DASH: DIFFICULTY-AWARE AND MATURITY-ORIENTED STREAMING FOR REAL-TIME FLOW-BASED VLAS

Anonymous authors
Paper under double-blind review

ABSTRACT

Flow-based Vision-Language-Action (VLA) models generate continuous robot actions through iterative denoising, but this process creates a latency–quality trade-off for real-time control. Recent studies adopt asynchronous and streaming execution schemes to reduce Time to First Action (TTFA) by overlapping generation with execution or releasing early action tokens before the full action chunk is denoised. However, we find that early-released actions must remain reliable within each chunk, while denoising and observation frequency should be coordinated according to sample-level convergence difficulty of actions. To this end, we propose *DASH* (Difficulty-Aware Maturity-Oriented Streaming), which combines Maturity-Oriented Matching to improve causal consistency within each action chunk with Difficulty-Aware Denoising to jointly adapt denoising budgets and observation frequency based on convergence difficulty, thereby enhancing both action reliability and closed-loop responsiveness. Our simulation and real-world experiments demonstrate up to **3.44**$\times$ reaction speedup over synchronous inference and up to **3.26%** improvement in average CALVIN success over the strongest streaming baseline.

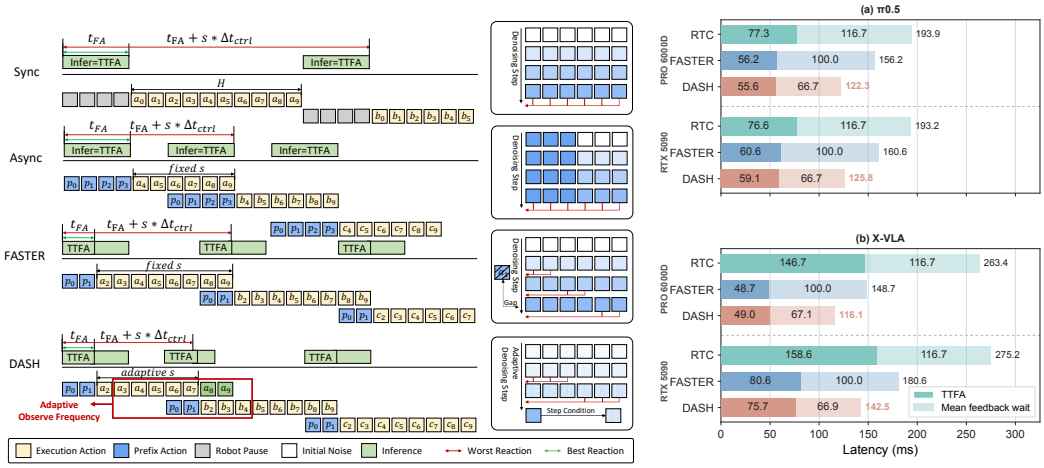
1 INTRODUCTION

Vision-Language-Action models (VLAs) have enabled robots to perform a wide range of manipulation tasks (Brohan et al., 2023b;a; Kim et al., 2024). Flow-based VLAs generate continuous action chunks through iterative denoising (Zhao et al., 2023; Chi et al., 2023; Lipman et al., 2023; Black et al., 2024). During *synchronous* execution, the robot waits for action generation, executes the actions, and acquires a new observation for the next inference cycle. This process introduces pauses between chunks and prevents the robot from incorporating new observations during execution (Black et al., 2025a;b). Consequently, it may continue following an outdated plan after the environment changes, limiting its ability to perform tasks that require fast reactions.

Two lines of work attempt to reduce this idling. Similar to overlapping communication and computation to hide latency in distributed systems (Zhang et al., 2017), *asynchronous* methods reduce pauses by generating the next action chunk while the robot executes the current one (Zhao et al., 2023; Black et al., 2025b). Streaming inference further reduces Time to First Action (TTFA) by releasing early actions before the entire chunk has been denoised (Chen et al., 2024; Lu et al., 2026). Analogous to optimizing Time to First Token (TTFT) in LLM serving Kwon et al. (2023), this allows the robot to *shorten its execution horizon* and acquire new observations more frequently, thereby improving its responsiveness to environmental changes. However, shorter execution horizons require more frequent transitions between new action chunks, which may introduce discontinuities at chunk boundaries. In real-world deployment, the robot should react quickly and reliably when new feedback is needed. Although existing asynchronous and streaming schemes reduce inference-induced delay, they still operate with a fixed execution horizon, leaving the feedback interval unchanged across states (see Fig. 1 for detailed comparison) (Tang et al., 2025). This raises a fundamental question: *How should VLAs coordinate inference, execution, and feedback to balance responsiveness with reliable control?*

Achieving both in streaming VLAs imposes two challenges: ❶ An adaptive schedule is definitely needed to decide when to observe, how much to compute, and how long to act before observing

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.



(a) Responsiveness timing in streaming flow-based VLAs. (b) Composition of reaction time.

Figure 1: Comparison of different VLA schemes. (a) Timing diagram of different execution schemes. Unlike Async and FASTER, which rely on a fixed execution horizon s, DASH adapts s according to state difficulty. (b) In addition to TTFA, DASH also reduces the second term of mean feedback wait, thereby optimizing the overall responsiveness.

again, like human checking progress more often during delicate manipulation. The robot should adjust observation times according to the demands of the current state. Our empirical analysis indicates that action predictions also converge at different rates across observations, thereby requiring joint adaptation of the denoising budget and execution horizon. ❷ Streaming execution induces a distribution mismatch between training and inference. Once early actions are released, they are frozen and used as context to generate the remaining actions. These predictions could have errors, whereas training supplies ground-truth prefixes. Consequently, the model must generate subsequent actions from imperfect context that is missing during training.

To address these challenges, we propose DASH, a difficulty-aware and maturity-oriented streaming framework. To address varying convergence difficulty across denoising states, the Difficulty-Aware Denoising (DAD) module predicts a per-DOF (degrees of freedom) convergence difficulty from the hidden states of the first denoising pass using a multi-head MLP. This successfully accounts for different observations and action dimensions converging at different rates, which jointly adapts the denoising budget and execution horizon to the predicted difficulty. Because adaptive scheduling frequently releases action tokens before the full chunk is denoised, these frozen prefixes inevitably contain prediction errors. To mitigate this side effect, the Maturity-Oriented Matching (MOM) module encodes the denoising progress of action tokens and finetunes the action expert on mixed-maturity chunks, so that suffix actions can condition reliably on early-released prefixes. As a metaphor, DAD and MOM work like a family: DAD acts as the strategist to dynamically adapt execution to difficulty; MOM acts as the safeguard to handle the imperfect prefixes caused by such early releases. Across two VLA backbones and two GPUs evaluation, DASH preserves low TTFA and enhances real-time reactivity at the same task accuracy. It achieves up to 3.44× reaction speedup over synchronous inference while improving task performance on LIBERO (Liu et al., 2023) and CALVIN (Mees et al., 2022).

2 Related Work

Vision-Language-Action Models. Vision-Language-Action (VLA) models extend large-scale vision-language pretraining to embodied action learning (Brohan et al., 2023b;a). The existing VLA architectures primarily fall into two directions. The first one directly adapts the autoregressive LLM backbone to predict discretized actions, exemplified by OpenVLA (Kim et al., 2024) and a line of optimization works Zheng et al. (2025); Qu et al. (2025). While effective, these methods often struggle to generate smooth, continuous trajectories due to the discrete nature of action tokenization. The second direction employs flow matching for action generation, represented by the $\pi_{0.5}$ (Black et al., 2024) and GR00T (NVIDIA et al., 2025). These approaches incorporate flow-matching action expert to the VLM backbone, iteratively denoising Gaussian noise into high-quality continuous

action chunks conditioned on vision-language features (Lipman et al., 2023; Chi et al., 2023). By producing smoother and more stable trajectories, flow-based methods are dominating VLA research for precise control. To this end, this work mainly focuses on flow-based VLAs.

Real-Time VLAs. Existing methods improve real-time VLA execution by either reducing inference cost or overlapping inference with robot execution. Early works primarily reduce the inference cost by using smaller VLM backbones (Wen et al., 2025; Liu et al., 2024), compressing LLM layers (Reuss et al., 2025; Yue et al., 2024; Zhang et al., 2026), pruning visual tokens (Xu et al., 2025; Yang et al., 2025). While these methods have shortened inference latency, they treat VLA as a software pipeline and ignores the dynamics of physical robot execution. On the other hand, execution-aware scheduling methods adjust the temporal relationship between action generation and execution. Asynchronous inference overlaps action generation with execution (Black et al., 2025a;b; Tang et al., 2025). FASTER releases near-term actions before the full chunk is denoised and StreamingVLA adopts saliency-aware early observation to overlap generation, execution, and observation (Lu et al., 2026; Shi et al., 2026). However, they do not jointly consider computation and feedback scheduling with the convergence difficulty, while those early-released actions provide imperfect context for subsequent generation. In contrast, this work delves into the trade-off between real-time responsiveness and action quality by ensuring the consistency of early released actions.

3 Analyze Denoising Dynamics in Streaming VLAs

To understand what limits real-time streaming control, we first conduct an empirical analysis to answer three central questions: (1) How does the execution horizon affect responsiveness and motion continuity? (2) Is a fixed denoising budget suitable for each robot state? (3) Are prefix action tokens sufficiently accurate when released early?

Observation #1 (Shorter Horizons Increase Motion Fluctuations). Shorter execution horizons lead to more frequent and larger command-velocity fluctuations, whereas longer horizons produce smoother motion.

Given a predicted action chunk $A_t = [a_t, a_{t+1}, \ldots, a_{t+H-1}]$ with horizon H, we define the execution horizon s as the number of actions executed before acquiring the next observation ($s \leq H$). To investigate this, we deploy a finetuned $\pi_{0.5}$ model on a real robot with control frequency $f_{ctrl} = 30$ Hz and only vary the execution horizon s. We trace six arm-joint velocities $v_t = f_{ctrl} \cdot \sum_{j=1}^{6} |a_{j,t+1} - a_{j,t}| /6$. As shown in Fig. 2, smaller execution horizons lead to more velocity fluctuations and higher peak magnitudes, and vice versa. This implies a potential trade-off between execution horizon (responsiveness) and stability (precision).

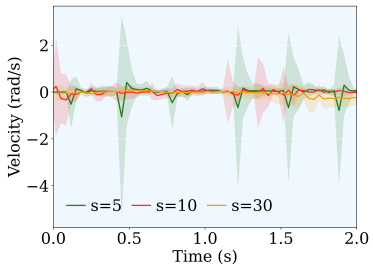


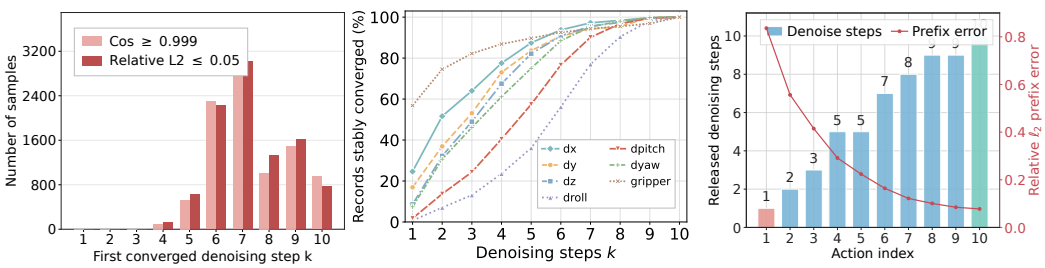
Figure 2: Different execution horizons.

