URGENT ACTIONS GO FIRST: URGENCY-AWARE DENOISING FOR REAL-TIME VLA CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion and flow-matching Vision-Language-Action (VLA) policies generate action chunks through iterative denoising, incurring substantial inference latency that severely limits real-time robotic control. Existing acceleration methods treat an action chunk as a monolithic computational unit, ignoring a crucial physical reality of receding-horizon control: actions are generated jointly but consumed sequentially, resulting in inherently heterogeneous execution urgencies. We exploit this asymmetry to introduce Urgency-Aware Denoising (UAD), a novel inference-time framework that allocates denoising computation according to when each action is physically needed. UAD releases time-critical urgent actions after fewer denoising steps while overlapping the continued background refinement of tail actions with physical execution. However, heterogeneous denoising introduces two key challenges: early-release errors in urgent actions and trajectory inconsistency in tail actions. UAD elegantly resolves both through two core mechanisms: Trajectory Reconciliation, which reconstructs unified internal state evolution to restore joint denoising coherence without additional model evaluations, and Ghost Action Correction, which leverages non-executed ghost continuations to dynamically compensate for early-release errors across remaining executable actions. Extensive evaluations across multiple VLA architectures, simulation benchmarks, and real-world manipulation tasks demonstrate that UAD achieves up to a $1.89\times$ speedup in average action availability latency while maintaining comparable success rates to vanilla inference with optimal denoising budget, offering a more favorable success-latency trade-off than state-of-the-art VLA acceleration baselines.

1 INTRODUCTION

Diffusion and flow-matching policies have emerged as a dominant paradigm for Vision-Language-Action (VLA) models, enabling expressive and high-quality generation of temporally coherent action sequences (Chi et al., 2025; Black et al., 2024; Physical Intelligence et al., 2025). In diffusion-based VLA policies, actions are generated in chunks, where each chunk is progressively refined through iterative denoising steps (Physical Intelligence et al., 2025). While this iterative process improves action quality, repeated evaluations of the diffusion transformer (DiT) incur substantial inference latency, which severely constrains real-time robotic control (Black et al., 2025). In a typical setting of our measurements, executing an action chunk takes 500 ms, while generating the next chunk requires approximately 300 ms even on datacenter-grade hardware. Under synchronous execution, this chunk-level generation delay forces the robot to spend over 30% of each execution cycle idle while waiting for new actions, a bottleneck that is more pronounced on resource-constrained edge devices (Nguyen et al., 2026).

To mitigate this inference bottleneck, existing acceleration methods have explored general optimizations (*e.g.*, quantization (Xu et al., 2026), pruning (Pei et al., 2026), and feature caching (Ji et al., 2026a)) as well as VLA-oriented techniques like adaptive step reduction (Yu et al., 2025) and generation-execution overlapping (Black et al., 2025). However, existing acceleration approaches largely treat an action chunk as a monolithic computational unit, applying a shared denoising budget across all action tokens. This overlooks a fundamental physical reality in receding-horizon control: *actions within a chunk are generated jointly but consumed sequentially, resulting in inherently heterogeneous execution urgencies.*

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

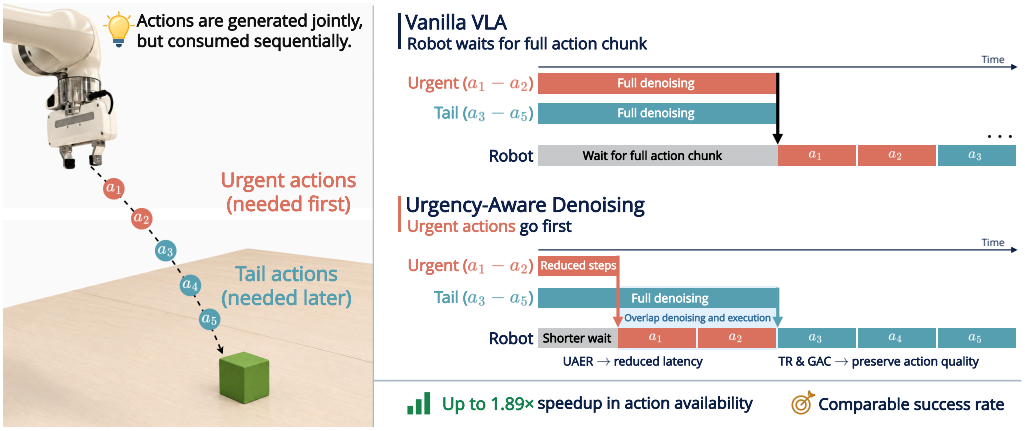


Figure 1: **Illustration of Urgency-Aware Denoising (UAD). Insight:** Actions are generated jointly but consumed sequentially, creating heterogeneous execution urgencies. **Method:** UAD releases urgent actions after fewer denoising steps and overlaps the remaining denoising with physical execution. **Modules:** Urgency-Aware Early Release (UAER) releases urgent actions early to reduce latency, Trajectory Reconciliation (TR) restores consistent joint denoising, and Ghost Action Correction (GAC) compensates for early-release errors using ghost actions. **Benefits:** UAD reduces action availability latency while maintaining comparable task success rates.

This asymmetry suggests a new design principle: *denoising computation should be allocated according to execution urgency rather than distributed uniformly across the chunk.* The first actions are the most urgent, as their generation latency directly determines when the robot can initiate physical execution. In contrast, subsequent tail actions have a larger temporal buffer before execution, allowing their refinement to naturally overlap with the physical execution of early actions. In practice, this principle concretely materializes as processing urgent actions with fewer denoising steps to release them early, while allowing tail actions to retain full denoising in the background.

However, realizing such heterogeneous denoising is non-trivial, as it introduces two core challenges. First, releasing urgent actions early inevitably introduces *early-release errors*, causing action deviations from the baseline. Second, since actions within a chunk are coupled through shared attention in the DiT, altering the state trajectories of urgent actions disrupts the joint attention context. This induces severe *trajectory inconsistency* and errors in the tail actions, even if those tail actions undergo full denoising steps. Consequently, naive early release fails to maintain action fidelity.

To address these challenges, we propose **Urgency-Aware Denoising (UAD)**, an inference-time framework that accelerates action availability while safeguarding action trajectory. Fig. 1 illustrates the insight, method, and benefits of UAD. The framework comprises three core components: (1) **Urgency-Aware Early Release (UAER)** performs fewer denoising steps for urgent actions to make them available immediately, allowing tail action denoising to overlap with physical execution. (2) **Trajectory Reconciliation (TR)** reconstructs unified hidden state trajectories for urgent actions using already-computed velocities, restoring joint denoising consistency for tail actions with zero additional model evaluations. (3) **Ghost Action Correction (GAC)** uses the non-executed continuation ("ghost action") of released urgent actions to estimate early-release errors, dynamically compensating for them across remaining executable actions. Together, these components effectively mitigate the trade-off between action latency and trajectory coherence, enabling rapid action availability while mitigating errors caused by early release.

In summary, our contributions are:

- **Urgency-Aware Inference Paradigm.** We introduce a new inference-time perspective for real-time VLA control, which allocates denoising computation according to temporal execution urgency rather than uniformly across an action chunk. Driven by the observation that chunked actions are generated jointly but consumed sequentially, this principle enables urgent actions to be released significantly earlier, cutting action availability latency.

- **Practical System Design.** We make heterogeneous denoising practical through the proposed UAD framework. Beyond Urgency-Aware Early Release (UAER) for latency reduction, UAD resolves the resulting early-release errors and trajectory inconsistency via two core components: Trajectory Reconciliation (TR), which restores joint state alignment with zero additional DiT evaluations, and Ghost Action Correction (GAC), which dynamically compensates for execution deviations.
- **Comprehensive Empirical Evaluation.** We evaluate UAD across diverse VLA architectures, simulation benchmarks, and real-world robotic manipulation tasks. In simulation, UAD achieves up to a $1.89\times$ speedup in average action availability latency by overlapping background denoising with physical execution, while preserving task success rates comparable to vanilla inference with optimal denoising step configuration.

