selection, and additionally evaluate **TT-RTC +** `PAINT` to test whether prefix-anchored noise selection can further improve a delay-aware policy.

A2C2 [18] is an inference-time action-correction method that uses the recent execution context to improve chunk-boundary consistency without retraining the base policy. It operates in action space after generation, while `PAINT` acts earlier by selecting the initial noise. **A2C2 +** `PAINT` uses A2C2's correction mechanism to chunks generated from `PAINT`'s prefix-anchored noise. The goal is to test whether noise-space anchoring and action-space correction provide complementary benefits.

A.2 PAINT vs. Training-Time Methods

A2C2 [18] and TT-RTC [7] improve asynchronous execution by modifying the policy during training, whereas `PAINT` operates purely at inference time by selecting a prefix-anchored initial noise. We therefore evaluate `PAINT` on top of both trained policies to test whether inference-time noise selection is complementary to training-time delay adaptation. As shown in Figure 7, `PAINT` substantially reduces prefix mismatch for both A2C2 and TT-RTC across all nonzero delays. Adding `PAINT` to TT-RTC decreases prefix consistency error (CON) from 0.11 to 0.08 at $d = 4$ while preserving or improving task success, and adding `PAINT` to A2C2 similarly reduces CON by approximately 80%. In terms of task success, adding `PAINT` largely preserves A2C2's performance across execution horizons and delays, while TT-RTC+`PAINT` achieves comparable success at small delays and greater robustness at larger delays. These results indicate that `PAINT` is orthogonal to training-time adaptation: it can be applied on top of existing delay-aware policies to improve prefix consistency and, in some settings, high-delay robustness, without additional retraining.

B Real-Robot Environment Details

We evaluate `PAINT` on six real-world manipulation tasks spanning single-arm manipulation, bimanual deformable-object manipulation, and humanoid part placement (see Figure 8). All methods are evaluated using the same policy checkpoint, robot hardware, camera observations, action horizon, and inference infrastructure; the only difference is the inference-time execution strategy.

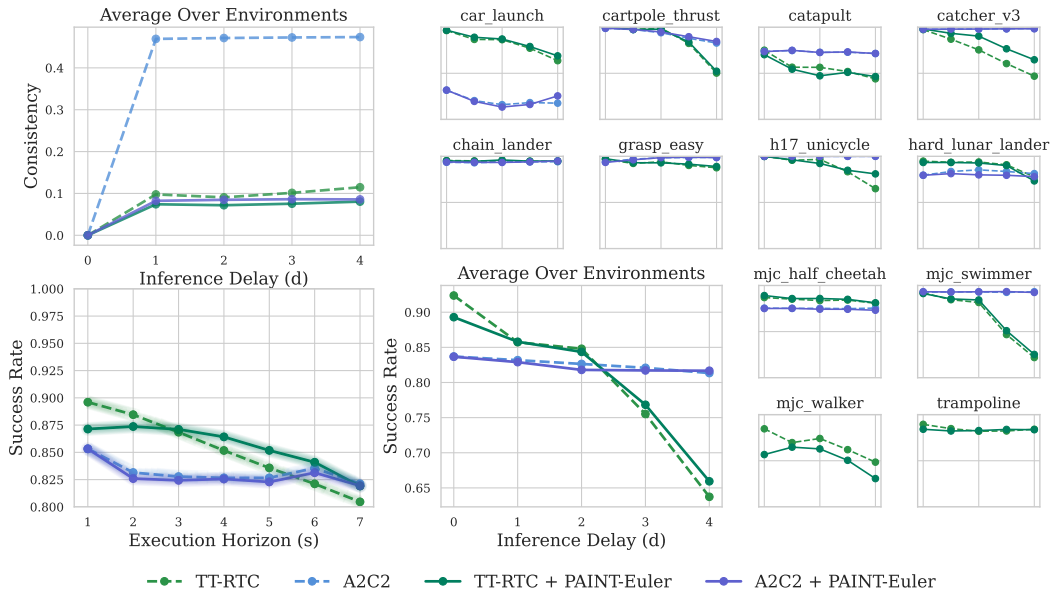


Figure 7: PAINT v.s. training-time delay-aware methods on the Kinetix benchmark. We evaluate PAINT on top of A2C2 [18] and training-time RTC (TT-RTC) [7] , measuring both consistency score and task success rate under varying execution horizons and inference delays.