Observation #2 (Denoising Difficulty Varies Across Samples and Action DoFs). Different samples and action DoFs require different denoising steps to converge.

As shown in Fig. 3a, some samples converge after a few steps, whereas others require more computation. This suggests that the robot can allocate computation according to the current observation rather than use a fixed budget. The same is observed in action DoFs as well (shown in Fig. 3b): at the 4-th denoising step, over 67% of the translational and 87% of gripper actions have converged, compared with less than 61% of rotational actions. These observations suggest convergence difficulty as a proxy to guide denoising computation, which reflects how much refinement the current prediction needs.

Observation #3 (Early-Released Actions Are Error-Prone). Actions released with fewer denoising steps can deviate substantially from fully denoised reference, which offsets the speedup gains with less reliability.

To examine the reliability of prefix actions thoroughly, we collect 10K action chunks from LIBERO (Liu et al., 2023) and CALVIN (Mees et al., 2022). For each observation, we fix the policy conditioning and initial noise, and compare each action at its release step with the corresponding action in the 10-step reference $\widehat{A}_{n,\text{ref}} = A_n^{(10)}$. As shown in Fig. 3c, early-released actions can have substantial errors. Once released, these actions are frozen and used as context for the remaining actions to condition on. Unfortunately, prior streaming methods ignore maturity of the released



(a) Sample-level convergence. (b) DoF-level convergence. (c) Token-level reliability gap.

Figure 3: **Denoising reliability and sample difficulty in streaming flow-based VLAs.** (a) The histogram of minimum denoising steps K^* required to reach convergence (relative $\ell_2 < 0.05$) across 10K validation observations, showing substantial variation in sample difficulty. (b) Percentage of records stably converged at each denoising step, shown separately for the translational, rotational, and gripper action DoFs with different convergence behavior. (c) The relative ℓ_2 error between the action released at step k (right y axis) and the fully-denoised reference. Releasing at the tenth step reduces the error from 0.836 to 0.078.

prefix Lu et al. (2026); Shi et al. (2026), such that the model cannot distinguish an early-released prefix that is error-prone, from a fully denoised one. By the same analogy of *positional embedding* in Transformers Vaswani et al. (2017), we should provide the model with maturity information.

4 Design of DASH

Our real-robot analysis shows that the execution horizon s affects both control accuracy and responsiveness. Inspired by how humans adjust reasoning effort and observation frequency to task difficulty (Petitet et al., 2021), we propose DASH to adapt both denoising computation and execution to the current state. As shown in Fig. 5, it is comprised of two modules: **Difficulty-Aware Denoising (DAD)** utilizes convergence difficulty as a proxy to select the denoising steps and execution horizon; **Maturity-Oriented Matching (MOM)** mitigates the distributional mismatch between training and inference in prefix conditioning. Before we present the design details, we first model the response of adaptive scheduling.

4.1 Responsiveness Modeling for Adaptive Scheduling

Given an observation o and robot state q, the policy predicts an action chunk $A = [a_0, a_1, \ldots, a_{H-1}]$ with prediction horizon H. Denote $s \leq H$ as the execution horizon: after every s control steps, the controller acquires a new observation and issues an inference request (Black et al., 2025b; Tang et al., 2025). The robot runs at control frequency f_{ctrl}, with interval $\Delta t_{\text{ctrl}} = 1/f_{\text{ctrl}}$. $t_{\text{FA}}(N)$ is the TTFA with N denoising steps, and N is fixed in (Lu et al., 2026). Streaming inference releases actions before the full chunk is generated. For continuous execution, the first action of the next chunk must arrive before the current execution interval ends. The minimum execution horizon is thus the inference delay in control steps:

$$s_{\min}(N) = \left\lceil \frac{t_{\text{FA}}(N)}{\Delta t_{\text{ctrl}}} \right\rceil. \tag{1}$$

Eq. 1 is the requirement for real-time manipulation. When inference runs, the robot continues executing the current chunk for at least $s_{\min}$ control steps with $s \geq s_{\min}(N)$. Assume an environmental change occurs uniformly within an execution interval of $s\Delta t_{\text{ctrl}}$. The robot must wait for the next observation as well as the first action conditioned on that observation. Based on (Lu et al., 2026), the expected reaction latency for a fixed execution horizon is,

$$\mathbb{E}[\Delta t_{\text{react}}] = \mathbb{E}[t_{\text{FA}}] + \frac{1}{2} s \Delta t_{\text{ctrl}}. \tag{2}$$

The two terms represent TTFA of the next request and the mean waiting time until the next observation, respectively. In contrast to fixed horizon, DASH adopts a state-dependent execution horizon $s(o, q)$ with N denoising steps. The excepted reaction latency for adaptive execution horizon is,

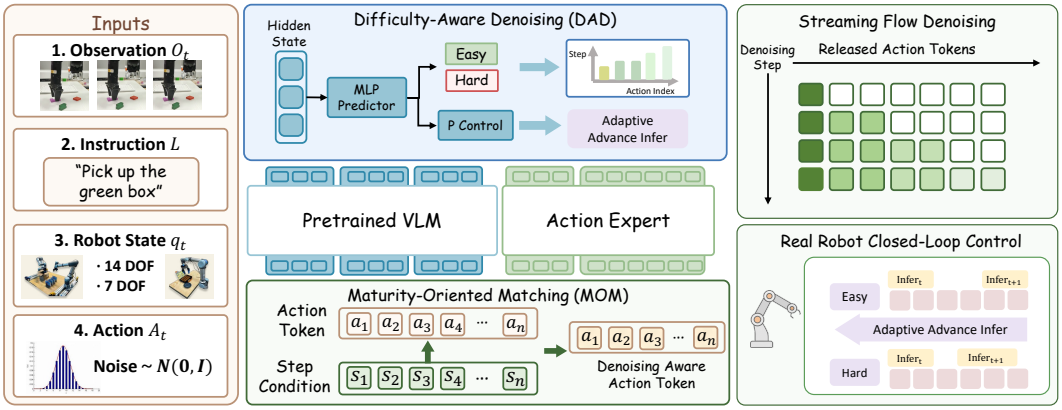


Figure 5: **Overview of the DASH framework.** Difficulty-Aware Denoising (DAD) predicts the convergence difficulty from intermediate hidden states and jointly adapts the denoising budget and execution horizon, and Maturity-Oriented Matching (MOM) conditions action tokens on their denoising maturity to improve generation under mixed-maturity prefixes.

$$\mathbb{E}[\Delta t_{\text{react}}]_{\text{adapt}} = \frac{\mathbb{E}[s(o,q)\, t_{\text{FA}}(N)]}{\mathbb{E}[s(o,q)]} + \frac{1}{2}\frac{\mathbb{E}[s(o,q)^2]}{\mathbb{E}[s(o,q)]}\Delta t_{\text{ctrl}}. \quad (3)$$

We refer to Appendix A.2 for detailed derivation. Although it is difficult to make a direct comparison between Eqs. 2 and 3, we recall from Observation 1 which implies a trade-off between responsiveness and precision. That is, naively reducing s or N would degrade action quality. However, using fixed values tend to lock them at sub-optimal regions as illustrated by Fig. 4. In contrast, by adaptively adjusting s and N, we could better explore the Pareto-frontier along the speed-accuracy curve.

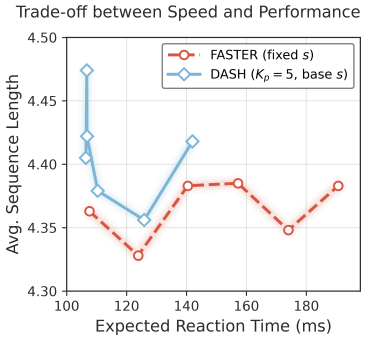


Figure 4: Speed and Performance Trade-off.

4.2 Difficulty-Aware Denoising (DAD)

Recall that Observation 2 implies convergence difficulty as an effective proxy for denoising budget and execution horizon. In this subsection, we first design the DAD module to assign more denoising steps and shortens the execution horizon; for fast converging predictions, DAD uses fewer denoising steps and retains a longer execution horizon, both under $s_{\min}(N) \le s \le s_{\max}$.

Convergence Difficulty Prediction. We reuse the last-layer action-expert hidden states from the first denoising step and apply mean pooling to obtain a condensed tensor h. Since Observation 2 shows that translational, rotational, and gripper DoFs converge at different rates, we leverage a multi-head MLP to predict a difficulty score for each group, and take the maximum as the chunk-level convergence difficulty r,

$$r = \max_{g \in \{\text{trans,rot,grip}\}} \text{sigmoid}(f_g(h))\,. \quad (4)$$

f_g is the MLP head for the DoF group g. Taking the maximum makes the predicted difficulty depend on the slowest-converging group. Accordingly, DASH models the convergence difficulty of the translational, rotational, and gripper action groups independently, thereby capturing their distinct convergence characteristics and using the most difficult group as the bottleneck to determine a shared denoising budget for the entire action chunk.

Adaptive Inference. During inference, we compute r from the first action-expert forward pass and map it monotonically to the number of denoising steps N. Difficult states receive more denoising steps, while easy states receive fewer. We then adjust the execution horizon with a feedback controller (Åström & Hägglund, 2001). Starting from the fixed horizon s_{fixed}, we use r as feedback and subtract a correction $K_p r$, where K_p is the proportional gain. Applying the lower bound in Eq. 1 gives

$$s = \max\{s_{\min}(N), \text{round}(s_{\text{fixed}} - r \cdot K_p)\}\,. \quad (5)$$

The gain K_p controls how much the execution horizon decreases with difficulty. We round the adjusted horizon to an integer and enforce $s_{\min}(N)$ as its lower bound. As validated in Fig. 6, this difficulty predictor accurately tracks the oracle denoising budgets and successfully reduces average computational steps while preserving task success. However, while DAD resolves the dynamic allocation of computation and feedback, it exacerbates the reliance on early-released action tokens. This raises a critical training-inference mismatch that MOM should address next.

4.3 Maturity-Oriented Matching (MOM)

Under the adaptive schedule, the policy frequently releases early actions before the full action chunk is completely denoised. Once released, these prefix actions are frozen and serve as the conditioning context for the remaining suffix actions. However, during standard training, the policy is only exposed to ground-truth action prefixes. This creates a distribution mismatch analogous to exposure bias under teacher forcing (Ranzato et al., 2016). That is, at inference time, the model must generate subsequent actions conditioned on its own imperfect, frozen predictions, but it has never been trained to do so. To bridge this gap, inspired by conditioning augmentation (Ho et al., 2022; Saharia et al., 2022), we propose MOM to explicitly *simulate* these imperfect prefix conditions during training by injecting position-dependent Gaussian noise. Since the model cannot distinguish a fully denoised prefix from an early-released one based on action values alone, we equip the action expert with *maturity embeddings*, encoding each token's planned denoising steps and remaining steps before release.