2 Related Work

DiT Acceleration for VLA Policies. A broad range of acceleration techniques has been explored for diffusion and flow-matching VLA policies. One line of work reduces the per-step evaluation cost via model quantization (Xu et al., 2026; Zhang et al., 2026), layer skipping (Yang et al., 2026), pruning (Ji et al., 2026b; Wu et al., 2025), distillation (Prasad et al., 2024; Wang et al., 2025), and feature caching (Yang et al., 2025; Ji et al., 2026a; Reuss et al., 2025). These architectural techniques are orthogonal to our denoising step allocation. Beyond model-level acceleration, a more closely related line of work leverages the domain-specific physical and temporal properties of robotic tasks to adaptively orchestrate inference execution (Yu et al., 2025; Salla et al., 2026; Duan et al., 2025; Chun et al., 2025; Chen et al., 2025a). For instance, D3P (Yu et al., 2025) dynamically assigns denoising steps based on task-level action criticality, while POGP (Salla et al., 2026) employs adaptive early stopping based on predicted denoising gain. Concurrently, streaming architectures such as RTC (Black et al., 2025) and asynchronous execution frameworks (Kim et al., 2026b; Wang et al., 2026) hide latency by overlapping action generation with physical execution. In contrast to these approaches, UAD operates along a novel dimension: it exploits temporal execution urgency to allocate heterogeneous denoising budgets within a jointly generated action chunk, reducing action availability latency while systematically mitigating the resulting trajectory errors.

Heterogeneous Action Chunk Generation. Recent works have explored heterogeneous generation across different positions within an action chunk to improve temporal modeling, action quality, or real-time reactivity. Diffusion Forcing (Chen et al., 2024) introduced per-position diffusion states by allowing different tokens to follow distinct noise levels, while subsequent methods extended this idea to robotic control through progressively varying refinement levels across the action horizon (Høeg et al., 2025; Chen et al., 2025b; Jiang et al., 2025; Zhou et al., 2026). More recent approaches further exploit position-dependent noise schedules for reactive control and cross-horizon action updating (Park & Tulsiani, 2026; Kim et al., 2026a). While these approaches share with UAD the high-level idea of introducing intra-chunk heterogeneity, they differ fundamentally in purpose and mechanism. Existing methods manipulate per-position noise levels or flow trajectories primarily for causal modeling and horizon refinement. In contrast, UAD allocates heterogeneous denoising computation explicitly according to execution urgency, and introduces TR and GAC to mitigate the resulting trajectory inconsistencies and early-release errors.

3 Background

In this section, we introduce the generation and execution lifecycle of flow-matching VLA policies, and analyze the fundamental trade-off between denoising fidelity and inference latency.

Under a receding-horizon control paradigm, a flow-matching VLA policy repeatedly generates control trajectories conditioned on multi-modal context. At each decision step, the policy ingests the current observation o (comprising visual frames and sensory feedback) alongside a natural language instruction $\mathcal{L}$, and generates an action chunk $\mathbf{a} = [a^0, a^1, \ldots, a^{H-1}]$ containing H future control steps. The robot executes only the leading prefix of S actions ($S \leq H$), captures a fresh observation, and re-invokes the policy. Because these S executable actions are dispatched to physical hardware sequentially, earlier actions have greater execution urgency than later ones. This temporal asymmetry establishes distinct execution urgencies across actions within the same chunk.

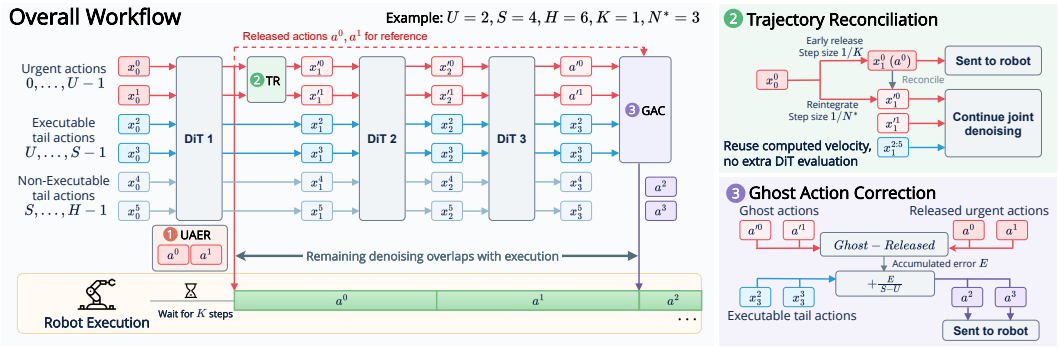


Figure 2: Overview of UAD

Generating an action chunk corresponds to one DiT inference, consisting of N sequential denoising steps. The policy initializes hidden state $\mathbf{x}_0 \sim \mathcal{N}(0, \mathbf{I})$, where $\mathbf{x}_k = [x_k^0, x_k^1, \dots, x_k^{H-1}]$ denotes the hidden state at denoising step k, and $k = 1, \dots, N$. At step k, the DiT backbone predicts a velocity field $\mathbf{v}_k$ conditioned on the current state and context[1]:

$$\mathbf{v}_k \leftarrow F_\theta(\mathbf{x}_{k-1}, o, \mathcal{L}), \tag{1}$$

where $\mathbf{v}_k = [v_k^0, v_k^1, \dots, v_k^{H-1}]$, and v_k^i denotes the predicted velocity for the i-th action. Under a uniform Euler discretization scheme, each action state is integrated as:

$$x_k^i = x_{k-1}^i + \frac{1}{N} v_k^i, \qquad i = 0, \dots, H-1. \tag{2}$$

After N updates, the terminal state $\mathbf{x}_N = [x_N^0, \dots, x_N^{H-1}]$ is released as the action chunk $\mathbf{a}$.

Increasing N does not monotonically yield better performance. Across diverse VLA architectures and benchmarks, task success rates exhibit non-monotonic behavior with N (see Appendix A). An empirically optimal budget $N^* \in [2, 7]$ can be identified as the minimal step achieving peak success rates in settings sensitive to N. Consequently, a reasonable deployment strategy fixes $N = N^*$ for all generated actions.

However, action generation latency scales virtually linearly with N, augmented by a fixed overhead from vision-language encoding (Appendix A). While aggressively truncating N from N^* down to a single step ($N = 1$) promises a 2–7× speedup in DiT inference, uniform step reduction degrades success rates by 3.9-10.7 pp on sensitive tasks. This tension reveals a critical mismatch: uniform denoising treats all actions equally, ignoring their heterogeneous temporal urgencies. Since physical execution only requires immediate access to the earliest actions, it is unnecessary to reduce denoising steps uniformly across the entire action chunk. Instead, we can selectively truncate the denoising steps for urgent actions to enable immediate release, while preserving optimal N^* for tail actions, accompanied by dedicated mechanisms to reconcile the resulting trajectory errors.

4 Method

In this section, we present UAD, an inference-time framework that exploits heterogeneous execution urgency within an action chunk to reduce action availability latency while preserving action quality.

4.1 Overview of Urgency-Aware Denoising

Fig. 2 illustrates the overall workflow of UAD. An action chunk generated by a VLA policy can be categorized into three functionally distinct segments based on execution urgency: **urgent actions** needed immediately for physical execution, **executable tail actions** that have some temporal buffer, and **non-executable tail actions** (indexed by $S \le i < H$, if $S < H$) that serve solely as temporal context. UAD exploits this temporal urgency hierarchy to accelerate action availability while maintaining full trajectory quality.

[1]We omit the flow-time argument t for simplicity; see Appendix C.1 for details.

Heterogeneous denoising, however, introduces two critical challenges. First, releasing urgent actions early after fewer denoising steps introduces *early-release errors* due to incomplete denoising. Second, as action tokens interact via shared attention within the DiT, early release disrupts the alignment of denoising procedure between urgent and tail actions, causing *trajectory inconsistency* that degrades subsequent tail action denoising.

UAD resolves these challenges through three integrated components, as structured in Fig. 2. **Urgency-Aware Early Release (UAER)** assigns a reduced denoising budget to urgent actions while retaining the full budget for tail actions, releasing urgent actions early to initiate physical execution and create an overlap window. **Trajectory Reconciliation (TR)** restores consistent joint state evolution prior to subsequent DiT evaluations. By re-integrating already computed velocities, TR reconstructs compatible internal states for urgent actions with zero additional DiT evaluations, mitigating or even eliminating *trajectory inconsistency*. **Ghost Action Correction (GAC)** compensates for accumulated *early-release errors*. As reconstructed urgent states continue joint denoising alongside the tail, they yield fully denoised **ghost actions**. GAC uses the deviation between ghost and released urgent actions to dynamically adjust the remaining executable tail actions. Together, these components allow UAD to substantially reduce action availability latency while preserving the quality and execution accuracy of the generated trajectory.

4.2 Urgency-Aware Early Release

Each inference generates an action chunk of length H, of which only the first S actions are executed before the next inference cycle is triggered. We designate the first U actions ($U < S \leq H$) as **urgent actions** and the remaining actions as **tail actions**, defining their index sets as

$$\mathcal{U} = \{0, \ldots, U-1\}, \qquad \mathcal{T} = \{U, \ldots, H-1\}. \tag{3}$$

The urgent actions are released early before full denoising completes, enabling their physical execution to overlap with the ongoing generation of the tail actions.