Hardware Setup. For ALOHA [39] experiments, the robot consists of two ViperX 6-DoF arms, each equipped with a parallel-jaw gripper, for a total of 14 controllable degrees of freedom (12 arm + 2 gripper). We use a single-arm configuration (7 DoFs: 6 arm + 1 gripper) for rigid-object manipulation tasks and the full bimanual configuration (14 DoFs) for deformable-object manipulation. For humanoid experiments, we use a robot with a single arm of 7 DoFs and a 6-DoF dexterous hand. The policy receives 640×480 RGB observations from 3 cameras, proprioceptive joint positions, and language/task conditioning when required by the base VLA.

Control and Inference. The controller runs at 20 Hz on a workstation equipped with an NVIDIA RTX 4070 (16 GB), AMD Ryzen 9 7900X, and 64 GB RAM, running Ubuntu 24.04. Each policy call predicts an action chunk of horizon H, executed asynchronously while the next chunk is generated, yielding a natural inference delay of $d = \lfloor \delta / \Delta t \rfloor$ where δ is the wall-clock inference time including network transfer.

Evaluation Procedure. Each method-task pair is evaluated over 20 trials with random initial-state distributions. We report success rate (SR), average time for successful rollouts (ATR), and prefix consistency error (CON) when applicable. A trial is marked as failed if the robot violates the task success criterion, loses contact with the target object in an unrecoverable way, enters an unsafe configuration, or reaches the maximum episode length. We provide the task description in Table 2.

C Inversion Methods

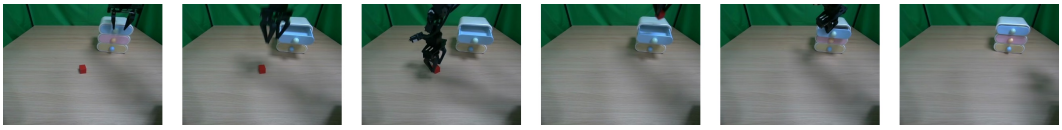
We compare several ways to recover a prefix-anchored initial noise. All variants construct a target endpoint x_1^{target} by combining the executed prefix with a suffix generated from a naive forward pass. They differ only in how they recover the corresponding initial noise. *The number of model calls in Table 3 refers to the inversion step alone*; total PAINT inference cost (including the naive forward pass and final forward pass) is reported in Table 4.

Backward Euler. The default PAINT implementation starts from x_1^{target} and integrates backward:

$$x_{\tau - \Delta\tau} = x_\tau - \Delta\tau \, v_\theta(x_\tau, o, \tau) \, . \tag{7}$$

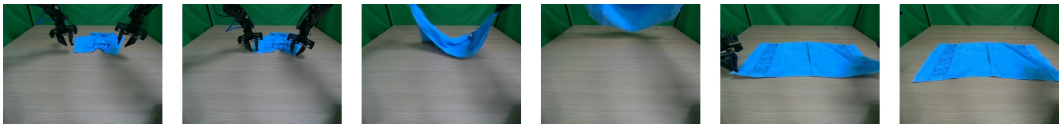
This requires N model evaluations and no gradients.

(a) Block Stacking



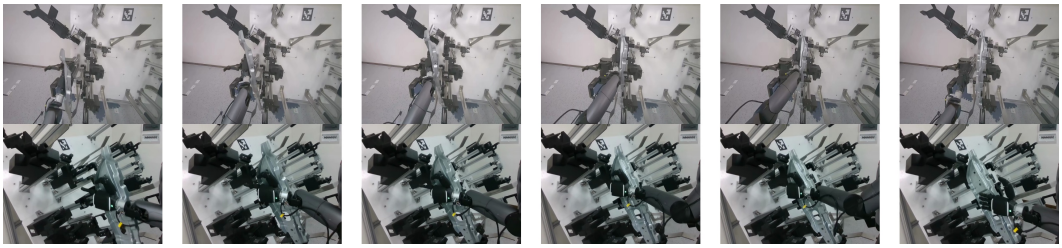
(b) Toy in Drawer

(c) Banana in Pot



(d) Towel Flinging

(e) Shorts Folding



(f) Part Placing

Figure 8: Examples of completed sequences in real-world tasks.

DPM2 / Midpoint Inversion. A higher-order solver reduces local truncation error by evaluating the vector field at an intermediate point. This can improve inversion accuracy when the vector field is curved, but approximately doubles the number of model evaluations.

Single-step RFM Inversion. For flow matching with a linear interpolant, a single-step estimate can be obtained near $\tau = 1$:

$$x_0^* \approx x_1^{\text{target}} - v_\theta(x_1^{\text{target}}, o, 1)\,. \tag{8}$$

This is computationally attractive but can be inaccurate when the learned trajectory is nonlinear or when the target prefix lies in a high-curvature region of the action distribution.