Mixed-Maturity Finetuning. Following Training-time RTC (Black et al., 2025b), we sample a clean prefix of length d_{RTC}, retain its ground-truth actions, and set $\tau_i = 0$ for $i < d_{\mathrm{RTC}}$, so that the policy learns to generate subsequent actions conditioned on a clean, fixed prefix. However, unlike Training-time RTC, we additionally introduce a noisy prefix of length d_{noise} to model the imperfect frozen predictions encountered during streaming inference. Specifically, with total prefix length $d = d_{\mathrm{RTC}} + d_{\mathrm{noise}} < H$, we perturb the ground-truth actions at positions $d_{\mathrm{RTC}} \le i < d$ with position-dependent Gaussian noise $\delta_i \sim \mathcal{N}(0, \sigma_i^2 I)$, while keeping $\tau_i = 0$ for these positions as well. *This teaches the policy to rely on noisy, frozen contexts rather than only clean prefixes.* For $i \ge d$, local timesteps are constructed from a sampled global timestep using the horizon-aware schedule. The resulting action input is,

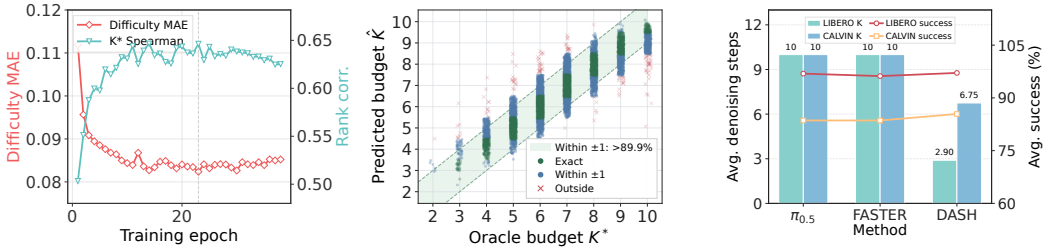
$$\widetilde{A}_i = \begin{cases} A_i, & i < d_{\mathrm{RTC}}, \\ A_i + \delta_i, & d_{\mathrm{RTC}} \le i < d, \\ (1-\tau_i)A_i + \tau_i \epsilon_i, & i \ge d. \end{cases} \tag{6}$$

In Eq. 6, A_i is the ground-truth action and $\epsilon_i \sim \mathcal{N}(0, I)$ is Gaussian noise. The first two cases represent the frozen prefix (clean and perturbed, respectively), while the third case represents the postfix that follows the standard flow matching path (Lipman et al., 2023). Within the noisy prefix, we decrease σ_i with the action index to reflect the prefix-error trend in Observation 2. The local flow matching timestep τ_i specifies a token's noise level, but does not specify its discrete denoising budget. We therefore encode each token's planned and remaining denoising steps. At inference, n_i denotes the planned denoising steps for token i, determined by the global budget N and the horizon-aware schedule (Algorithm 1). For denoising steps $j > 1$, we construct the condition B_i^j and encode it using DenoiseMLP, a lightweight MLP that projects the token-wise denoising-step condition into the action embedding space,

$$B_i^j = \begin{cases} [1, 0], & i < d, \\ \left[\dfrac{n_i}{N_{\max}}, \dfrac{\max(n_i - j + 1, 0)}{N_{\max}} \right], & i \ge d, \end{cases} \qquad j > 1. \tag{7}$$

$$e_i^j = E(A_i) + \text{DenoiseMLP}(B_i^j). \tag{8}$$

The first denoising forward pass uses $B^1 = 0$, and its velocity is reused for the first Euler update after the global budget has been selected. A_i denotes the current action input, $E(\cdot)$ is the action embedding layer, and DenoiseMLP encodes the maturity condition. $N_{\max}$ is the maximum denoising budget used for normalization. The d initial prefix tokens receive $[1, 0]$. For each remaining token, B_i^j encodes its planned and remaining denoising steps, both normalized by $N_{\max}$. Intuitively, this



(a) Validation dynamics. (b) Predicted versus oracle budget. (c) Average steps and success.

Figure 6: **Difficulty Predictor Analysis.** (a) Training dynamics showing the predictor effectively learns sample difficulty. The budget MAE (mean absolute error between predicted and oracle-optimal denoising steps) steadily decreases, while the rank correlation with oracle budgets increases. (b) Scatter plot of predicted versus oracle (optimal) denoising budgets. Over 89.9% of predictions fall within the shaded ± 1 step error band, demonstrating high prediction reliability. (c) DASH lowers average denoising steps across LIBERO and CALVIN while preserving task success.

condition tells the action expert exactly how "mature" a given prefix token is. If a token is released early, the model learns to be more conservative compared to a fully denoised token. Once a token finishes, its remaining-step component becomes zero, while its planned-step component stays unchanged.

Postfix-Only Supervision. The frozen prefix serves only as conditioning context and is excluded from the training objective. Given observation o, the action expert takes the mixed-maturity input from Eq. 6 and local timesteps $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{H-1})$. We average the flow-matching loss over the $H - d$ postfix positions with $i \geq d$, including those with $\tau_i = 0$ under the horizon-aware schedule,

$$\mathcal{L}_{\text{MOM}} = \frac{1}{H-d} \sum_{i=d}^{H-1} \left\| v_{\theta,i}\left(o, \widetilde{\mathbf{A}}, \boldsymbol{\tau}\right) - (\epsilon_i - A_i) \right\|_2^2 . \tag{9}$$

where $v_{\theta,i}$ denotes the flow-matching velocity vector predicted at action position i by the action expert with parameters θ. This training procedure teaches the action expert to denoise the postfix conditioned on an imperfect frozen prefix with different token maturities. In sum, DAD determines the denoising computation and observation-update timing, while MOM supports reliable action generation under the resulting mixed-maturity states.

5 Experiments

Baselines. We compare DASH with five representative execution strategies: the vanilla Synchronous inference (Sync), Temporal Ensembling (TE) (Zhao et al., 2023), Training-time RTC (Black et al., 2025a;b) and StreamingVLA (Shi et al., 2026). All methods use matched evaluation settings with further details in Appendix A.4.

5.1 Simulated Benchmark

Experimental Setup. We evaluate all methods on the LIBERO (Liu et al., 2023) and CALVIN (Mees et al., 2022) simulation benchmarks using the $\pi_{0.5}$ (Intelligence et al., 2025) and X-VLA (Zheng et al., 2026) backbones under matched evaluation settings. When reporting real-time performance, we use $s^\star$ as the execution horizon that achieves the highest task accuracy on the CALVIN benchmark. All reported success rates are averaged over five independent random seeds. Table 1 reports the latency and responsiveness of each method at these accuracy-optimal operating points, while $s_{\min}$ denotes the hardware-dependent feasibility lower bound determined by inference latency. This allows us to analyze the trade-off between task accuracy and responsiveness.

Results. As shown in Table 1, DASH achieves the best overall responsiveness across both backbones. For $\pi_{0.5}$, it reduces the expected reaction time to 122.27 ms, yielding a 2.29× speedup over Sync. On X-VLA, although FASTER achieves a slightly lower TTFA (48.72 ms vs. 49.01 ms), DASH achieves a substantially lower reaction time of 116.09 ms and a 3.44× speedup over Sync, indicating that minimizing TTFA alone is insufficient for fast closed-loop response. As analyzed in Sec. 4, responsiveness also depends on the execution horizon s: shorter horizons refresh observations sooner but increase chunk-transition frequency, whereas longer horizons preserve motion

Table 1: Comparison of real-time inference efficiency on PRO 6000D and RTX 5090.

Model	Method	PRO 6000D					RTX 5090				
		TTFA (ms) ↓	$s_{\min}$ ↓	$\mathbb{E}[\Delta t_{\text{infer}}]$ ↓	$\mathbb{E}[\Delta t_{\text{react}}]$ ↓	Speedup ↑	TTFA (ms) ↓	$s_{\min}$ ↓	$\mathbb{E}[\Delta t_{\text{infer}}]$ ↓	$\mathbb{E}[\Delta t_{\text{react}}]$ ↓	Speedup ↑
$\pi_{0.5}$	Sync	$75.49_{\pm 0.77}$	3	**75.49**	279.90	1.00×	$74.40_{\pm 1.55}$	3	74.40	278.27	1.00×
	TE	$76.48_{\pm 0.85}$	3	76.48	193.15	1.45×	$73.25_{\pm 1.78}$	3	**73.25**	189.91	1.47×
	T-RTC	$77.28_{\pm 0.81}$	3	77.28	193.94	1.44×	$76.56_{\pm 0.87}$	3	76.56	193.22	1.44×
	StreamingVLA	$65.49_{\pm 0.92}$	2	120.02	198.83	1.41×	$87.08_{\pm 1.84}$	3	229.36	220.41	1.26×
	FASTER	$56.20_{\pm 0.76}$	2	88.38	156.20	1.79×	$60.57_{\pm 1.66}$	2	112.79	160.57	1.73×
	DASH	**55.60**$_{\pm 0.67}$	**2**	78.44	**122.27**	**2.29**×	**59.12**$_{\pm 1.64}$	**2**	95.00	**125.79**	**2.21**×
X-VLA	Sync	$154.85_{\pm 3.06}$	5	154.85	398.94	1.00×	$155.47_{\pm 3.19}$	5	155.47	399.87	1.00×
	TE	$146.19_{\pm 2.56}$	5	146.19	229.52	1.74×	$150.40_{\pm 2.13}$	5	150.40	233.74	1.71×
	T-RTC	$146.74_{\pm 2.67}$	5	146.74	263.41	1.51×	$158.58_{\pm 3.79}$	5	158.58	275.25	1.45×
	FASTER	**48.72**$_{\pm 3.57}$	2	124.21	148.72	2.68×	$80.61_{\pm 4.79}$	3	180.20	180.61	2.21×
	DASH	$49.01_{\pm 2.83}$	**2**	**85.89**	**116.09**	**3.44**×	**75.65**$_{\pm 4.65}$	**3**	**117.84**	**142.52**	**2.81**×

Table 2: Task success rates on LIBERO and CALVIN ABC→D.

Model	Method	LIBERO					CALVIN ABC→D					
		Spatial	Object	Goal	10	Avg.	1	2	3	4	5	Avg. Len
$\pi_{0.5}$	Sync	98.8	98.2	98.0	**92.4**	96.9	96.1	89.2	82.1	77.0	69.0	4.134
	TE	91.2	88.9	93.2	78.8	88.0	93.3	81.2	67.2	56.3	48.2	3.462
	T-RTC	90.8	91.2	94.5	88.6	91.3	94.4	87.2	76.7	69.0	61.5	3.888
	StreamingVLA	96.6	96.6	95.4	91.0	94.9	88.2	80.8	64.2	50.4	45.6	3.292
	FASTER	98.2	**98.4**	97.2	90.8	96.2	98.2	93.6	88.1	82.4	76.0	4.383
	DASH	**99.4**	98.2	**98.6**	92.2	**97.1**	**99.1**	**94.3**	**88.5**	**84.8**	**80.7**	**4.474**
X-VLA	Sync	97.0	98.8	98.0	**97.4**	97.8	**95.8**	89.4	82.9	77.4	69.6	4.151
	TE	87.4	85.8	92.2	87.6	88.3	87.8	76.6	64.6	53.9	42.0	3.249
	T-RTC	91.6	90.4	91.8	87.8	90.4	92.0	83.0	72.9	64.5	54.3	3.667
	FASTER	**99.0**	99.2	97.6	92.0	97.0	97.7	**91.1**	81.2	72.1	63.7	4.058
	DASH	97.4	**99.2**	**98.4**	96.0	**97.8**	95.4	90.2	**85.0**	**79.4**	**72.1**	**4.221**

continuity at the cost of stale feedback. DASH addresses this trade-off by shortening s for difficult states and retaining longer execution for easier states, thereby jointly improving feedback responsiveness and execution stability.