Under the vanilla policy, all actions undergo N^* denoising steps, where N^* is the empirically determined optimal setting. UAD instead assigns a reduced budget $K < N^*$ to urgent actions while retaining N^* steps for tail actions:

$$N_i = \begin{cases} K, & i \in \mathcal{U}, \\ N^*, & i \in \mathcal{T}. \end{cases} \tag{4}$$

Compared with the uniform Euler update introduced in Section 3, UAER preserves the same integration interval while assigning different numbers of denoising steps to urgent and tail actions, resulting in action-specific Euler step sizes. Specifically, the state of action i evolves as

$$x_k^i = x_{k-1}^i + \frac{1}{N_i} v_k^i, \qquad k = 1, \ldots, N_i. \tag{5}$$

Upon completing K denoising steps, the state x_K^i of each urgent action $i \in \mathcal{U}$ is released directly as the executable action $a^i = x_K^i$. Meanwhile, tail actions continue their denoising updates until all N^* steps are completed.

Latency Benefit. UAER releases urgent actions after only K denoising steps, allowing their physical execution to overlap with the remaining denoising for tail actions. When the execution duration of U urgent actions is sufficient to cover this remaining computation, the tail denoising latency is entirely masked. A formal latency model and detailed analysis are provided in Appendix B.

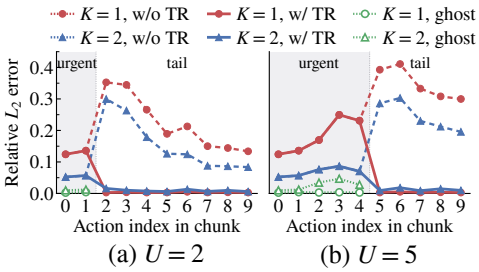
4.3 Trajectory Reconciliation

While UAER successfully reduces action availability latency, naively applying heterogeneous denoising severely degrades the quality of the entire action chunk. As empirically verified in Fig. 3, naive early release introduces substantial relative L_2 errors across both urgent and tail actions. First, urgent actions naturally deviate from their fully denoised counterparts due to reduced denoising steps. Second, because flow-matching VLA policies model the entire action chunk jointly via shared attention, operating urgent actions on a larger step size ($1/K$) disrupts joint state alignment, propagating divergence to tail actions and corrupting trajectory consistency. This highlights the vital need for a dedicated mechanism to reconcile internal trajectories before subsequent joint denoising.

To eliminate such misalignment, we introduce Trajectory Reconciliation (TR), which reconstructs compatible internal states for urgent actions prior to subsequent denoising. Since DiT predicts velocity fields v_k and action states are updated via Euler integration, previously computed velocities can be re-integrated under the canonical step size $1/N^*$. Specifically, for each urgent action $i \in \mathcal{U}$, TR computes a reconciled state x'^i_K after K steps as

$$x'^i_K = x^i_0 + \frac{1}{N^*} \sum_{k=1}^{K} v^i_k. \tag{6}$$

Crucially, this reconstruction requires **zero additional DiT evaluations**, as all velocity vectors v^i_k ($k \le K$) are already computed and cached.



(a) $U = 2$ (b) $U = 5$

Figure 3: Relative L_2 error to vanilla inference ($N^* = 3$). TR reduces tail action error while ghost states remain close to vanilla.

Following reconciliation, the complete state evolution across the action chunk is summarized as

$$\begin{cases} x^i_k = x^i_{k-1} + \frac{1}{K} v^i_k, & i \in \mathcal{U},\ k \le K, \\ x'^i_k = x'^i_{k-1} + \frac{1}{N^*} v^i_k, & i \in \mathcal{U},\ k > K, \\ x^i_k = x^i_{k-1} + \frac{1}{N^*} v^i_k, & i \in \mathcal{T}. \end{cases} \tag{7}$$

By feeding the reconciled states x'^i_K into subsequent DiT steps ($k > K$), TR reduces the joint-state misalignment before continued denoising. As demonstrated in Fig. 3, TR sharply reduces the tail action error across different U settings.

Exact Recovery for $K = 1$. When $K = 1$, since all actions share identical initial states and time conditions, the first DiT evaluation is identical to vanilla inference. Re-integrating this velocity with step size $1/N^*$ therefore exactly recovers the corresponding vanilla state, after which subsequent internal states follow the vanilla trajectory. Consequently, for $K = 1$, tail actions incur theoretically *zero additional error* relative to vanilla inference before applying GAC (Fig. 3). For $K > 1$, this exactness no longer holds because the cached velocities are evaluated along the heterogeneous early-release trajectory, and TR remains effective in substantially reducing tail action error empirically.

Ghost States and Ghost Actions. As released urgent actions are already dispatched for execution, their reconciled internal continuations x'^i_k ($k \ge K$) are not executed directly. We define these continued trajectories as **ghost states**, and denote their fully denoised final states as **ghost actions**: $a'^i \triangleq x'^i_{N^*}$ ($i \in \mathcal{U}$). Ghost states maintain necessary temporal context for tail actions during joint denoising, while providing a reference that we exploit next for mitigating early-release errors.

4.4 Ghost Action Correction

While TR addresses the trajectory inconsistency introduced to tail actions, the early-release errors inherent to urgent actions remain uncompensated. Since released urgent actions have already been dispatched for physical execution, they cannot be corrected retroactively. To mitigate this residual execution drift, we introduce Ghost Action Correction (GAC), which estimates the accumulated early-release error via ghost actions and dynamically compensates for it through subsequent executable tail actions.

Error Estimation via Ghost Actions. Correcting early-release error requires an accurate reference of the ideal, fully denoised trajectory. As verified in Fig. 3, Trajectory Reconciliation preserves joint temporal context, enabling ghost actions a'^i to closely track the vanilla references with remarkably low relative L_2 error (achieving exact zero error when $K = 1$). Consequently, a'^i serves as an ideal proxy for the uncorrupted actions. Since physical execution errors accumulate over time, we quantify the total early-release error vector $\mathbf{E}$ across all U urgent actions as

$$\mathbf{E} = \sum_{i=0}^{U-1} \left(a'^i - a^i\right) = \sum_{i=0}^{U-1} \left(x'^i_{N^*} - x^i_K\right), \tag{8}$$

where $a^i = x^i_K$ is the released urgent action and $a'^i = x'^i_{N^*}$ is the corresponding ghost action.

Dynamic Compensation across Executable Tail Actions. Although executed urgent actions are immutable, VLA actions encode continuous spatial commands (*e.g.*, Cartesian translation and axis-angle rotation). In robotic control, small execution deviations can be compensated through subsequent corrective motion commands (Shi et al., 2024; Wang et al., 2026). GAC leverages this physical compensability by uniformly distributing the accumulated error $\mathbf{E}$ over the remaining $S-U$ executable tail actions prior to their release. We adopt this simple allocation to preserve the total compensating displacement without introducing additional optimization or learned components:

$$a^i = x^i_{N^*} + \frac{\mathbf{E}}{S-U}, \qquad i = U, \ldots, S-1, \tag{9}$$

where $x^i_{N^*}$ denotes the denoised state of tail action i before correction. This adjustment leaves executed urgent actions untouched while compensating for the accumulated command discrepancy over the control horizon. In practice, this compensation is applied selectively based on physical semantics: additive dimensions (*e.g.*, Cartesian translation) follow Eq. 9, rotations are corrected through composition on $\mathrm{SO}(3)$, and decision-based dimensions (*e.g.*, gripper binary open/close) are masked to prevent unintended state toggling (detailed in Appendix C.1). GAC is applicable when action dimensions admit compensation across consecutive control steps. For action representations defined relative to a fixed or chunk-level reference, we omit GAC and retain UAER with TR only.

In conclusion, TR and GAC together enable UAD to retain the latency benefit of UAER while addressing the two main sources of action degradation introduced by heterogeneous denoising.

5 Evaluation

We evaluate UAD through simulation and real-world experiments, examining its effectiveness in reducing action availability latency while preserving task success rates. We implement UAD and baselines with the LeRobot framework (Cadene et al., 2026). Implementation details for the benchmarks, UAD, and baselines, including parameter settings, are provided in Appendix C. Additional evaluation results are provided in Appendix D.