Optimization-based Inversion optimizes x_0 directly:

$$\min_{x_0} \left\| \pi_\theta(x_0, o) - x_1^{\text{target}} \right\|_2^2 . \tag{9}$$

This can produce accurate inversions but requires repeated forward passes and gradients through the policy, making it less suitable for latency-critical deployment.

Table 2: Real-world task descriptions.

Task	Embodiment	Task Description
Block Stacking	Single-arm	A precision pick-and-place task requiring the robot to grasp a triangular block and stack it stably on top of a square block.
Toy in Drawer	Single-arm	A multi-stage contact task involving drawer opening, cube placement inside the drawer, and drawer closing.
Banana in Pot	Single-arm	A sequential object-manipulation task requiring the robot to place a banana into a pot and then close the pot with a lid.
Towel Flinging	Bimanual	A dynamic deformable-object manipulation task requiring coordinated bimanual grasping, lifting, and flinging to spread a towel flat on the table. Performance is reported as a post-hoc score from 0 to 10; the score measures the extent to which the towel covers the table, with flatter configurations receiving higher scores.
Shorts Folding	Bimanual	A bimanual deformable-object task requiring coordinated folding of the shorts into a compact final configuration.
Part Placing	Humanoid arm-hand	A precision-alignment task requiring the humanoid robot to place a part onto a two-pin jig with accurate pose alignment.

Table 3: Comparison of inversion methods used in PAINT ablations. N denotes the number of flow-matching solver steps.

Method	Description	Model Calls (inversion only)	Gradients?	Expected Behavior
Backward Euler	Reverse ODE integration	N	No	Stable, simple, good cost–quality trade-off
DPM2 / midpoint	Higher-order reverse integration	$\approx 2N$	No	Lower discretization error, higher cost
Single-step RFM	Linear-interpolant inverse at $\tau = 1$	1	No	Fast but less accurate for curved trajectories
Optimization	Minimize endpoint reconstruction error	$\gg N$ + grad.	Yes	Accurate but slow and deployment-unfriendly
Slide-Naive	Shift stored noise and resample suffix	≈ 0 (amortized)	No	Cheap but accumulates stale-noise error

Slide-Naive shifts the stored initial noise forward by d positions and samples fresh noise for the suffix. This is inexpensive, but the shifted noise may become stale as the observation and action distributions evolve over time.

D Invertibility, Discretization Error, and Stability

PAINT relies on approximately inverting the learned flow dynamics from a target action chunk to an initial noise. For continuous-time flow matching, the ODE

$$\frac{dx_\tau}{d\tau} = v_\theta(x_\tau, o, \tau)\,, \qquad \tau \in [0, 1]\,, \tag{10}$$

admits a unique solution under standard regularity conditions, such as Lipschitz continuity of v_θ in x_τ and continuity in τ [36]. Under these conditions, the continuous flow map is invertible. However, practical policies are evaluated with finite-step numerical solvers, so the inverse recovered by PAINT is only approximate.

Discretization Error. PAINT-Euler uses backward Euler integration:

$$x_{\tau-\Delta\tau} = x_\tau - \Delta\tau\, v_\theta(x_\tau, o, \tau)\,. \tag{11}$$

This introduces a discretization error that depends on the step size $\Delta\tau = 1/N$, the local curvature of the learned vector field, and the numerical stability of the reverse integration. When N is small, the step size is large, and the recovered noise may deviate from the true inverse trajectory, increasing the prefix mismatch after the final forward pass. A larger N improves the inversion accuracy but increases the inference cost.

Stability Considerations. Reverse integration is stable when the solver step size is sufficiently small relative to the local Lipschitz constant of the vector field. If $\Delta\tau$ is too large, the Euler inversion can overshoot in regions where v_θ changes rapidly, especially for highly multimodal action distributions or architectures with strong cross-token mixing. In practice, we find that using the same number of reverse steps as the forward inference solver provides a good trade-off between stability and cost.

Approximate Inversion in Practice. Our method does not require the discretized policy to be exactly invertible. It only requires that backward integration recovers an initial noise whose forward rollout produces a prefix sufficiently close to the executed prefix. `PAINT` should therefore be interpreted as an approximate inverse procedure for prefix anchoring, not as an exact inverse of the learned policy. Notably, all backward integration steps are gradient-free forward evaluations of v_θ, making `PAINT` compatible with compiled inference runtimes such as TensorRT.

E Inference Method Comparison

Table 4 compares `PAINT` with representative inference-time and training-time approaches for asynchronous action-chunk execution. Model-call counts in this table refer to the *full inference pipeline*, including any naive forward pass, inversion, and final forward pass.

Table 4: Comparison of asynchronous inference methods. N denotes the number of denoising or flow-matching steps, B denotes the number of candidate chunks in rejection-sampling methods, and VJP denotes vector-Jacobian product.