5.2 Real-world Experiments

Experimental Setup. We design a dynamic pick-and-place task that requires both rapid state response and stable motion execution. Specifically, the robot must grasp a target object and place it into a designated receptacle. During execution, we randomly relocate the target object, the receptacle, requiring the robot to recognize the scene change, and correct its trajectory using the latest observations. We evaluate three conditions: *no perturbation, target relocation and receptacle relocation*. We compare Sync, Training-time RTC, FASTER, and DASH using the same control frequency, action horizon, and camera configuration. All methods are evaluated under the same maximum number of inference calls, and we report the resulting task success rate. A trial is counted as a failure if the robot does not complete the task within this inference budget.

Results. We evaluate DASH on the AgileX 7-DoF platform over 150 real-robot trials, with 50 trials each under no perturbation, target relocation, and receptacle relocation. DASH achieves success rates of 92%, 82%, and 84%, ranking the highest in two conditions (Fig. 7). On average, DASH outperforms FASTER by 8.7%, which is the strongest baseline under perturbation. We further observe that fixing $s = s_{\min}$ allows more frequent responses to new observations but introduces discontinuities between action chunks, reducing grasping or placement precision and occasionally causing failures. Increasing s produces smoother trajectories but delays redirection after target relocation. These observations reveal a trade-off between motion continuity and closed-loop responsiveness: shorter execution horizons increase chunk transition frequency, whereas longer horizons prolong reliance on stale observations. DASH addresses this trade-off by adapting the execution horizon to new observations. Thus, real-world closed-loop performance depends not only on TTFA, but also on observation-to-action latency and consistency between successive action chunks.

5.3 Ablation Studies

Component Analysis. We ablate DASH to isolate the contributions between the Maturity-Oriented Matching and Difficulty-Aware Denoising modules. Table 3 compares the full method with the

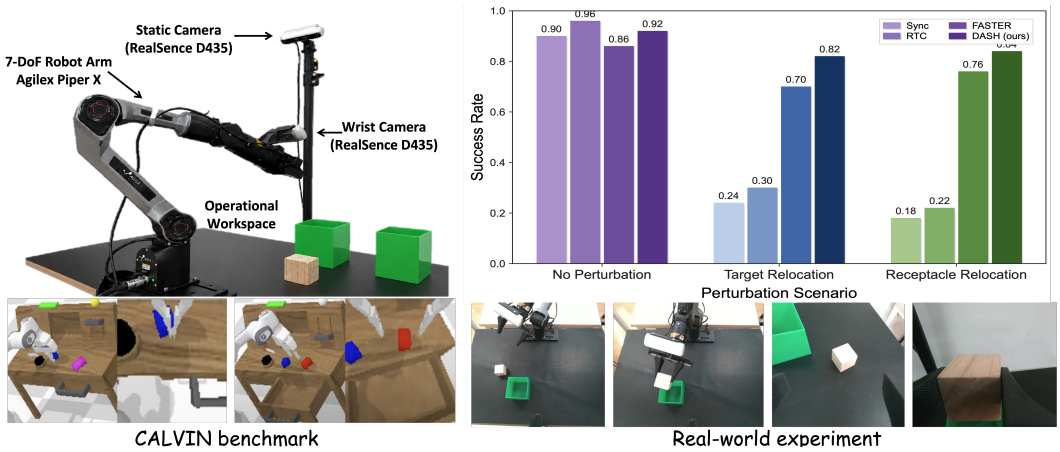


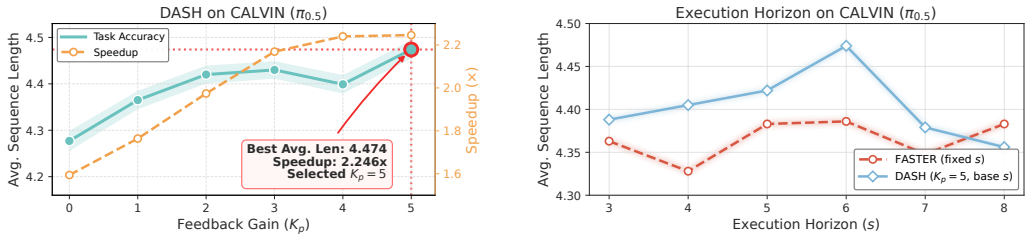
Figure 7: **Real-world evaluation setup and results.** Left: the dual-camera robot setup. Right: success rates of Sync, training-time RTC, FASTER, and DASH under no perturbation, target relocation, and receptacle relocation, with representative real-world rollouts shown below.

fixed-schedule streaming baseline and variants without MOM, DAD, or adaptive observation refresh; we also disable adaptive denoised steps N and execution horizon s to distinguish computation from feedback allocation.

Table 3: **Component Ablation of DASH.**

Variant	MOM	DAD		Real-Time Responsiveness			CALVIN Accuracy
		N	s	TTFA (ms) ↓	$\mathbb{E}[\Delta t_{\text{react}}]$ (ms) ↓	Speedup ↑	Avg. Len ↑
Streaming baseline	–	–	–	60.77	177.44	1.57×	4.383
DASH w/o MOM	–	✓	✓	59.12	125.79	2.21×	4.336
DASH w/o DAD	✓	–	–	60.77	177.44	1.57×	4.355
DASH w/o adaptive refresh	✓	✓	–	58.21	158.21	1.76×	4.376
DASH	✓	✓	✓	59.12	125.79	2.21×	4.474

Hyperparameter Sensitivity. Finally, we study the two hyperparameters governing adaptive closed-loop observation. We first vary the feedback gain K_p, which controls how strongly the predicted difficulty shortens the execution horizon. We then vary the base execution horizon and compare DASH with the corresponding fixed-horizon FASTER configurations over $s \in \{3, \ldots, 8\}$. Together, these results suggest that DASH exhibits low sensitivity to both hyperparameters, maintaining relatively stable task performance across the tested ranges.



(a) Accuracy-responsiveness trade-off under K_p. (b) Effect of the base execution horizon.

Figure 8: **Hyperparameter sensitivity on CALVIN.** (a) task accuracy and responsiveness speedup of DASH under different feedback gains. (b) CALVIN accuracy across execution-horizon operating points.

6 Conclusion

In this paper, we propose a difficulty-aware and maturity-oriented streaming framework for flow-based VLA models. By improving the reliability of early-released actions and jointly adapting denoising budgets and execution horizons, we enable more responsive and reliable closed-loop control. Extensive experiments demonstrate substantial speed-up compared to SOTA streaming baselines.

7 Ethics Statement

This work adheres to the ICLR Code of Ethics. In this study, no human subjects or animal experimentation was involved. All datasets used, including the LIBERO and CALVIN simulation benchmarks, are publicly released robot-learning benchmarks that were obtained and used in compliance with their respective usage guidelines, and no personally identifiable information is involved. Our real-world experiments involve only a robotic manipulator performing pick-and-place of household objects on a test bench, with no interaction with human participants, and raise no privacy or security concerns. We have taken care to avoid biases or discriminatory outcomes in our research process, and we are committed to transparency and integrity throughout the research process.

8 Reproducibility Statement

We have made every effort to ensure that the results presented in this paper are reproducible. The experimental setup, including finetuning steps, model configurations, and hardware details (backbones, GPUs, and the real-robot deployment), is described in detail in the appendix. We also provide a full description of the difficulty-predictor training procedure and the construction of oracle-budget labels, to assist others in reproducing our experiments. Additionally, the public robot-manipulation benchmarks used in the paper, such as LIBERO and CALVIN (ABC→D), are openly released with standardized evaluation protocols, ensuring consistent and reproducible evaluation results.

9 LLM Usage

Large language models (LLMs) were used to aid the writing and polishing of this manuscript, including sentence rephrasing, grammar checking, and improving the overall flow of the text. The LLM was not involved in ideation, research methodology, experimental design, or data analysis; all research concepts, algorithms, experiments, and analyses were developed and conducted by the authors. The authors take full responsibility for the content of the manuscript, including any text polished with the assistance of LLMs, and have reviewed all AI-assisted text to ensure that it introduces no plagiarism or scientific misconduct.

References

Karl Johan Åström and Tore Hägglund. The future of pid control. *Control engineering practice*, 2001.

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 Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *NeurIPS*, 2025a.

Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025b.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023a.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems*, 2023b.

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

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

Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *JMLR*, 2022.

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.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, 2023.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *ICLR*, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS*, 2023.

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, Xiaoqi Li, Kaichen Zhou, Senqiao Yang, Renrui Zhang, Yandong Guo, and Shanghang Zhang. RoboMamba: Efficient vision-language-action model for robotic reasoning and manipulation. In *NeurIPS*, 2024.

Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. FASTER: Rethinking real-time flow VLAs. *arXiv preprint arXiv:2603.19199*, 2026.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 2022.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Pierre Petitet, Bahaaeddin Attaallah, Sanjay G Manohar, and Masud Husain. The computational cost of active information sampling before decision-making under uncertainty. *Nature human behaviour*, 2021.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *ICLR*, 2016.

Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-flow models. In *Proceedings of The 9th Conference on Robot Learning*, 2025.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, 2022.

Yiran Shi, Dongqi Guo, Tianchen Zhao, Feng Gao, Liangzhi Shi, Chao Yu, ZhiJian Mo, Qihua Xiao, XiaoShuai Peng, Qingmin Liao, et al. Streamingvla: Streaming vision-language-action model with action flow matching and adaptive early observation. *arXiv preprint arXiv:2603.28565*, 2026.

Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. TinyVLA: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient vision-language-action manipulation via adaptive token caching. In *NeurIPS*, 2025.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. EfficientVLA: Training-free acceleration and compression for vision-language-action models. In *NeurIPS*, 2025.

Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. DeeR-VLA: Dynamic inference of multimodal large language models for efficient robot execution. In *NeurIPS*, 2024.

Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017.

Rongyu Zhang, Menghang Dong, Yuan Zhang, Liang Heng, Xiaowei Chi, Gaole Dai, Li Du, Dan Wang, Yuan Du, and Shanghang Zhang. MoLe-VLA: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. In *AAAI*, 2026.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Tai Wang, Ya-Qin Zhang, Jingjing Liu, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *ICLR*, 2026.

Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. In *ICLR*, 2025.