5.1 Experiment Setup

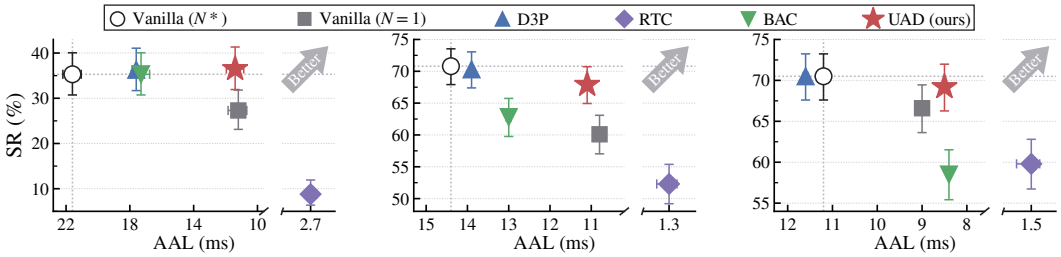
Models and Benchmarks. We evaluate UAD with two representative VLA models across simulation and real-world robotic manipulation. In simulation, we evaluate SmolVLA (Shukor et al., 2025) on LIBERO-Plus (Fei et al., 2025) and Meta-World+ (McLean et al., 2025), and $\pi_{0.5}$ (Physical Intelligence et al., 2025) on Meta-World+. We further deploy $\pi_{0.5}$ on a real robotic platform. The benchmark configuration follows our characterization of denoising step behavior in Appendix A. For the simulation experiments, we use pretrained checkpoints finetuned for the corresponding environments from the LeRobot framework or the community (LeRobot, c;d; tiantianx).

Baselines. We compare UAD with five baselines. **Vanilla (N^*)** uses the empirically optimal number of denoising steps for each setting, providing the reference for task success rate and latency. **Vanilla ($N = 1$)** uniformly reduces denoising to a single step, serving as a direct comparison with aggressive step reduction. We further compare UAD with three state-of-the-art acceleration methods: **D3P** (Yu et al., 2025), which dynamically selects denoising budgets; **RTC** (Black et al., 2025), which overlaps action generation with execution; and **BAC** (Ji et al., 2026a), which adaptively reuses intermediate features at the block level across denoising steps to reduce redundant computation.

Metrics. We evaluate task performance using **success rate (SR)**, defined as the fraction of evaluation episodes completed successfully, and inference latency using **average action availability latency (AAL)**, which measures the waiting time from requesting an action until it becomes available, averaged over all executed actions. AAL is therefore an action-level measure of exposed inference latency: given a fixed action duration $T_{\mathrm{act}} = 1/f_{\mathrm{ctrl}}$, it directly corresponds to an idle-time fraction of $\mathrm{AAL}/(\mathrm{AAL}+T_{\mathrm{act}})$. To facilitate comparison with the standard inference setting, we additionally report **ΔSR**, the change in success rate relative to Vanilla (N^*) in percentage points, and **speedup**, computed as the ratio of the AAL of Vanilla (N^*) to that of each method.

Table 1: Main results across three simulation settings. See **Metrics** for the definitions of metrics.

Method	SmolVLA + LIBERO-Plus				SmolVLA + Meta-World+				$\pi_{0.5}$ + Meta-World+			
	SR (%) ↑	AAL (ms) ↓	ΔSR (pp) ↑	Speedup (×) ↑	SR (%) ↑	AAL (ms) ↓	ΔSR (pp) ↑	Speedup (×) ↑	SR (%) ↑	AAL (ms) ↓	ΔSR (pp) ↑	Speedup (×) ↑
Vanilla (N^*)	35.3	21.6	0.0	1.00	70.8	14.4	0.0	1.00	70.5	11.2	0.0	1.00
Vanilla ($N = 1$)	27.3	11.2	−8.0	1.93	60.1	10.8	−10.7	1.33	66.6	9.0	−3.9	1.24
D3P	36.3	17.6	+1.0	1.23	70.3	13.9	−0.5	1.04	70.5	11.6	0.0	0.97
RTC	8.8	2.7	−26.5	8.00	52.3	1.3	−18.5	11.08	59.8	1.5	−10.7	7.47
BAC	35.3	17.3	0.0	1.25	62.8	13.0	−8.0	1.11	58.5	8.4	−12.0	1.33
UAD (ours)	36.5	11.4	+1.2	1.89	67.9	11.1	−2.9	1.30	69.2	8.5	−1.3	1.32



(a) SmolVLA + LIBERO-Plus (b) SmolVLA + Meta-World+ (c) $\pi_{0.5}$ + Meta-World+

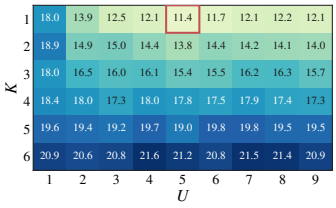
Figure 4: Success rate-latency trade-offs across three simulation settings. UAD achieves low AAL while retaining high SR. Broken horizontal axes accommodate RTC's substantially lower AAL. Error bars indicate Wilson 95% CIs for SR, and episode bootstrap 95% CIs for AAL. Some AAL CIs are too narrow to be visible at the plotted scale.

Settings. Following the convention of existing research or the default LeRobot configuration, we use an action chunk size of $H = 50$ and execute $S = 10$ actions before replanning. For UAD, we set $K = 1$ to minimize urgent-action latency while enabling lossless recovery of tail and ghost actions. We choose the smallest U for which urgent-action execution fully overlaps the remaining tail denoising. All VLA inference and latency measurements are performed on an NVIDIA H20 GPU. Additional hardware and configuration details are provided in Appendix C.

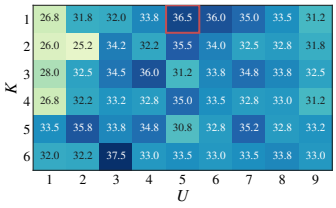
5.2 Results and Analysis

Overall Performance. As shown in Table 1, in three simulation settings, UAD achieves 1.30–1.89× speedup in AAL over Vanilla (N^*), with SR changes ranging from −2.9 to +1.2 pp. Compared with Vanilla ($N = 1$), UAD improves SR by 9.2, 7.8, and 2.6 pp, respectively, while achieving comparable AAL. These results demonstrate that allocating denoising steps according to action urgency mitigates the performance loss caused by single-step inference. D3P adopts conservative denoising step selection in the experiments, largely preserving SR but yielding only limited AAL speedups (0.97-1.23×), whereas UAD achieves larger speedups across all three settings. Compared with BAC, UAD consistently achieves higher SR with lower or comparable AAL. RTC achieves very low AAL by fully overlapping VLA computation with action execution. However, its SR drops substantially by 10.7–26.5 pp relative to Vanilla (N^*) in the evaluated settings. Overall, UAD offers a favorable balance between action availability latency and task success, as is shown in Fig. 4.

Parameter Study. We study the two key parameters of UAD, K and U, on SmolVLA with LIBERO-Plus, whose relatively large $N^* = 7$ provides a broad design space. We evaluate all feasible settings ($K = 1, \ldots, 6$ and $U = 1, \ldots, 9$), with results shown in Fig. 5. Reducing K lowers AAL by shortening the blocking time for urgent actions. Increasing U allows urgent action execution to hide more tail denoising, but provides little additional latency benefit once the remaining computation is fully overlapped. For SR, very small or large U tends to underperform intermediate values, particularly at small K. Meanwhile, increasing K yields no consistent SR improvement. With TR



(a) AAL.



(b) SR.

Figure 5: Parametric study of UAD under different urgent-action counts U and urgent denoising depths K. The applied setting is marked by a red square.

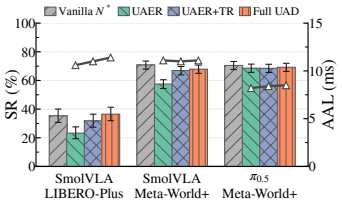


Figure 6: Ablation study of the components in UAD. Bars: SR; markers: AAL; error bars: Wilson 95% CI.

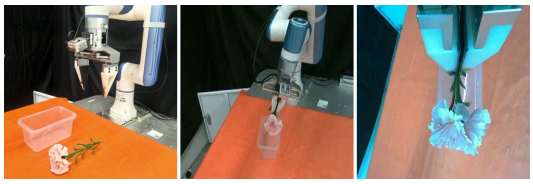


Figure 7: Real-world deployment on Dobot Nova2.

Table 2: Real-world deployment results.

Method	SR (%)↑	AAL (ms)↓
Vanilla ($N = 10$)	84	34.3
Vanilla ($N = 1$)	71	14.9
UAD	84	15.2

and GAC, $K = 1$ maintains consistently strong SR across a range of intermediate U values while minimizing AAL, supporting our choice of $K = 1$ with U covering the remaining computation.

