FASTER: Rethinking Real-Time Flow VLAs

Yuxiang Lu[1,2] **Zhe Liu**[1*] **Xianzhe Fan**[1] **Zhenya Yang**[1] **Jinghua Hou**[1]
Junyi Li[1] **Kaixin Ding**[1] **Hengshuang Zhao**[1†]
[1]The University of Hong Kong [2]ACE Robotics
https://hku-sail.github.io/FASTER

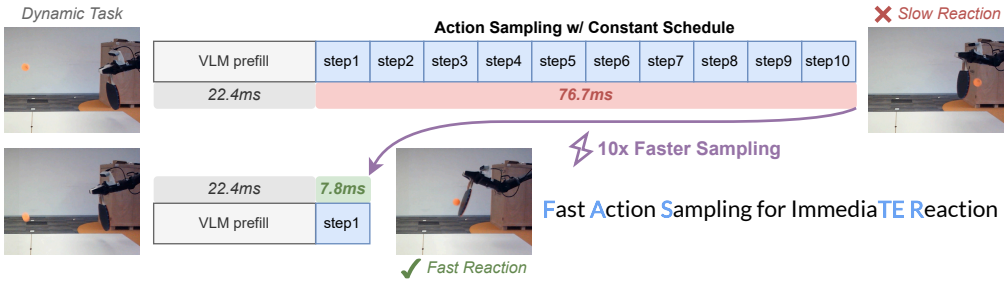


Figure 1: We propose FASTER to alleviate the reaction latency bottleneck in action chunking flow policies. By compressing the sampling iterations of the immediate reaction into a single step, FASTER (bottom) achieves a **10× acceleration** compared to original $\pi_{0.5}$ and X-VLA (top). This enables real-time responsiveness in highly dynamic real-world tasks such as playing table tennis. Moreover, FASTER is a plug-and-play solution for flow-based VLAs that integrates seamlessly into the standard fine-tuning pipeline and requires no architectural modifications.

Abstract

Real-time execution is crucial for deploying Vision-Language-Action (VLA) models in the physical world. Existing asynchronous inference methods primarily optimize trajectory smoothness, but neglect the critical latency in reacting to environmental changes. By rethinking the notion of reaction in action chunking policies, this paper presents a systematic analysis of the factors governing reaction time. We show that reaction time follows a uniform distribution determined jointly by the Time to First Action (TTFA) and the execution horizon. Moreover, we reveal that the standard practice of applying a constant schedule in flow-based VLAs can be inefficient and forces the system to complete all sampling steps before any movement can start, forming the bottleneck in reaction latency. To overcome this issue, we propose **F**ast **A**ction **S**ampling for Immedia**TE** **R**eaction (**FASTER**). By introducing a Horizon-Aware Schedule, FASTER adaptively prioritizes near-term actions during flow sampling, compressing the denoising of the immediate reaction by tenfold (*e.g.*, in $\pi_{0.5}$ and X-VLA) into a single step, while preserving the quality of long-horizon trajectory. Coupled with a streaming client-server pipeline, FASTER substantially reduces the effective reaction latency on real robots, especially when deployed on consumer-grade GPUs. Real-world experiments, including a highly dynamic table tennis task, prove that FASTER unlocks substantially improved real-time responsiveness for generalist policies, enabling rapid generation of accurate and smooth trajectories.

*Project Leader
†Corresponding Author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

The paradigm of robot learning is undergoing a profound transformation with the advent of Vision-Language-Action (VLA) models [115, 78, 65, 77]. By formulating continuous motor control as a generative sequence modeling problem, recent approaches leveraging diffusion models [30, 84] and flow matching [51] for action chunking have achieved unprecedented capabilities in dexterous robotic manipulation tasks [34, 2, 120, 102]. As research focus shifts from simulation to real-world physical deployment, real-time capability has become increasingly paramount.