A APPENDIX

We provide the following supplementary materials:

- Appendix A.1 presents detailed empirical analyses of the challenges motivating DASH, including the trade-off between execution horizon and closed-loop control, the reliability of early-released action tokens, and sample- and degree-of-freedom-level heterogeneity in denoising convergence.
- Appendix A.2 provides the complete theoretical formulation of the streaming control process, including the derivation of the expected reaction time under fixed and adaptive execution horizons and the analysis of the physically feasible execution interval.
- Appendix A.3 presents the complete DASH algorithmic pipeline and pseudocode.
- Appendix A.4 summarizes all experimental protocols, including benchmark and dataset settings, backbone initialization, baseline implementation, policy fine-tuning, difficulty-predictor training, simulation evaluation, and latency measurement.
- Appendix A.5 describes the real-world robot platform, deployment and perturbation protocols, evaluation results.

A.1 DETAILED ANALYSIS OF THE CHALLENGES

This subsection provides additional details for the empirical observations in Sec. 3. These studies isolate three difficulties that arise when a flow-based VLA is used with streaming execution: the execution horizon affects both feedback frequency and cross-chunk continuity; an action released before the end of denoising may still contain substantial error; and the denoising effort required for convergence varies across observations and action degrees of freedom (DoFs). Together, these observations motivate the adaptive scheduling and maturity-aware training used in DASH.

A.1.1 EXECUTION HORIZON: FEEDBACK FREQUENCY VERSUS MOTION CONTINUITY

Execution Horizon. For an action chunk $A_t = [a_t, a_{t+1}, \ldots, a_{t+H-1}]$, the robot executes only the first s actions before acquiring a new observation and starting the next inference cycle. A smaller s shortens the time during which the robot acts on the same observation. It therefore increases the closed-loop observation frequency and allows the policy to respond sooner when the scene changes. However, it also increases the frequency of transitions between independently generated action chunks. A larger s has the opposite effect: it reduces the number of chunk transitions and preserves a longer continuous motion segment, but delays the next opportunity to observe the environment.

Controlled Real Robot Study. We isolate this trade-off by deploying the same fine-tuned $\pi_{0.5}$ policy on the real robot while changing only the execution horizon s. The policy checkpoint, camera observations, control frequency, action representation, and inference configuration are kept fixed. At every transition between two successive chunks, we measure the magnitude of the executed action change,

$$D_{\text{switch}} = \|a_t - a_{t-1}\|_2 \,, \tag{10}$$

where a_{t-1} is the last action executed from the previous chunk and a_t is the first action executed from the newly received chunk. We average this quantity over all chunk transitions. This metric does not measure task success by itself; it measures how abruptly the commanded trajectory changes when the controller replaces one chunk with another.

Observed Behavior. Short execution horizons lead to more frequent chunk replacement and larger average changes at chunk boundaries. On the physical robot, these changes can appear as jitter or repeated corrections, especially during smooth reaching and manipulation phases. Increasing s makes chunk transitions less frequent and improves motion continuity, but the robot then follows actions conditioned on an older observation for a longer time. Consequently, neither the shortest nor the longest feasible horizon is uniformly preferable. The appropriate horizon depends on the current control state: rapid replanning is more useful when the prediction is difficult or the scene may change, whereas a longer continuous segment is suitable when the motion is stable.

Connection to DASH. This observation motivates adapting the observation-update interval rather than using one fixed s throughout an episode. DASH shortens the execution horizon for difficult states to obtain a fresh observation sooner and retains a longer horizon for easier states to avoid unnecessary chunk switching. The feasibility bound derived in Appendix A.2 prevents this adaptation from requesting a new chunk before its first action can be delivered.

A.1.2 Residual Error in Early-Released Prefix Actions

Notation. Let H be the number of actions in a chunk and N the number of denoising steps. We index actions by $i = 0, \ldots, H-1$ and denoising states by $k = 0, \ldots, N$. The vector $x_i^{(k)}$ is action i after k updates, and $X^{(k)}$ collects all action vectors. The conditioning c contains the observation, instruction, and robot state. The first d positions, if any, are fixed actions supplied from the previous chunk; the newly generated actions occupy positions $i \geq d$. Local denoising time is denoted by $\tau_i^{(k)}$: time one is the noise endpoint and time zero is the action endpoint. This time is a flow-matching coordinate, not physical time or a measured prediction error.

Position-Dependent Completion in FASTER. We examine FASTER as a representative streaming approach. Its Horizon-Aware Schedule (HAS) assigns each action position a local denoising timetable, so near-term actions reach the terminal time earlier than distant actions. With the immediate action scheduled to finish after one integration step, the controller can begin execution while the remaining actions are still being generated. This reduces TTFA, but the release criterion is determined by the prescribed timetable rather than a measurement of prediction convergence for the current observation. Specifically, the HAS implementation described here uses

$$u_i = u_0 \left[1 - \left(\frac{i-d}{H-1-d}\right)^{\alpha}\right], \qquad \tau_i^{(k)} = \mathrm{clip}\left(\frac{1 - k/N - u_i}{1-u_i}, 0, 1\right), \quad i \geq d, \tag{11}$$

where u_i is the global-time threshold at which action i reaches its terminal local time, $u_0 \in [0, 1)$ controls the earliest completion time, and $\alpha > 0$ controls how completion times vary with action position. Here $\mathrm{clip}(z, 0, 1) = \min(1, \max(0, z))$, and we assume $d < H - 1$ so that at least two new action positions remain. For the supplied prefix, $\tau_i^{(k)} = 0$ for all $i < d$. With $u_0 = 1 - 1/N$, the first new action has $\tau_d^{(0)} = 1$ and $\tau_d^{(1)} = 0$: its full local integration interval is traversed in one update.

Scheduled Completion Does Not Guarantee Convergence. Reaching local time zero means that the allotted integration interval has ended; it does not guarantee that a finite-step prediction matches a more fully refined output. A short integration schedule may suffice for an easy observation while leaving appreciable residual error for a difficult one. Our pilot study examines this distinction using 10K action chunks from LIBERO and CALVIN. With the conditioning and initial noise held fixed, we compare released actions against the ten-step reference described in the main text. The observed deviations show that actions can still differ from the reference when they become eligible for release. This comparison measures agreement with a reference prediction, rather than error against ground-truth actions or a direct measure of task failure. To express this distinction, define the terminal step and released value by

$$k_i = \min\{k : \tau_i^{(k)} = 0\}, \qquad a_i^{\mathrm{release}} = x_i^{(k_i)}. \tag{12}$$

Let a_i^{ref} be the corresponding action from a separate ten-step reference prediction with the same conditioning and initial noise. A direct measure of their discrepancy is

$$e_i = \left\| a_i^{\mathrm{release}} - a_i^{\mathrm{ref}} \right\|_2, \qquad \tau_i^{(k_i)} = 0 \not\Rightarrow e_i = 0. \tag{13}$$

The norm is taken over the action dimensions in a common representation. The main-text relative-error metric additionally normalizes this discrepancy. The reference must be a separate prediction: running the same streaming trajectory for more steps cannot refine a token that has already frozen.

Frozen Predictions Remain Active Conditioning Context. Once an action reaches local time zero, its subsequent integration increments are zero, so later denoising steps cannot refine its value. Nevertheless, the action remains visible to the other action tokens through attention. The suffix is

therefore refined using the fixed, potentially imperfect prefix as context. FASTER provides per-token time conditioning, so the model can distinguish terminal-time tokens from tokens still being denoised. What this signal does not specify is the residual error of a terminal-time prediction: a token obtained through aggressive finite-step integration and a clean reference token can both have local time zero. Writing $v_{\theta,i}$ for the action expert's predicted velocity at position i, the numerical update is

$$x_i^{(k+1)} = x_i^{(k)} + \left(\tau_i^{(k+1)} - \tau_i^{(k)}\right) v_{\theta,i}\left(X^{(k)}, \boldsymbol{\tau}^{(k)}, c\right), \tag{14}$$

where $\boldsymbol{\tau}^{(k)}$ collects the local times of all tokens. After the terminal step, both local times are zero, giving

$$x_i^{(k+1)} = x_i^{(k)} = a_i^{\text{release}}, \qquad k \geq k_i. \tag{15}$$

The full chunk $X^{(k)}$ remains an input to the velocity field for the other positions. Thus, a frozen value can influence suffix updates without itself being updated. Equations above use exact terminal time for clarity. The implementation's readiness test $\tau_i < 0.01$ can release a token slightly before zero; exact freezing occurs when its local time reaches zero.

Training/Inference Mismatch. FASTER already trains with HAS timetables and action conditioning, including zero-time prefixes and heterogeneous times across action positions. The remaining mismatch concerns the values at those positions. Training constructs terminal-time actions from clean demonstration targets, whereas inference uses the model's own finite-step predictions. These include both newly finalized actions within a chunk and reused actions supplied by the previous chunk. Consequently, the suffix can encounter imperfect fixed context that is insufficiently represented by clean-prefix training. Because the suffix depends on this context, prefix deviations may affect subsequent predictions. The release-error study establishes the presence of such deviations; establishing their downstream amplification requires a separate controlled comparison of suffix predictions under clean and predicted prefixes. For a fixed prefix of length ℓ, the distinction can be written as

$$X_{\text{pre}}^{\text{train}} = (a_0^{\text{GT}}, \ldots, a_{\ell-1}^{\text{GT}}), \qquad X_{\text{pre}}^{\text{infer}} = (\hat{a}_0, \ldots, \hat{a}_{\ell-1}), \qquad \boldsymbol{\tau}_{\text{pre}} = \mathbf{0}, \tag{16}$$

where a_i^{GT} is a demonstrated action and $\hat{a}_i$ is a fixed model prediction, either newly finalized or reused from the preceding chunk. Both contexts carry terminal local time, although their values need not have the same error distribution.

A.2 Theoretical Modeling and Derivations

A.2.1 Reaction-Time Analysis and Feasible Execution Scheduling

This subsection derives the reaction time used in the main paper and the feasible range of the execution horizon. We use only four quantities. The controller runs with period Δt_{ctrl}; the policy uses N denoising steps; $t_{\text{FA}}(N)$ is the time from acquiring an observation to receiving the first executable action; and s is the number of actions executed before the next observation is acquired. The predicted action chunk contains H actions.

Feasible Execution Horizon. Suppose an inference request is sent at time t_0. Although its first action becomes available at $t_0 + t_{\text{FA}}(N)$, the controller can switch to that action only at a control tick. Therefore, the first usable tick is

$$t_0 + \left\lceil \frac{t_{\text{FA}}(N)}{\Delta t_{\text{ctrl}}} \right\rceil \Delta t_{\text{ctrl}}. \tag{17}$$

While waiting, the robot continues to execute actions from the current chunk. The current execution interval must therefore contain at least this many control steps. This gives the minimum feasible execution horizon

$$s_{\min}(N) = \left\lceil \frac{t_{\text{FA}}(N)}{\Delta t_{\text{ctrl}}} \right\rceil. \tag{18}$$

For example, if $\Delta t_{\text{ctrl}} = 40\,\text{ms}$ and $t_{\text{FA}} = 90\,\text{ms}$, the new action can be used only at the third control tick, so $s_{\min} = \lceil 90/40 \rceil = 3$.