Ablation Study. UAD comprises urgency-aware early release (UAER), trajectory reconciliation (TR), and ghost action correction (GAC). UAER accelerates inference but may degrade action quality. TR addresses tail-action errors, while GAC uses ghost actions obtained through TR to further preserve action quality. We compare UAER alone, UAER+TR, and full UAD in Fig. 6. In both SmolVLA settings, UAER reduces SR, TR partially recovers the loss, and GAC further restores SR to a level comparable to Vanilla (N^*). For $\pi_{0.5}$ on Meta-World+, UAER causes little SR degradation, leaving limited room for TR and GAC to improve performance, consistent with their role in recovering behavior under N^* denoising. AAL remains similar across variants in all settings, indicating negligible additional exposed latency from TR and GAC.

Real-World Deployment. As shown in Fig. 7, we deploy UAD on a Dobot Nova2 robot using finetuned $\pi_{0.5}$ checkpoints across five tasks. We compare against vanilla inference with $N = 10$, the default deployment configuration, and $N = 1$, an aggressive acceleration baseline. As shown in Table 2, UAD reduces AAL from 34.3 to 15.2 ms, closely matching the 14.9 ms achieved with $N = 1$ and yielding a 2.26× speedup. UAD also maintains the default baseline's 84% success rate. In contrast, $N = 1$ reduces success to 71%. These results demonstrate that UAD translates its latency benefits to real-world control while preserving task performance.

6 Conclusion

We introduced **Urgency-Aware Denoising (UAD)**, an inference-time framework for accelerating flow-matching VLA policies by exploiting the heterogeneous execution urgency within an action chunk. Rather than uniformly allocating denoising computation to actions that are consumed at different times, UAD releases urgent actions early and overlaps the remaining denoising with physical execution, while Trajectory Reconciliation (TR) and Ghost Action Correction (GAC) preserve the quality of the resulting trajectory. Across multiple VLA models and robotic manipulation benchmarks, UAD consistently reduces action availability latency while maintaining success rates comparable to vanilla inference, achieving up to a 1.89× speedup in simulation and similarly strong latency reductions in real-world deployment. More broadly, our results suggest that *when* an action is needed provides an important dimension for allocating inference computation, offering a complementary direction for improving the real-time efficiency of VLA control.

AI USE STATEMENT

In this work, we used generative AI tools for designing or providing feedback on research methodology or experiments, implementing methods, interpreting results, and assisting with translation. We did not use generative AI tools for generating synthetic datasets, developing the core theoretical or conceptual framework of the work, formulating mathematical claims, or proposing or refining research hypotheses. Tasks related to providing critical ingredients for mathematical proofs, assisting with proof writing, cleaning or reformatting datasets, and qualitative or thematic data analysis were not applicable to this work. Additionally, we used generative AI tools to edit the background of a photograph of the robotic platform for use in Fig. 1, create or edit software code, assist with drafting and editing parts of the manuscript, summarize and identify relevant literature, source information, and suggest improvements to the structure and readability of the paper. We reviewed all AI-assisted work. All content ultimately included in the paper was decided and evaluated by the authors, and AI-generated content was used only as reference. All AI-generated code used in the implementation was verified by the authors for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce our method and experimental results throughout the main paper and appendix. The formulation and inference procedure of UAD are described in Section 4, with detailed implementation procedures provided in Appendix C.1. The implementation and adaptation of the baselines are described in Appendix C.2. The evaluation protocols, configurations, and testbed implementation details are provided in Appendices C.3 and C.4 for the simulated environments and real-world deployment, respectively. Anonymous source code and evaluation scripts of the simulated environments are provided in the supplementary materials.

REFERENCES

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

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

Remi Cadene, Simon Alibert, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, et al. LeRobot: An open-source library for end-to-end robot learning. In *Proceedings of the International Conference on Learning Representations*, 2026.

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

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 International Conference on Machine Learning*, 2025a.

Zhuoqun Chen, Xiu Yuan, Tongzhou Mu, and Hao Su. Responsive noise-relaying diffusion policy: Responsive and efficient visuomotor control. *Transactions on Machine Learning Research*, 2025b.

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

Inkook Chun, Seungjae Lee, Michael Albergo, Saining Xie, and Eric Vanden-Eijnden. Dynamic test-time compute scaling in control policy: Difficulty-aware stochastic interpolant policy. In *Advances in Neural Information Processing Systems*, 2025.

Yufei Duan, Hang Yin, and Danica Kragic. Real-time iteration scheme for diffusion policy. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2025.

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

Sigmund H Høeg, Yilun Du, and Olav Egeland. Fast policy synthesis with variable noise diffusion models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025.

Kangye Ji. Block-wise adaptive caching for diffusion policy: Opensource implementation. Github. URL `https://github.com/ky-ji/BAC/`.

Kangye Ji, Yuan Meng, Hanyun Cui, Ye Li, Jianbo Zhou, Shengjia Hua, Lei Chen, and Zhi Wang. Block-wise adaptive caching for accelerating diffusion policy. In *Proceedings of the International Conference on Learning Representations*, 2026a.

Kangye Ji, Jianbo Zhou, Yuan Meng, Ye Li, Hanyun Cui, and Zhi Wang. Sparse actiongen: Accelerating diffusion policy with real-time pruning. In *Proceedings of the International Conference on Machine Learning*, 2026b.

Sunshine Jiang, Xiaolin Fang, Nicholas Roy, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Siddharth Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories. In *Proceedings of the Conference on Robot Learning*, 2025.

Seonsoo Kim, Seongil Hong, and Jun-Gill Kang. Diffusion reroll: Revisable denoising for robotic sequential prediction. *arXiv preprint arXiv:2607.19919*, 2026a.

Sunwoo Kim, Jeongjun Kim, and Joseph Lim. Time optimal execution of action chunk policies beyond demonstration speed. In *Proceedings of the International Conference on Learning Representations*, 2026b.

LeRobot. pi05_libero_finetuned: Model checkpoint. Hugging Face, a. URL `https://huggingface.co/lerobot/pi05_libero_finetuned`.

LeRobot. smolvla_libero_finetuned: Model checkpoint. Hugging Face, b. URL `https://huggingface.co/lerobot/smolvla_libero`.

LeRobot. smolvla_libero_plus_finetuned: Model checkpoint. Hugging Face, c. URL `https://huggingface.co/lerobot/smolvla_libero_plus`.

LeRobot. smolvla_metaworld_finetuned: Model checkpoint. Hugging Face, d. URL `https://huggingface.co/lerobot/smolvla_metaworld`.

Reginald McLean, Evangelos Chatzaroulas, Luc McCutcheon, Frank Röder, Tianhe Yu, Zhanpeng He, K. R. Zentner, Ryan Julian, J. K. Terry, Isaac Woungang, Nariman Farsad, and Pablo Samuel Castro. Meta-world+: An improved, standardized, RL benchmark. In *Advances in Neural Information Processing Systems*, 2025.

Khanh D Nguyen, Hung T Ho, Chinh T Nguyen, Thanh Q Duong, Linh D Le, Duy MH Nguyen, Vien A Ngo, and An T Le. vla. cpp: A unified inference runtime for vision-language-action models. *arXiv preprint arXiv:2606.08094*, 2026.

Sungjae Park and Shubham Tulsiani. πr^2: Reactive real-time flow policies. *arXiv preprint arXiv:2607.26055*, 2026.

Xiaohuan Pei, Yuxing Chen, Siyu Xu, Yunke Wang, Yuheng Shi, and Chang Xu. Action-aware dynamic pruning for efficient vision-language-action manipulation. In *Proceedings of the International Conference on Learning Representations*, 2026.

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

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

Moritz Reuss, Jyothish Pari, Pulkit Agrawal, and Rudolf Lioutikov. Efficient diffusion transformer policies with mixture of expert denoisers for multitask learning. In *Proceedings of the International Conference on Learning Representations*, 2025.

Rohit Kumar Salla, Manoj Saravanan, and Simon Stepputtis. Learning when to stop: Prefix-optimal dynamic diffusion policies for continuous control. *arXiv preprint arXiv:2608.05084*, 2026.

Lucy Xiaoyang Shi, Zheyuan Hu, Tony Z. Zhao, Archit Sharma, Karl Pertsch, Jianlan Luo, Sergey Levine, and Chelsea Finn. Yell at your robot: Improving on-the-fly from language corrections. In *Proceedings of Robotics: Science and Systems*, 2024.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andrés Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Rémi Cadène. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

tiantianx. pi05_metaworld_finetuned: Model checkpoint. Hugging Face. URL `https://huggingface.co/tiantianx/pi05_metaworld`.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. In *Proceedings of the International Conference on Learning Representations*, 2026.

Zhendong Wang, Max Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, Ming-Yu Liu, and Yu Zeng. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. In *Proceedings of the International Conference on Machine Learning*, 2025.