Existing real-time execution methods primarily address the "stop-and-wait" issue in standard synchronous inference [16] for action chunking policies [82]. By introducing an asynchronous pipeline, the robot can initiate the next inference request before the current action chunk is exhausted, thereby eliminating inter-chunk pauses and enhancing motion continuity [4]. While state-of-the-art advances in asynchronous inference strategy [5, 94, 89, 58, 119] further reinforce trajectory smoothness, these methods largely overlook another essential dimension of real-time embodied intelligence: **reaction**. Beyond smooth execution, a practical VLA system must promptly and precisely respond to dynamically changing physical environments. Delayed reactions to unexpected perturbations create a perilous "blind spot" in closed-loop control, limiting the effectiveness of generalist policies in open-world scenarios.

Our in-depth analysis of the inference pipeline in Section 2 reveals that reaction time is not a trivial constant determined by inference latency. Instead, it should be modeled as a random variable following a uniform distribution, due to the stochastic timing of external events relative to robot controller. We further illustrate that existing asynchronous methods are inherently limited, and a collaborative enhancement in both perception-execution latency and the frequency of inference-execution cycle is entailed to acquire truly responsive behavior.

We then revisit a common design in flow-based VLAs: a constant timestep schedule across the action chunk, which allocates an equal number of sampling steps to every action. Under this scheme, the full multi-step denoising process must be completed before any action can be dispatched, severely inflating the reaction delay. Considering the intrinsic causal structure of physical interaction, near-term actions are more tightly coupled with current observations and typically lie in a significantly narrower solution space. Our pilot study in Section 3.2 supports this intuition. We clearly observe that early actions follow straighter interpolation paths and attain precise estimation of clean actions within only a few sampling steps, whereas a constant schedule over-samples these dimensions. This naturally raises a key question: *since earlier actions are easier to predict than later ones, can flow-based VLAs generate these latency-critical actions with fewer sampling steps for immediate reaction?*

To address these challenges, we propose **F**ast **A**ction **S**ampling for Immedia**TE** **R**eaction (**FASTER**), a simple yet effective method applicable to flow-based VLAs [3, 120] without architectural modifications or additional training cost. As shown in Figure 1, FASTER aims to accelerate the sampling process of leading actions, as quantified by the newly introduced Time to First Action (TTFA) metric for reactivity. Concretely, a Horizon-Aware Schedule (HAS) is incorporated to decouple the local denoising timestep for each frame within the chunk. HAS adaptively allocates more aggressive sampling steps to near-term actions while maintaining a slower schedule for long-horizon ones. Consequently, the model can output the immediate action as fast as one-step sampling, while largely preserving long-term trajectory accuracy.

Beyond algorithmic acceleration, FASTER also catalyzes a paradigm shift from conventional asynchronous pipeline to a streaming client-server interaction, wherein early actions can be dispatched to the robot controller instantly upon completion. While the robot executes these initial movements, the VLA model continues refining subsequent actions in parallel and progressively replenishes the client's action buffer. Real-world evaluations on two GPU platforms (*i.e.*, RTX 4060 and RTX 4090) demonstrate that FASTER substantially reduces inference latency, as reflected by lower TTFA, while simultaneously boosting inference-execution cycle frequency through the synergization of streaming output and early-stopping strategies. Real-robot experiments further confirm the superior reaction capability of FASTER, even when deployed on resource-constrained GPUs, offering a general and promising path toward genuinely real-time VLAs.

Our contributions are summarized as: (1) We present a systematic analysis of reaction attributes in action chunking VLA policies, revealing the inherent limitations of existing methods for real-time responsiveness. (2) We propose the FASTER framework, capitalizing on a Horizon-Aware Schedule