The upper bound $s_{\max}$ is the maximum number of fresh actions that the system allows the robot to execute from one chunk; hence $s_{\max} \leq H$. In streaming execution, it is set after excluding any prefix reserved for overlapping the current inference. The feasible scheduling interval is thus

$$s_{\min}(N) \leq s \leq s_{\max}. \tag{19}$$

This interval exposes the basic scheduling trade-off. Increasing N can improve action quality, but it may increase $t_{\mathrm{FA}}(N)$ and therefore raise the smallest execution horizon that can run without a pause.

Reaction Time with Synchronous Inference. We first consider synchronous inference with a fixed execution horizon s. After acquiring an observation, the robot waits for the complete action chunk to be generated, executes s actions, and then acquires the next observation. Inference and execution therefore occur sequentially. For a fixed inference configuration, we use a constant-latency approximation with $t_{\mathrm{FA}} = \mathbb{E}[t_{\mathrm{FA}}]$. The interval between consecutive observations is then approximated by

$$T_{\mathrm{sync}} = t_{\mathrm{FA}} + s\Delta t_{\mathrm{ctrl}}. \tag{20}$$

Suppose an unexpected change occurs uniformly over this entire interval, including both inference and execution. Let w_{sync} denote the remaining time until the next observation. Under this approximation,

$$w_{\mathrm{sync}} \sim \mathcal{U}(0, T_{\mathrm{sync}}), \qquad \mathbb{E}[w_{\mathrm{sync}}] = \frac{1}{2} T_{\mathrm{sync}}. \tag{21}$$

The current action chunk cannot incorporate a change that occurs after its conditioning observation was acquired. The robot must therefore wait until the next observation and then complete another inference before an updated action becomes available. The expected reaction time is consequently approximated by

$$\begin{aligned} \mathbb{E}[\Delta t_{\mathrm{react}}]_{\mathrm{sync}} &\approx \frac{1}{2}\left(t_{\mathrm{FA}} + s\Delta t_{\mathrm{ctrl}}\right) + t_{\mathrm{FA}} \\ &= \frac{3}{2} t_{\mathrm{FA}} + \frac{1}{2} s\Delta t_{\mathrm{ctrl}}. \end{aligned} \tag{22}$$

Synchronous inference thus incurs two sources of delay: waiting for the next observation and generating an action conditioned on that observation. Because inference also postpones the next observation, it contributes an additional $\frac{1}{2} t_{\mathrm{FA}}$ to the mean feedback wait.

Reaction Time with Fixed Execution Horizons. We next consider a fixed execution horizon s. Two consecutive observations are separated by $s\Delta t_{\mathrm{ctrl}}$. Suppose an unexpected change occurs uniformly during this interval. Let w be the remaining time from the change to the next observation. Then

$$w \sim \mathcal{U}(0, s\Delta t_{\mathrm{ctrl}}). \tag{23}$$

After the next observation is acquired, the robot waits another t_{FA} for the first action that reflects the change. The complete reaction time is therefore

$$\Delta t_{\mathrm{react}} = w + t_{\mathrm{FA}}. \tag{24}$$

The mean remaining wait is

$$\mathbb{E}[w] = \frac{1}{s\Delta t_{\mathrm{ctrl}}} \int_0^{s\Delta t_{\mathrm{ctrl}}} w \, dw = \frac{1}{2} s\Delta t_{\mathrm{ctrl}}. \tag{25}$$

Substituting Eq. 25 into Eq. 24 gives

$$\boxed{\mathbb{E}[\Delta t_{\mathrm{react}}]_{\mathrm{fixed}} = \mathbb{E}[t_{\mathrm{FA}}] + \frac{1}{2} s\Delta t_{\mathrm{ctrl}}.} \tag{26}$$

The first term is the latency after the updated observation is acquired. The second term is the average time spent waiting for that observation. Thus, reducing TTFA alone does not minimize reaction time: the execution horizon also controls how quickly the policy can observe a change.

Reaction Time with Adaptive Execution Horizons. DASH changes the execution horizon from one cycle to another. Consider M consecutive cycles, where cycle i executes s_i actions and lasts $s_i\Delta t_{\mathrm{ctrl}}$. A change sampled uniformly in physical time is more likely to occur in a longer cycle. Therefore,

$$\Pr(\text{change occurs in cycle } i) = \frac{s_i}{\sum_{j=1}^{M} s_j}. \tag{27}$$

Let t_i denote the TTFA of the request issued after cycle i, namely the request whose observation can first contain that change. If the change occurs in cycle i, the mean reaction time is

$$t_i + \frac{1}{2} s_i \Delta t_{\mathrm{ctrl}}. \tag{28}$$

Weighting Eq. 28 by the probability in Eq. 27 gives

$$\begin{aligned} \mathbb{E}[\Delta t_{\mathrm{react}}] &= \sum_{i=1}^{M} \frac{s_i}{\sum_{j=1}^{M} s_j} \left(t_i + \frac{1}{2} s_i \Delta t_{\mathrm{ctrl}} \right) \\ &= \frac{\sum_{i=1}^{M} s_i t_i}{\sum_{i=1}^{M} s_i} + \frac{\sum_{i=1}^{M} s_i^2}{2 \sum_{i=1}^{M} s_i} \Delta t_{\mathrm{ctrl}}. \end{aligned} \tag{29}$$

This is the finite-sequence form used to understand the derivation: each cycle is weighted by the amount of physical time it occupies.

For a long sequence of cycles, the sums in Eq. 29 become expectations. Writing s and t_{FA} for the horizon and subsequent TTFA associated with a cycle, we obtain

$$\boxed{\mathbb{E}[\Delta t_{\mathrm{react}}]_{\mathrm{adapt}} = \frac{\mathbb{E}[s\, t_{\mathrm{FA}}]}{\mathbb{E}[s]} + \frac{\mathbb{E}[s^2]}{2\mathbb{E}[s]} \Delta t_{\mathrm{ctrl}}.} \tag{30}$$

Equation 30 is the general expression reported in the main paper. The first term is the duration-weighted TTFA, and the second term is the duration-weighted wait for the next observation.

When TTFA varies little across cycles, as under a fixed model, device, and streaming configuration, the first term is well approximated by $\mathbb{E}[t_{\mathrm{FA}}]$. The expression then simplifies to

$$\mathbb{E}[\Delta t_{\mathrm{react}}]_{\mathrm{adapt}} \approx \mathbb{E}[t_{\mathrm{FA}}] + \frac{\mathbb{E}[s^2]}{2\mathbb{E}[s]} \Delta t_{\mathrm{ctrl}}. \tag{31}$$

Finally, if every cycle uses the same horizon $s_i = s$, then $\mathbb{E}[s^2]/\mathbb{E}[s] = s$, and Eq. 30 reduces exactly to the fixed-horizon result in Eq. 26.

A.3 ALGORITHM

Algorithms 1 and 2 summarize the complete training and inference procedures. During training, we construct mixed-maturity action chunks that contain three types of action tokens: a clean RTC prefix inherited from previous execution, an additional perturbed prefix that mimics imperfect frozen predictions, and a postfix that remains under denoising. We randomly choose between the horizon-aware schedule and the constant schedule so that the policy is exposed to both position-dependent streaming trajectories and standard action-conditioning configurations. Gaussian perturbations are applied only to the additional frozen prefix, while the clean RTC prefix is kept unchanged. The prefix tokens serve only as conditioning context, and the flow-matching objective is evaluated on the postfix positions. This training procedure exposes the action expert to imperfect and mixed-maturity contexts that more closely resemble those encountered during streaming inference.

Algorithm 1 Policy Training

Require: Training dataset $\mathcal{D}$, pretrained VLA model v_θ, prediction horizon H, HAS parameters α, u_d, mixing probability p, maximum RTC prefix length $d_{\max} < H$, prefix perturbation scales $\{\sigma_i\}_{i=0}^{H-1}$

Ensure: Fine-tuned VLA model v'_θ

for each training step **do**

Sample data $(\mathbf{o}_t, \hat{\mathbf{A}}_t) \sim \mathcal{D}$, noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, global timestep $\rho \sim \mathcal{U}(0,1)$, RTC prefix length $d_{\mathrm{RTC}} \sim \mathrm{Unif}\{0, \dots, d_{\max}\}$, and schedule type $z \sim \mathrm{Bernoulli}(p)$

Sample noisy-prefix length d_{noise} with $0 \le d_{\mathrm{noise}} \le H - 1 - d_{\mathrm{RTC}}$; set $d = d_{\mathrm{RTC}} + d_{\mathrm{noise}}$

Compute the action prefix mask $\mathbf{m}$:

$$m_i = \mathbf{1}(i \ge d), \qquad i = 0, \dots, H-1.$$

if $z = 1$ **then** ▷ Horizon-Aware Schedule

Compute hit times:

$$u_i = \left[1 - \left(\frac{i-d}{\max(H-1-d, 1)}\right)^{\alpha}\right] u_d, \qquad i = d+1, \dots, H-1.$$

Compute local timesteps $\boldsymbol{\tau}$:

$$\tau_i = \begin{cases} 0, & i < d, \\ \max\left(0, \dfrac{\rho - u_i}{1 - u_i}\right), & i \ge d. \end{cases}$$

else ▷ Constant Schedule with Action Conditioning

Set local timesteps $\boldsymbol{\tau}$:

$$\tau_i = \begin{cases} 0, & i < d, \\ \rho, & i \ge d. \end{cases}$$

end if

Perturb only the noisy prefix: sample $\boldsymbol{\delta}_i \sim \mathcal{N}(\mathbf{0}, \sigma_i^2 \mathbf{I})$ and set

$$\widetilde{\mathbf{A}}_{t,i} = \hat{\mathbf{A}}_{t,i} + \mathbf{1}(d_{\mathrm{RTC}} \le i < d)\, \boldsymbol{\delta}_i.$$

Construct noisy actions:

$$\mathbf{A}_t^{\boldsymbol{\tau}} = \boldsymbol{\tau} \odot \boldsymbol{\epsilon} + (1 - \boldsymbol{\tau}) \odot \widetilde{\mathbf{A}}_t.$$

Compute the masked loss:

$$\mathcal{L}(\theta) = \frac{\left\| \mathbf{m} \odot \left[v_\theta(\mathbf{o}_t, \mathbf{A}_t^{\boldsymbol{\tau}}, \boldsymbol{\tau}) - \left(\boldsymbol{\epsilon} - \hat{\mathbf{A}}_t\right) \right] \right\|_2^2}{\|\mathbf{m}\|_1}.$$

Update model parameters θ using $\nabla_\theta \mathcal{L}(\theta)$

end for

Algorithm 2 Policy Inference

Require: Fine-tuned VLA model v'_θ, multi-head difficulty predictor $\{f_{\phi,g}\}$, budget mapping G, maturity MLP MLP_ψ, maximum schedule length $\bar{N}$, prediction horizon H, HAS parameters α, u_d, observation $\mathbf{o}_t$, delay d, action prefix $\mathbf{A}^{\mathrm{pre}}$, baseline execution horizon s_{fixed}, proportional gain K_p, control period Δt_{ctrl}, and profiled TTFA $t_{\mathrm{FA}}(N)$

Initialize $\mathbf{A}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and overwrite $\mathbf{A}_t[0:d-1]$ with $\mathbf{A}^{\mathrm{pre}}$