Yiming Wu, Huan Wang, Zhenghao Chen, Jianxin Pang, and Dong Xu. On-device diffusion transformer policy for efficient robot manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

Yuhao Xu, Yantai Yang, Zhenyang Fan, Yufan Liu, Yuming Li, Bing Li, and Zhipeng Zhang. Qvla: Not all channels are equal in vision-language-action model's quantization. In *Proceedings of the International Conference on Learning Representations*, 2026.

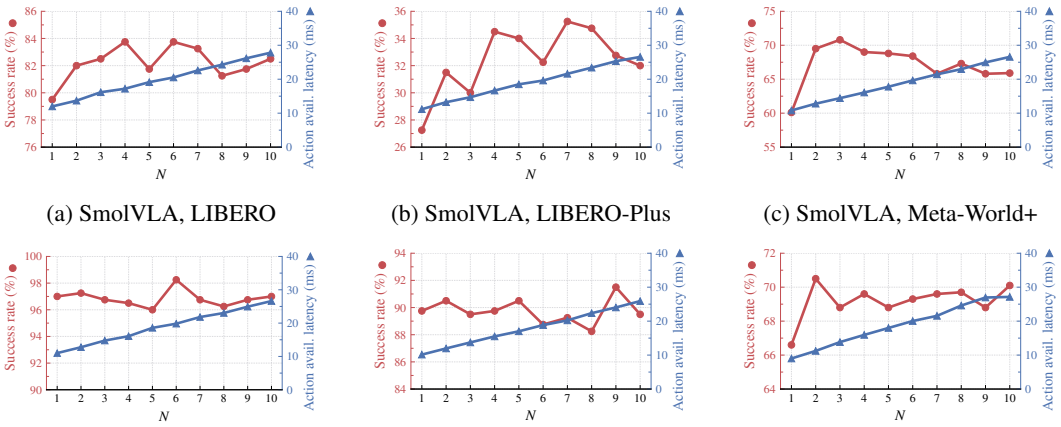
Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. In *Advances in Neural Information Processing Systems*, 2025.

Zebin Yang, Yijiahao Qi, Tong Xie, Bo Yu, Shaoshan Liu, and Meng Li. Dysl-vla: Efficient vision-language-action model inference via dynamic-static layer-skipping for robot manipulation. *arXiv preprint arXiv:2602.22896*, 2026.

Shu-Ang Yu, Feng Gao, Yi Wu, Chao Yu, and Yu Wang. D3p: Dynamic denoising diffusion policy via reinforcement learning. *arXiv preprint arXiv:2508.06804*, 2025.

Jingxuan Zhang, Yunta Hsieh, Zhongwei Wan, Haokun Lin, Xin Wang, Ziqi Wang, Yingtie Lei, and Mi Zhang. Quantvla: Scale-calibrated post-training quantization for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Zhexuan Zhou, Yichen Lai, Jinhao Zhang, Huizhe Li, Youmin Gong, and Jie Mei. Latent diffusion policy: Shaping latent spaces for diffusion-based robotic manipulation. *arXiv preprint arXiv:2606.08657*, 2026.



(a) SmolVLA, LIBERO (b) SmolVLA, LIBERO-Plus (c) SmolVLA, Meta-World+

(d) $\pi_{0.5}$, LIBERO (e) $\pi_{0.5}$, LIBERO-Plus (f) $\pi_{0.5}$, Meta-World+

Figure 8: Task success rate (red circles, left axis) and action availability latency (blue triangles, right axis) for different numbers of denoising steps N across six model-benchmark settings.

A Effect of the Number of Denoising Steps

We evaluate SmolVLA and $\pi_{0.5}$ on LIBERO, LIBERO-Plus, and Meta-World+, yielding six model-benchmark settings. For each setting, we use a checkpoint finetuned on the corresponding benchmark (LeRobot, b;c;d;a; tiantianx), and vary the number of uniform denoising steps N from 1 to 10. We measure both task success rate and action availability latency. These experiments characterize how the denoising budget affects the performance-latency trade-off and guide the selection of model-benchmark settings and the optimal denoising steps N^* used in our main evaluation.

Task Performance and Benchmark Setting. Fig. 8 shows that the effect of the denoising budget strongly depends on the model-benchmark combination. When a model is sufficiently strong for a relatively easy benchmark, task performance can become saturated and consequently insensitive to N. For example, $\pi_{0.5}$ achieves consistently high success rates on LIBERO across the entire sweep, making changes in denoising quality difficult to distinguish from the task metric alone. Such saturated settings provide limited information for studying the trade-off between denoising steps and action quality. For each setting, we define N^* as the smallest N attaining the highest success rate and use it as the vanilla reference. We therefore focus our main evaluation on settings that remain non-saturated and exhibit a clear sensitivity to the denoising steps: SmolVLA on LIBERO-Plus ($N^* = 7$) and Meta-World+ ($N^* = 3$), and $\pi_{0.5}$ on Meta-World+ ($N^* = 2$). In these settings, success rate is non-monotonic in N, indicating that simply increasing the number of denoising steps does not necessarily improve task performance. Reducing denoising steps uniformly from N^* to $N = 1$ decreases success rate by 3.9-10.7 pp across these settings. This establishes a regime where reducing denoising computation reduces latency but noticeably degrades action quality. Results for the three additional settings not selected for the main evaluation are presented in Appendix D.4.

Latency. Across all six settings, average action availability latency increases approximately linearly with N, with a nonzero intercept. This trend reflects the combination of repeated DiT evaluations and fixed inference overhead, including VLM context processing. Consequently, reducing N directly lowers denoising latency, but the fixed overhead limits the corresponding reduction in end-to-end action availability latency. Together with the performance results above, this motivates reducing the computation exposed before execution rather than uniformly reducing the denoising steps of the entire action chunk.

B Latency Analysis of Urgency-Aware Early Release

We formulate a simplified timing model to explain how UAER reduces action availability latency by overlapping denoising computation with action execution. Let T_{VLM} denote the latency of vision-

language context encoding, T_D the latency of a single DiT backbone evaluation, and T_{act} the execution duration of one action. Each control cycle executes S actions before invoking the policy again. The analysis assumes constant computation and action execution times, ideal overlap between computation and execution, and no additional scheduling or synchronization overhead. For simulation, T_{act} corresponds to the fixed overlap budget assigned to each action.

Vanilla Inference. Under vanilla inference, all H actions undergo N^* denoising steps and become available simultaneously. The controller waits for VLM encoding and all N^* DiT evaluations before execution begins. Averaged over the S executed actions, the action availability latency is:

$$T_{\text{avail,vanilla}} = \frac{T_{\text{VLM}} + N^* T_D}{S}. \tag{10}$$

Urgency-Aware Early Release. UAER releases the first U urgent actions after $K < N^*$ denoising steps, allowing execution to begin after $T_{\text{VLM}} + KT_D$. Executing these urgent actions provides an overlap window of UT_{act} for the remaining $(N^* - K)$ DiT evaluations. The controller incurs additional waiting only if tail denoising exceeds this window. The resulting average action availability latency is:

$$T_{\text{avail,UAER}} = \frac{T_{\text{VLM}} + KT_D + \max\left((N^* - K)T_D - UT_{\text{act}},\, 0\right)}{S}. \tag{11}$$

Thus, K determines the initial waiting time, whereas U determines the execution window available to overlap the remaining denoising. When $UT_{\text{act}} \geq (N^* - K)T_D$, tail denoising is fully hidden under this model, reducing the average action availability latency to $(T_{\text{VLM}} + KT_D)/S$.

Practical Measurement and Parameter Selection. The timing model provides intuition for latency reduction, while actual latency depends on runtime overheads and timing variability. We therefore base our empirical evaluation on measured AAL and blocking latency rather than the analytical estimates. For each model-benchmark setting, we select the smallest U whose measured average tail blocking time is below 0.2 ms, as detailed in Appendix C.1. This criterion keeps residual waiting negligible while limiting the number of actions released with fewer denoising steps.

C Implementation and Evaluation Details

C.1 UAD Implementation

We implement UAD within LeRobot for SmolVLA and $\pi_{0.5}$ by modifying their inference procedures, without introducing additional trainable parameters or requiring further training.

Per-Action Time Conditioning. Standard flow-matching inference conditions all action tokens on a shared flow time t. To support heterogeneous denoising schedules while retaining a single joint DiT forward pass, we extend the models to accept a separate time condition for each action token. For SmolVLA, the corresponding time embedding is provided to each action token before the existing action MLP; for $\pi_{0.5}$, it conditions the adaptive normalization layers of the action expert. These modifications reuse the pretrained weights and introduce no additional trainable parameters.