Method	Retraining?	Gradients?	Approx. Model Evaluations	Notes
Naive Async	No	No	N	Independent noise per chunk; no prefix enforcement
Temporal Ensembling	No	No	N per query	Dense re-querying with overlapping chunks
BID	No	No	BN	Rejection sampling over candidate chunks
RTC	No	Yes	N (forward + VJP)	Steers velocity field at each denoising step
TT-RTC	Yes	No	N	Trains policy to tolerate delay
`PAINT`-RFM	No	No	$N+1$	Single-step inverse, then final forward pass
`PAINT`-Euler	No	No	$3N$	Naive forward, backward Euler inversion, final forward
`PAINT`-DPM2	No	No	$\approx 4N$	Higher-order inversion, higher cost
`PAINT`-Optim	No	Yes	$\gg N$ + grad.	Direct latent optimization; slow but diagnostic
`PAINT`-Slide-Naive	No	No	$\approx N$	Reuses shifted noise after initial inversion

Deployment Implications. RTC and optimization-based inversion require gradients or VJPs through the policy during deployment. This may be difficult to support in graph-compiled inference runtimes or systems optimized only for forward execution. In contrast, `PAINT`-Euler requires only forward evaluations of the velocity network, making it easier to integrate with deployment runtimes such as TensorRT or other graph-mode accelerators.

Compute Trade-off. `PAINT`-Euler is more expensive than naive asynchronous inference in raw model calls, but avoids backpropagation and improves prefix consistency. `PAINT`-Slide provides a lower-cost variant by reusing and shifting the recovered noise across consecutive chunks, at the cost of possible noise staleness. This creates a practical trade-off between consistency, latency, and deployment simplicity.

F Repainting Design

The term *repainting* refers to replacing the prefix portion of the initial noise, not the suffix. The suffix noise $x_0^{\text{free}}[d:]$ is sampled from the prior $\mathcal{N}(0, I)$ at the very beginning and is never changed. What motivates the procedure is that a naive forward pass from x_0^{free} produces a chunk x_1^{naive} whose prefix does not satisfy the prefix constraint. Rather than discarding the entire noise and starting over, we keep the suffix noise intact — it already encodes a valid continuation under the current observation — and *repaint only the prefix noise* $x_0^{\text{free}}[:d]$ with the inverted $x_0^*[:d]$ that anchors the output to the executed prefix. The final repainted noise is:

$$x_0^{\text{repaint}} = \big[\ \underbrace{x_0^*[:d]}_{\text{repainted}},\ \underbrace{x_0^{\text{free}}[d:]}_{\text{original}}\ \big]. \tag{12}$$

Why Discarding $x_0^*[d:]$? The inverted suffix $x_0^*[d:]$ was recovered from a target x_1^{target} whose prefix was manually replaced. Although it produces a valid reverse trajectory for this constructed target, it aligns too closely with the artificial target endpoint and reduces suffix diversity. Repainting the prefix with keeping $x_0^{\text{free}}[d:]$ instead preserves the stochastic continuation of the original policy while anchoring only the prefix.

Why Keeping $x_0^{\text{free}}[d:]$ Rather Than Fresh Noise? The suffix noise was sampled from the correct prior and used to generate $x_1^{\text{naive}}[d:]$, which is also the free region of x_1^{target} that we inverted. The model already associates $x_0^{\text{free}}[d:]$ with a valid on-manifold continuation under observation o. Replacing it with fresh noise $\varepsilon \sim \mathcal{N}(0, I)$ introduces a discontinuity at position d: the token mixing layers of v_θ propagate this mismatch between $x_0^*[:d]$ and $\varepsilon[d:]$ across all positions during the final forward pass, degrading prefix quality. Keeping the original suffix avoids this coupling error.

G Evaluation Metrics

Success Rate (SR) is the fraction of trials that satisfy the task-specific success criterion:

$$\text{SR} = \frac{\#\text{successful trials}}{\#\text{total trials}}. \tag{13}$$

Average Time for Successful Rollouts (ATR) is the mean completion time over successful trials:

$$\text{ATR} = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} T_i \,, \tag{14}$$

where $\mathcal{S}$ is the set of successful trials and T_i is the completion time of trial i.

Prefix Consistency Error (CON) measures how closely the newly generated chunk matches the actions already executed during inference:

$$\text{CON} = \frac{1}{d} \sum_{i=0}^{d-1} \|A_t[i] - A_{t-1}[s+i]\|_2 \,. \tag{15}$$

For real-world experiments, actions are reported in joint-angle space (radians) without per-dimension weighting; rotational and translational dimensions are treated equally. Lower CON indicates better satisfaction of the prefix constraint.