Forward the VLM backbone with $\mathbf{o}_t$ and cache its features

Set initial local timesteps $\tau_i^1 = \mathbf{1}(i \geq d)$

Run the first denoising forward pass with $\mathbf{B}^1 = \mathbf{0}$ to obtain velocity $\mathbf{v}^1$ and last-layer action-expert hidden states $\mathbf{Z}^1$

Predict difficulty with the multi-head MLP:

$$\mathbf{h}^1 = \mathrm{MeanPool}(\mathbf{Z}^1), \qquad r = \max_{g \in \{\mathrm{trans,rot,grip}\}} \mathrm{sigmoid}\big(f_{\phi,g}(\mathbf{h}^1)\big)\,.$$

Select the denoising budget $N = G(r)$ ▷ Budget mapping

Adjust the execution horizon:

$$s = \max\{s_{\min}(N), \mathrm{round}(s_{\mathrm{fixed}} - K_p r)\}\,, \qquad s_{\min}(N) = \left\lceil \frac{t_{\mathrm{FA}}(N)}{\Delta t_{\mathrm{ctrl}}} \right\rceil.$$

Plan global timesteps using the predicted budget:

$$\rho^j = \frac{N - j + 1}{N}, \qquad j = 1, \ldots, N+1.$$

Compute hit times:

$$u_i = \left[1 - \left(\frac{i-d}{\max(H-1-d, 1)}\right)^{\alpha}\right] u_d, \qquad i = d+1, \ldots, H-1.$$

Compute each token's planned denoising steps:

$$n_i = \min\big\{k \in \{1, \ldots, N\} \mid \rho^{k+1} \leq u_i\big\}\,, \qquad i \geq d.$$

for $j = 1$ to N **do**

  Compute local timesteps $\boldsymbol{\tau}^j$ with respect to ρ^j:

$$\tau_i^j = \begin{cases} 0, & i < d, \\ \max\left(0, \dfrac{\rho^j - u_i}{1 - u_i}\right), & i \geq d. \end{cases}$$

  Similarly, compute $\boldsymbol{\tau}^{j+1}$ with respect to ρ^{j+1} ▷ $\rho^{N+1} = 0$

  Overwrite $\mathbf{A}_t[0:d-1]$ with $\mathbf{A}^{\mathrm{pre}}$

  if $j > 1$ **then**

    Construct the normalized budget condition:

$$\mathbf{B}_i^j = \begin{cases} [1, 0], & i < d, \\ \left[\dfrac{n_i}{\bar{N}}, \dfrac{\max(n_i - j + 1, 0)}{\bar{N}}\right], & i \geq d. \end{cases}$$

    Inject the budget condition into action embeddings and predict velocity:

$$\mathbf{e}_i^j = E(\mathbf{A}_{t,i}) + \mathrm{MLP}_\psi(\mathbf{B}_i^j), \qquad \mathbf{v}^j = v_\theta\big(\mathbf{o}_t, \mathbf{A}_t, \boldsymbol{\tau}^j; \mathbf{B}^j\big)\,.$$

  end if ▷ Reuse the first-pass velocity when $j = 1$

  Compute delta timestep $\Delta\boldsymbol{\tau} = \boldsymbol{\tau}^{j+1} - \boldsymbol{\tau}^j$

  Euler update $\mathbf{A}_t \leftarrow \mathbf{A}_t + \mathbf{v}^j \odot \Delta\boldsymbol{\tau}$

  for each valid action index $i \in [d, H-1]$ **do**

    if $\tau_i^{j+1} = 0$ **and** action i has not been streamed **then**

      Dispatch $\mathbf{a}_{t+i}$ to the client immediately ▷ Streaming interface

    end if

  end for

  if all actions within execution horizon $[d, d+s-1]$ have been finalized **then**

    break ▷ Early stopping

  end if

end for

A.4 Experimental Protocols

A.4.1 Training Protocol

Baselines. The vanilla Synchronous inference (Sync) waits for the complete action chunk before execution. Temporal Ensembling (TE) (Zhao et al., 2023) maintains overlapping action chunks and executes an exponentially weighted average of the predictions aligned with the current control timestep. Training-time RTC (Black et al., 2025a;b) generates the next action chunk asynchronously while conditioning on overlapping actions from the currently executing chunk, with the action-conditioning mechanism learned during training time. StreamingVLA (Shi et al., 2026) combines state-based action flow matching with adaptive early observation for streaming execution. FASTER (Lu et al., 2026) uses a horizon-aware denoising schedule to release prefix actions before the full action chunk is completed.

Protocol Overview. We evaluate two flow-based VLA backbones, $\pi_{0.5}$ and X-VLA, on LIBERO and CALVIN. For LIBERO, we use the `physical-intelligence/libero` dataset and evaluate on the LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-10 task suites. For CALVIN, we use the ABC training split and evaluate on split D. We compare DASH with synchronous inference, RTC, FASTER, and StreamingVLA. The following paragraphs provide the training and evaluation protocols.

Sync and RTC. For Sync, we use the standard implementation from OpenPI. For RTC, we use the implementation released with FASTER. On LIBERO, Sync uses the official $\pi_{0.5}$-LIBERO checkpoint provided by OpenPI. We initialize Training-time RTC from $\pi_{0.5}$-LIBERO checkpoint and continue fine-tuning for 10,000 steps. On CALVIN, we first obtain a shared $\pi_{0.5}$-CALVIN checkpoint by LoRA fine-tuning $\pi_{0.5}$-base on `InternRobotics/InternData-Calvin_ABC` for 30,000 steps. Starting from this checkpoint, we continue fine-tuning both Sync and RTC for another 10,000 steps with their respective training objectives. This gives both methods the same initialization and additional training budget. For X-VLA, Sync directly uses the official `2toINF/X-VLA-Libero` checkpoint on LIBERO and the official `2toINF/X-VLA-Calvin-ABC_D` checkpoint on CALVIN without additional fine-tuning. RTC is initialized from the same benchmark-specific checkpoints and further fine-tuned for 10,000 and 10,000 steps on LIBERO and CALVIN, respectively, using the RTC training objective. At inference time, Sync uses the standard action-generation interface, while RTC uses the asynchronous action-chunk controller in the X-VLA codebase.

FASTER. We use the implementation released by FASTER. On LIBERO, we initialize FASTER from the official $\pi_{0.5}$-LIBERO checkpoint and continue LoRA fine-tuning for 10,000 steps with the FASTER training objective. On CALVIN, we initialize FASTER from the shared $\pi_{0.5}$-CALVIN checkpoint obtained after the first 30,000 training steps. We then continue LoRA fine-tuning for another 10,000 steps with the same FASTER objective. FASTER therefore uses the same benchmark-specific initialization and additional training budget as Sync and RTC. For X-VLA, we initialize FASTER from the official `2toINF/X-VLA-Libero` checkpoint on LIBERO and the official `2toINF/X-VLA-Calvin-ABC_D` checkpoint on CALVIN. We apply LoRA fine-tuning for 30,000 steps with the FASTER training objective on both benchmarks. During training, we replace the shared flow timestep with the per-action HAS schedule with probability 0.5. We set $\alpha = 0.7$, $u_0 = 0.9$, and the maximum delay to 0. We apply LoRA to the Transformer with rank 8 and alpha 16. The action encoder, action decoder, and soft prompt remain trainable. We use a batch size of 128 and a learning rate of 1×10^{-4}.

StreamingVLA. We use the implementation released by StreamingVLA. On LIBERO, we directly evaluate the checkpoint provided in their project without additional fine-tuning. On CALVIN, we obtain a separate StreamingVLA checkpoint by LoRA fine-tuning $\pi_{0.5}$-base on `InternRobotics/InternData-Calvin_ABC` for 30,000 steps.

DASH. DASH uses the same benchmark-specific initialization and additional fine-tuning budget as RTC and FASTER. On LIBERO, we initialize DASH from the official $\pi_{0.5}$-LIBERO checkpoint and apply LoRA fine-tuning for 10,000 steps. On CALVIN, we initialize DASH from the shared $\pi_{0.5}$-CALVIN checkpoint obtained after the first 30,000 training steps and apply LoRA fine-tuning for another 10,000 steps. In both settings, we train DASH with horizon-calibrated step

conditions. For X-VLA, we initialize DASH from the official `2toINF/X-VLA-Libero` and `2toINF/X-VLA-Calvin-ABC_D` checkpoints. We LoRA fine-tune each model for 30,000 steps using 10 step conditions and a readiness threshold of 0.01. We use the same LoRA and optimization settings as X-VLA FASTER, while jointly training the step-condition MLP with the action policy. To train the difficulty predictor, we freeze the policy and extract the mean and standard deviation of the last-layer action-token hidden states from the first denoising pass. We define the oracle budget as the smallest $k \in [1, 10]$ whose action chunk has a relative ℓ_2 deviation below 0.02 from the 10-step reference. On CALVIN, we collect 10,000 records and use a 90/10 training-validation split. We train a lightweight MLP with the Smooth L1 loss for up to 80 epochs using a batch size of 512, a learning rate of 5×10^{-4}, and a weight decay of 1×10^{-4}. We apply early stopping with a patience of 20 and select the checkpoint with the lowest validation MAE.

A.4.2 Evaluation Protocol

Task Performance. We evaluate task performance on the LIBERO (Liu et al., 2023) and CALVIN (Mees et al., 2022) benchmarks. LIBERO consists of four task suites—Spatial, Object, Goal, and Long-Horizon (LIBERO-10)—that evaluate different aspects of robotic manipulation. We train one policy across all four suites. Each suite contains 10 tasks, and each task is evaluated over 50 trials. We report the success rate for each suite and the average success rate across all four suites. For CALVIN, we adopt the widely used ABC→D evaluation protocol, which evaluates generalization from the ABC training environments to environment D. Each policy is evaluated on 1,000 unique instruction chains, with each chain containing five consecutive tasks. We report the success rates for completing at least one to five tasks and use the average number of successfully completed tasks per chain as the primary metric.

Performance-Responsiveness Evaluation. To evaluate the trade-off between task performance and responsiveness beyond a single operating point, we compare DASH and FASTER on CALVIN ABC→D using the $\pi_{0.5}$ backbone. We vary FASTER's fixed execution horizon s and DASH's base execution horizon s_{fixed} over $\{3, \ldots, 8\}$, while fixing $K_p = 5$ for DASH. Each configuration follows the task evaluation protocol described above. Expected reaction latency is estimated using the timing model in Sec. 4.1, accounting for the fixed execution horizon in FASTER and the state-dependent execution horizons in DASH. Figure 4 plots the average number of completed tasks per instruction chain against the corresponding expected reaction latency, enabling comparison across execution-horizon configurations rather than only at the same nominal horizon.

A.4.3 Inference Overhead

Profiling Setup. We profile the inference overhead of the MLP-enabled DASH implementation with the $\pi_{0.5}$ backbone on an NVIDIA PRO 6000D. We report three timing measurements: *policy inference*, the time spent in the policy inference call; *server inference*, the inference duration recorded on the server; and *client wall time*, the elapsed time from submitting a request to receiving its complete result at the client. The last metric captures end-to-end request latency, including communication and request-handling costs. For each measurement, we report the absolute latency increase $\Delta T = T_{\text{DASH+MLP}} - T_{\text{original}}$ and the relative overhead $\Delta T / T_{\text{original}} \times 100\%$.