Asynchronous Execution. Once the urgent actions complete their assigned denoising steps, we synchronize the GPU and release them to the execution queue. Tail denoising then continues asynchronously on a separate CUDA stream while the controller executes the available actions. If tail computation is not completed before the urgent actions are consumed, execution waits for the remaining actions. After S actions are executed, the controller replans from the latest observation.

TR and GAC. TR caches the initial noise and previously predicted velocities to reconstruct the internal urgent-action states without additional DiT evaluations, while keeping the released actions unchanged. GAC then uses the resulting ghost actions to correct the remaining executable actions, also without additional DiT evaluations. Corrections are applied in the appropriate action dimensions for each benchmark; in particular, rotational discrepancies are handled through composition on $\mathrm{SO}(3)$ rather than direct Euclidean subtraction.

Parameter Configuration. Unless otherwise specified, we use $H = 50$, $S = 10$, and set the urgent denoising depth to $K = 1$. For each model-benchmark setting, we evaluate $U \in \{1, \ldots, S-1\}$ and measure the average blocking time of tail actions, as shown in Fig. 9. We select the smallest U that reduces this average waiting time below 0.2 ms, allowing the remaining denoising computation to overlap with urgent action execution with negligible residual blocking. This yields $U = 5$ for SmolVLA on LIBERO-Plus, $U = 3$ for SmolVLA on Meta-World+, and $U = 2$ for $\pi_{0.5}$ on Meta-World+. Tail actions use the corresponding denoising budget N^* selected in Appendix A.

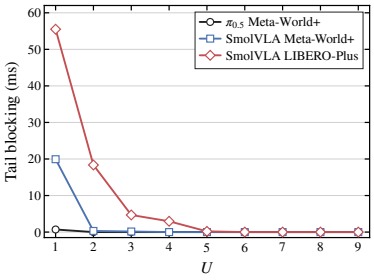


Figure 9: Average tail blocking time for different values of U at $K = 1$.

C.2 Baseline Implementations

All baselines use the same pretrained VLA checkpoints and observation/action processing pipelines as UAD. Unless otherwise specified, we use the same action horizon $H = 50$ and execution interval $S = 10$. Baseline-specific adaptations and configurations are described below.

Vanilla. Vanilla (N^*) and Vanilla ($N = 1$) use the standard LeRobot inference with different denoising steps. Both generate the complete action chunk using Euler integration and execute the first S actions before replanning. The two variants use N^* and one denoising step, respectively.

BAC. We implement BAC following its official open-source implementation (Ji). We retain its ACS and BUA mechanisms for adaptive recomputation and residual-cache updates, and adapt them to flow-matching VLAs by indexing the caching schedule over Euler steps and applying residual caching within the action expert. Caches are reset for every action chunk. We retain N^* Euler steps and evaluate all ACS recomputation budgets $m \in \{1, \ldots, N^* - 1\}$. Among configurations that reduce AAL relative to vanilla inference, $m = \lceil N^*/2 \rceil$ achieves the highest SR in all three simulation settings and is therefore used in our main comparison. BUA is enabled for the five highest-drift blocks. The caching schedule is calibrated offline on held-out rollouts using seeds distinct from evaluation, without additional policy training.

RTC. We use the original inference-time guided RTC implementation provided in LeRobot, following its standard inference design and configuration. We retain its flow-matching guidance and asynchronous action-queue update. Inference proceeds concurrently with action execution; when a new chunk becomes available, actions that became stale during inference are discarded before the remaining actions are inserted into the queue.

D3P. We reproduce D3P based on the original paper. We adapt its learned denoising-step selection to pretrained flow-matching VLAs. We let the VLA policy remain frozen, while a lightweight adaptor is trained with PPO to dynamically select the number of Euler intervals advanced at each denoising iteration. Specifically, we divide the flow trajectory into $M = N^*$ uniform intervals. At each denoising iteration, the adaptor selects an integer stride between one and the number of remaining intervals, and one velocity evaluation advances the action state across the selected range using the corresponding Euler step size. We train a separate adaptor for each model-benchmark setting on a single NVIDIA H20 GPU for eight hours. The reward combines task performance with a computation penalty based on the number of velocity evaluations, replacing the original diffusion-specific computation objective with its flow-matching counterpart. At evaluation time, we use the final checkpoint and include adaptor inference overhead in the reported AAL.

C.3 Simulation Evaluation Protocol

Benchmarks and Episodes. For LIBERO-Plus, we evaluate a fixed subset of 400 variants, with 100 sampled from each of the Spatial, Object, Goal, and Long suites, evenly covering all seven variation categories. We run one episode per variant using its designated initial state and the standard LeRobot evaluation horizon. Each episode begins with 10 settling steps that are excluded from evaluation. For Meta-World+, we evaluate 50 tasks with 20 episodes per task, yielding 1,000 episodes in

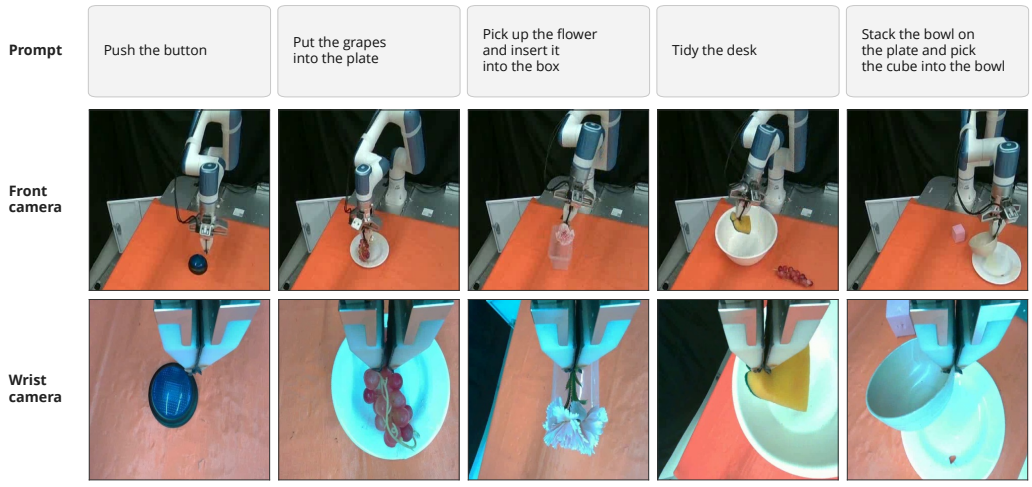


Figure 10: Real-world evaluation tasks and camera observations. We evaluate five manipulation tasks using front lateral and wrist-mounted camera views. Camera views are cropped.

total. Initial conditions vary across episodes. All methods use identical evaluation tasks and episode configurations, with a base random seed of 42. Episodes terminate upon environment-defined success or when the maximum horizon is reached.

Control-Time Alignment. The wall-clock duration of a simulation step can differ substantially from the physical execution duration of an action. Without explicit alignment, slow simulation or rendering could artificially enlarge the computation window available to asynchronous methods. We therefore assign each executed action a fixed overlap budget of $T_{\text{act}} = 1/f_{\text{ctrl}}$, corresponding to 50 ms for LIBERO-Plus at 20 Hz and 12.5 ms for Meta-World+ at 80 Hz. For methods that overlap inference with execution, including UAD and RTC, background computation is limited by this prescribed budget rather than by simulator wall-clock time. Consequently, each action provides exactly one T_{act} of potential overlap, independent of how quickly or slowly the simulator executes the corresponding step.

Latency Measurement. We measure AAL as the delay from requesting an action until it becomes available to the controller, averaged over executed actions. The measurement includes exposed policy computation and waiting for unavailable actions, while excluding computation hidden behind action execution. Model loading, environment stepping, and rendering are excluded, and GPU synchronization is used to measure completed computation. All VLA inference is performed on NVIDIA H20 GPUs, with each inference process running on a single GPU. For latency measurement, we use one evaluation process on one exclusive H20 GPU with an inference batch size of one. We evaluate all 50 Meta-World+ tasks and eight representative LIBERO-Plus variants spanning the four suites for latency measurement. We report steady-state AAL, excluding the first episode of each evaluation process as warm-up for all methods. This exclusion applies only to latency, and success rates are computed using all evaluation episodes. We compute 95% confidence intervals by resampling episodes with replacement (8,000 replicates) and taking the 2.5th and 97.5th percentiles of the bootstrap distribution.

C.4 Real-World Deployment

Hardware Setup. We conduct real-world experiments on a Dobot Nova 2 robotic manipulation platform equipped with a front lateral camera (D455) and a wrist-mounted camera (D405). The two cameras use the existing calibrated configuration of the shared robotic platform, with the front-lateral view providing global workspace observations and the wrist view capturing close-range information around the end effector. VLA inference is performed on a single NVIDIA H20 GPU. The policy takes both camera views together with robot proprioception as input. All compared methods share the same robot platform, observations, action interface, and pretrained $\pi_{0.5}$ policy.