Table 4: **Inference overhead with $\pi_{0.5}$ on NVIDIA PRO 6000D.** Latencies and absolute increases are reported in milliseconds. Relative overhead is computed against the original ten-step policy.

Metric	Original 10-step	DASH + MLP	Increase	Overhead
Policy inference	68.812	70.444	+1.632	+2.37%
Server inference	72.049	73.896	+1.847	+2.56%
Client wall time	72.517	74.194	+1.677	+2.31%

Results. Table 4 shows that the profiled DASH implementation introduces a modest latency increase. Policy inference latency increases by 1.632 ms (2.37%), while server inference latency increases by 1.847 ms (2.56%). At the client, end-to-end request latency increases from 72.517 to 74.194 ms, corresponding to an additional 1.677 ms (2.31%). Thus, the measured overhead remains

below 2.6% across all three timing scopes under this profiling configuration. These measurements characterize the net latency difference between the two implementations, rather than the isolated execution time of the MLP.

A.4.4 Ablation Study

Component-wise Ablation of MOM. Maturity-Oriented Matching (MOM) combines two components: prefix perturbation, which exposes the policy to imperfect frozen action context during training, and maturity conditioning, which encodes each token's planned and remaining denoising steps. To examine how these two components combine, we compare three variants: prefix perturbation only, maturity conditioning only, and full MOM. Table 5 reports the results. We evaluate the variants using the $\pi_{0.5}$ backbone on CALVIN ABC→D. To isolate the effects of MOM from adaptive scheduling, we disable DAD in this ablation and use the same fixed denoising budget N_{eval} and execution horizon s_{eval} for all variants. Each variant is evaluated on the same 1,000 instruction chains using five evaluation seeds. Full MOM achieves the best Avg. Len (4.355), slightly outperforming maturity conditioning only (4.342) and prefix perturbation only (4.330). This suggests that the two components are complementary in this setting, with maturity conditioning alone providing a slightly larger gain than prefix perturbation alone. Given the small margins, seed-level standard deviations and significance tests should be considered.

Table 5: Component-wise ablation of MOM on CALVIN ABC→D with $\pi_{0.5}$. Prefix perturbation and maturity conditioning are independently enabled or disabled. All variants use the same fixed denoising budget N_{eval} and execution horizon s_{eval}, with DAD disabled. Avg. Len denotes the average number of completed tasks per instruction chain.

Variant	Prefix perturbation	Maturity conditioning	CALVIN Avg. Len ↑
(a) Prefix perturbation only	✓	×	4.330
(b) Maturity conditioning only	×	✓	4.342
(c) Full MOM	✓	✓	4.355

Effect of State-Dependent Scheduling. To further examine whether the performance gains of DASH truly arise from state-dependent adaptive scheduling, rather than merely from using a more favorable combination of denoising budgets and execution horizons, we construct a *shuffled scheduling* control. Specifically, we first run the full DASH policy on an independent set of calibration trajectories and record the (N, s) configurations selected according to the predicted difficulty, where N denotes the denoising budget and s denotes the execution horizon. We then randomly shuffle the correspondence between these (N, s) configurations and the current robot states, such that the scheduling decision no longer depends on the predicted difficulty of the current state. Except for the scheduling strategy, both settings use the same $\pi_{0.5}$ checkpoint, MOM module, task set, and evaluation protocol, and are evaluated on CALVIN ABC→D using the same 1,000 instruction chains and five evaluation seeds.

As shown in Table 6, shuffled scheduling uses 6.34 denoising steps on average, which is close to the 6.75 steps used by DASH, but reduces Avg. Len from 4.474 to 4.335 while increasing the expected reaction latency from 125.79 ms to 132.85 ms. These results show that, under comparable computation budgets, full DASH achieves better task performance and responsiveness. This suggests that its gains do not arise solely from using a favorable overall budget and execution-horizon configuration, but also from assigning computation and feedback opportunities to more appropriate states according to the current predicted difficulty.

A.5 Real-World Experimental Setup and Results

We evaluate real-world manipulation using a dynamic pick-and-place task. The experiments examine whether an inference method can preserve task completion while responding to a change in the target or placement location. We report the physical platform, policy preparation, perturbation protocol, and success rates for Sync, training-time RTC, FASTER, and DASH.

Table 6: Ablation of state-dependent scheduling on CALVIN ABC→D with $\pi_{0.5}$. Shuffled scheduling preserves the overall distribution of (N, s) configurations collected from DASH, while randomly reassigning them across robot states. All variants use the same policy checkpoint and MOM module.

Method	$\mathbb{E}[\Delta t_{\text{react}}]$ (ms) ↓	Avg. Len ↑
Shuffled Scheduling	132.85	4.335
DASH	125.79	**4.474**

A.5.1 Robot Platform and Policy Preparation

Robot and sensing. The experiments use an AgileX X robot with seven controlled degrees of freedom, including the gripper. Two Intel RealSense D435 cameras provide visual observations: a static camera observes the workspace, and a wrist-mounted camera provides a view near the end effector, as illustrated in Fig. 5 of the main text. The controller operates at 30 Hz, corresponding to a control period of approximately 33.3 ms. Each action has seven components, including gripper control. The single-arm data pipeline represents the six arm-joint actions as offsets from the observed joint state and converts predictions back to absolute joint targets during output processing. The gripper component retains an absolute representation.

Inference deployment. A Jetson Orin client interfaces with the robot controller, while a remote GPU server performs policy inference. All real-world success rates reported in Fig. 5 are obtained using an NVIDIA PRO 6000D inference server. Asynchronous methods overlap action generation with robot execution; streaming methods additionally allow the client to receive executable actions before the complete chunk is generated. The methods use the same control frequency, prediction horizon, and camera configuration.

Policy and demonstrations. The real-world policy is based on a fine-tuned $\pi_{0.5}$ backbone. We collect 300 demonstrations for the real-world task and fine-tune the single-arm policy for 10,000 steps with a global batch size of 128. LoRA is enabled in both the VLM and the action expert, with ranks of 16 and 32, respectively, following the corresponding model variants in the implementation. The predicted action chunk contains $H = 50$ actions, and the default denoising budget is ten steps. The RTC comparison uses training-time RTC.

A.5.2 Task and Perturbation Protocol

The robot must grasp a wooden block and place it into a green box. We evaluate three conditions, each with 50 trials per method:

- **No perturbation.** Neither the wooden block nor the green box is deliberately relocated during execution.
- **Target relocation.** The wooden block is displaced by 5 cm, requiring the policy to update the target location.
- **Receptacle relocation.** The green box is displaced by 5 cm, requiring the policy to update the placement destination.

The relocation uses the same designated destination for a given condition, rather than varying the displacement across trials. The block and box are relocated in separate conditions; simultaneous relocation is not tested. Each trial in a relocation condition contains exactly one relocation.

A.5.3 Evaluation Protocol

Trial accounting. Each method is evaluated for 50 trials in each of the three conditions, yielding 150 trials per method and 600 trials across the four methods. A successful trial requires the robot to grasp the wooden block and place it into the green box. For each condition, success rate is the number of successful trials divided by 50. Each successful trial therefore contributes two percentage points. The aggregate success rate is computed over the 150 trials for each method; because the conditions have equal trial counts, this is also the arithmetic mean of their three success rates.

Table 7: Real-world pick-and-place results with $\pi_{0.5}$ on NVIDIA PRO 6000D. Each method is evaluated on 50 trials per condition. Count denotes successful trials over total trials; SR denotes success rate (%). Overall aggregates all 150 trials. RTC denotes training-time RTC. Bold indicates the highest success rate in each group.

Method	No Perturbation		Target Relocation		Receptacle Relocation		Overall	
	Count	SR	Count	SR	Count	SR	Count	SR
Sync	45/50	90.0	12/50	24.0	9/50	18.0	66/150	44.0
Training-time RTC	48/50	**96.0**	15/50	30.0	11/50	22.0	74/150	49.3
FASTER	43/50	86.0	35/50	70.0	38/50	76.0	116/150	77.3
DASH	46/50	92.0	41/50	**82.0**	42/50	**84.0**	129/150	**86.0**

A.5.4 Real-World Responsiveness

Task success measures whether a controller eventually recovers from a perturbation, but does not directly quantify how quickly the robot reacts to the environmental change. Following the real-time evaluation protocol in the main paper, we further characterize the responsiveness of each method under the same real-robot deployment setting used in the above experiments. The robot operates at 30 Hz, corresponding to a control period of 33.3 ms, and all methods are evaluated with the same $\pi_{0.5}$ policy backbone and NVIDIA PRO6000D inference server. We report Time to First Action (TTFA), the minimum feasible execution horizon $s_{\min}$, and the expected reaction latency $\mathbb{E}[\Delta t_{\text{react}}]$. Following the reaction-time formulation, the latter accounts for both the latency required to obtain the first executable action and the waiting time until the next observation is acquired. We additionally report the reaction speedup relative to synchronous inference.

Table 8: Real-time responsiveness on the physical robot with $\pi_{0.5}$ and NVIDIA PRO 6000D. TTFA denotes Time to First Action, $s_{\min}$ is the minimum feasible execution horizon, $\mathbb{E}[s_{\text{exec}}]$ is the mean execution horizon, and $\mathbb{E}[\Delta t_{\text{react}}]$ denotes the expected reaction latency under the real-robot deployment setting. Speedup is measured relative to synchronous inference. Lower is better for TTFA, $s_{\min}$, and reaction latency.

Method	TTFA (ms) ↓	$s_{\min}$ ↓	$\mathbb{E}[s_{\text{exec}}]$	$\mathbb{E}[\Delta t_{\text{react}}]$ (ms) ↓	Speedup ↑
Sync	$80.43_{\pm 0.61}$	3	10	287.31	1.00×
Training-time RTC	$80.24_{\pm 0.77}$	3	7	196.91	1.46×
FASTER	$56.90_{\pm 2.12}$	2	5	140.23	2.05×
DASH	$\mathbf{51.21}_{\pm 2.38}$	**2**	**2.84**	**124.54**	**2.31**×

Results. As shown in Table 8, DASH achieves the lowest expected reaction latency of 124.54 ms, corresponding to a 2.31× speedup over synchronous inference. DASH further reduces the overall reaction latency by adapting the execution horizon according to the predicted state difficulty. This result confirms that minimizing TTFA alone is insufficient for optimal closed-loop responsiveness: observation refresh frequency is also a critical factor in determining how quickly environmental changes can affect robot execution.

Qualitative Observations. In the target- and receptacle-relocation conditions, we qualitatively observed earlier corrective motion with DASH than with the evaluated fixed-horizon configurations. Following target relocation, the robot began redirecting its reaching motion toward the displaced wooden block sooner; following receptacle relocation, it began adjusting its placement trajectory toward the relocated box sooner. These observations suggest that adaptive execution can help the robot correct its ongoing motion more promptly after environmental changes, reducing the period during which it continues to follow an outdated trajectory. This qualitative behavior complements the model-based responsiveness estimates, although the time from perturbation onset to physical correction was not directly measured.