Table 3: Action availability latency (AAL) and DiT blocking latency (DBL) for the evaluated settings and methods.

Method	SmolVLA LIBERO-Plus		SmolVLA Meta-World+		$\pi_{0.5}$ Meta-World+		Real-World Deployment	
	AAL (ms) ↓	DBL (ms) ↓	AAL (ms) ↓	DBL (ms) ↓	AAL (ms) ↓	DBL (ms) ↓	AAL (ms) ↓	DBL (ms) ↓
Vanilla (N^*)	21.6	120.3	14.4	52.4	11.2	47.1	34.3	216.6
Vanilla ($N = 1$)	11.2	18.2	10.8	17.9	9.0	24.6	14.9	19.9
D3P	17.6	122.2	13.9	53.7	11.6	36.1	-	-
RTC	2.7	9.6	1.3	5.2	1.5	7.0	-	-
BAC	17.3	81.3	13.0	38.7	8.4	21.9	-	-
UAD (ours)	11.4	19.7	11.1	19.4	8.5	19.8	15.2	22.3

Tasks. We evaluate five tabletop manipulation tasks covering diverse interaction patterns, as illustrated in Fig. 10. Each task is specified by a natural-language instruction and requires the policy to coordinate the front-lateral and wrist-camera observations. We run 20 episodes per task, for a total of 100 real-world evaluation episodes. Together, the tasks cover contact interaction, pick-and-place, object rearrangement, and multi-stage manipulation.

Deployment and Evaluation. We deploy UAD on the pretrained $\pi_{0.5}$ policy. In this real-world setup, each action is represented as a relative change with respect to the beginning of the action chunk, rather than incrementally with respect to the preceding action. The additive compensation in GAC therefore does not apply to this action representation. Accordingly, our UAD deployment uses UAER together with TR, without GAC. We compare UAD against Vanilla ($N = 10$) and Vanilla ($N = 1$) under identical hardware and control settings. We measure success rate manually, and AAL from the delay between an action request and the corresponding action becoming available for execution. Unlike in simulation, no control-time alignment is required: denoising computation overlaps directly with the physical execution of actions, and the resulting overlap window is determined by the robot's actual execution time.

D Extra Evaluation Results

In this part, we present extra evaluation results not presented in Section 5.

D.1 Blocking Latency of DiT

VLA inference latency consists of two main components: a largely fixed context-processing overhead, dominated by the VLM, and the iterative DiT computation for action generation. The average action availability latency (AAL) reported in Section 5 measures the end-to-end waiting time for actions and therefore accounts for both components. However, UAD and most of the compared baselines primarily optimize the DiT computation, while leaving the VLM and other fixed overheads largely unchanged. We therefore additionally report a DiT-specific latency metric to more directly characterize the reduction in denoising time that blocks action execution. Specifically, we define **DiT blocking latency (DBL)** as the average amount of DiT computation per VLA inference that directly blocks action execution. DiT computation that is successfully overlapped with physical action execution does not contribute to DBL.

Table 3 reports both AAL and DBL across the simulation settings and our real-world deployment. UAD substantially reduces DBL in all evaluated settings. Notably, the relative reduction in DBL is consistently larger than that in AAL. This is expected because AAL additionally includes VLM processing and other fixed overheads that UAD, as well as most competing methods, does not optimize. Consequently, DBL more directly exposes the reduction in action-blocking DiT time, further demonstrating the effectiveness of UAD in reducing action waiting time.

Table 4: Success rate (%) on Meta-World+ by task difficulty (easy / medium / hard / very hard: 28 / 11 / 6 / 5 tasks, 20 episodes each).

	SmolVLA					$\pi_{0.5}$				
Method	Easy	Medium	Hard	Very hard	Overall	Easy	Medium	Hard	Very hard	Overall
Vanilla (N^*)	85.2	56.4	50.0	47.0	70.8	80.7	58.6	68.3	42.0	70.5
Vanilla ($N = 1$)	76.1	45.5	38.3	29.0	60.1	79.8	54.1	55.0	34.0	66.6
D3P	84.1	57.7	50.8	44.0	70.3	80.2	61.8	68.3	38.0	70.5
RTC	63.2	47.7	30.8	27.0	52.3	69.8	52.3	49.2	33.0	59.8
BAC	80.4	45.9	38.3	31.0	62.8	75.9	39.5	44.2	20.0	58.5
UAD (ours)	80.4	55.5	51.7	45.0	67.9	80.0	58.2	67.5	35.0	69.2

Table 5: Success rate (%) on individual real-world manipulation tasks. Each task is evaluated over 20 episodes. Detailed task descriptions are provided in Fig. 10.

Method	Button	Grapes	Flower	Tidy	Stack	Overall
Vanilla ($N = 10$)	100	85	65	95	75	84
Vanilla ($N = 1$)	100	70	55	75	55	71
UAD (ours)	100	90	65	90	75	84

D.2 Meta-World+ Difficulty Breakdown

To further examine whether acceleration affects tasks of different difficulty differently, following the official breakdown, we separate the Meta-World+ tasks into difficulties of easy, medium, hard, and very hard, and present the results in Table 4. Overall, the performance degradation introduced by aggressive acceleration tends to become more pronounced on more difficult tasks. For example, uniformly reducing denoising to $N = 1$, as well as RTC and BAC, causes substantially larger losses on the hard and very-hard subsets than on easy tasks in many cases. This suggests that challenging tasks are generally more sensitive to degradation in action quality. In contrast, UAD maintains competitive success rates across difficulty levels. These results indicate that the favorable success-latency trade-off of UAD extends to challenging tasks and is not driven solely by performance on any difficulty group.

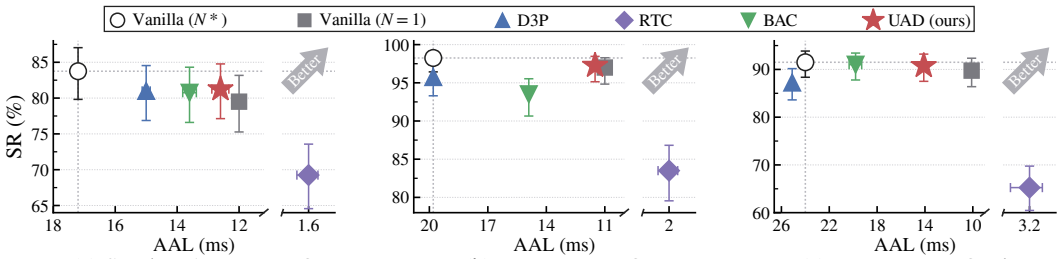
D.3 Real-World Task Breakdown

Our real-world evaluation consists of five manipulation tasks, and Table 5 reports the success rate for each task separately. Uniformly reducing the denoising budget to $N = 1$ degrades performance on four of the five tasks, reducing the overall success rate from 84% to 71%. In contrast, UAD maintains an overall success rate of 84% and remains close to Vanilla ($N = 10$) across all five tasks, indicating consistent performance preservation across different real-world manipulation behaviors.

D.4 Evaluation on Additional Model-Benchmark Settings

We extend our comparison to the three settings not included in the main evaluation: SmolVLA on LIBERO, and $\pi_{0.5}$ on LIBERO and LIBERO-Plus, using the same evaluation protocol. As characterized in Appendix A, these settings exhibit relatively small success rate differences between Vanilla ($N = 1$) and Vanilla (N^*), making uniform single-step denoising a simple and effective acceleration strategy.

As shown in Fig. 11, Vanilla ($N = 1$) achieves lower AAL than UAD with only modestly lower success rates, leaving limited room for UAD to improve upon uniform step reduction in these settings. Nevertheless, UAD consistently reduces AAL relative to Vanilla (N^*) while maintaining comparable success rates. Compared with D3P and BAC, UAD achieves lower AAL with comparable or higher success rates. RTC attains substantially lower AAL but incurs pronounced success rate losses in all three settings. These results complement the main evaluation: UAD remains effective across



(a) SmolVLA + LIBERO (b) $\pi_{0.5}$ + LIBERO (c) $\pi_{0.5}$ + LIBERO-Plus

Figure 11: Success rate-latency trade-offs across three additional simulation settings. Broken horizontal axes accommodate RTC's substantially lower AAL. Error bars indicate Wilson 95% CIs for SR, and episode bootstrap 95% CIs for AAL. Some AAL CIs are too narrow to be visible at the plotted scale.

the additional settings, while its advantage over uniform step reduction is more pronounced when task success depends more strongly on the denoising budget.