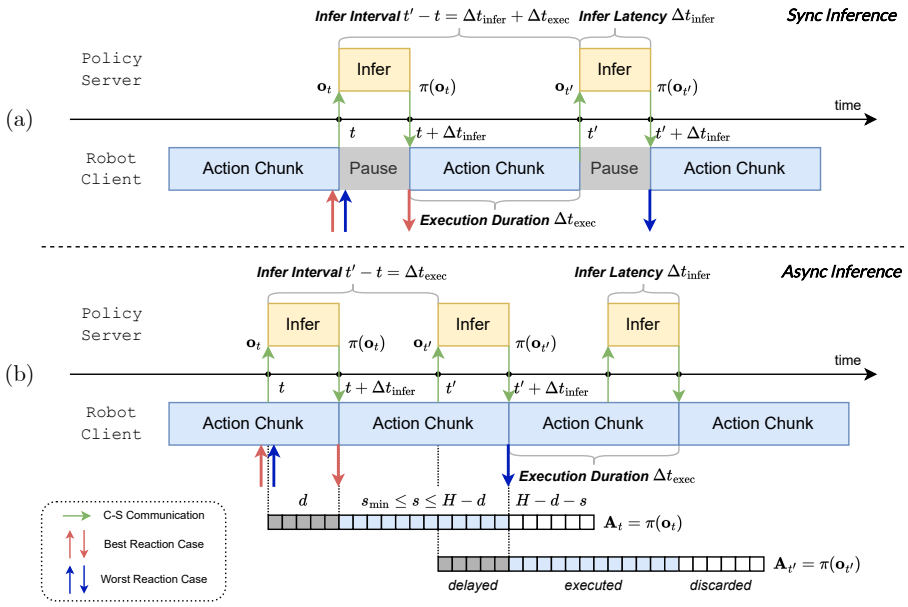


Figure 2: Temporal pipelines of (a) synchronous and (b) asynchronous inference in a robotic system composed of an action chunking policy server and a robot client. As indicated by the best and worst cases, reaction time depends on both inference latency and the interval between consecutive inference-execution cycles. We also illustrate the decomposition of two adjacent action chunks to clarify the discretized inference delay d and the execution horizon s (bounded by $s_{\min}$ and $H - d$) in the asynchronous client.

that prioritizes immediate actions during flow matching sampling, effectively lowering TTFA while progressively refining later actions. (3) We design a streaming client-server interface with early stopping, jointly trimming the delay and accelerating the closed loop of inference-execution. (4) Extensive experiments on real robots demonstrate significant improvements in reaction capability and promising performance in dexterous action generation for manipulation tasks.

2 Analysis on Action Chunking Policy Inference

Action chunking is a standard method in VLA policies [16, 118, 3]. Given a policy π, the model processes an observation $\mathbf{o}_t$ at real-world time t to predict a sequence of future actions $\mathbf{A}_t = [\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots, \boldsymbol{a}_{t+H-1}]$, where H denotes the *prediction horizon* specified by the policy. In practice, instead of executing the entire action chunk, it is common to execute only s actions, then trigger a new inference and discard the remaining actions. s is referred to as the *execution horizon* [4].

Deploying a VLA policy on a physical robot typically utilizes a client-server architecture, consisting of a policy server for model inference and a robot client for motor control. After initialization, the server remains active to process incoming requests from the client and returns predictions with a certain latency, giving rise to two interaction paradigms: synchronous and asynchronous inference.

We first define several time quantities to assist analysis of the pipeline:

- *Control period* $\Delta t_{\text{ctrl}} := 1/f$. In robotic systems, the controller operates at a specific frequency f (*e.g.*, 30Hz), corresponding to a fixed period between consecutive operations, such as executing action or triggering inference.
- *Inference latency* Δt_{infer}. This is defined as the time interval between transmitting an observation and receiving the predicted actions on the client side. It encompasses model inference, network communication, pre- and post-processing, memory I/O, and other system overheads. For analytical convenience, we model the total latency as a constant. Following prior work [4], we also define the discretized inference delay as $d := \lfloor \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rfloor$.
- *Execution duration* $\Delta t_{\text{exec}} := s \cdot \Delta t_{\text{ctrl}}$. This denotes the time required for the robot client to execute s actions.

Table 1: Reaction characteristics of synchronous and asynchronous inference. $\mathcal{U}(a, b)$ denotes a uniform distribution with lower and upper bounds a and b, so the expectation of Δt_{react} equals the midpoint $(a + b)/2$.

Mode	Infer Interval	$\Delta t_{\text{react}} \sim D_{\text{react}}$	$\mathbb{E}[\Delta t_{\text{react}}]$	s_{min}
Sync	$\Delta t_{\text{infer}} + \Delta t_{\text{exec}}$	$\mathcal{U}(\Delta t_{\text{infer}}, 2 * \Delta t_{\text{infer}} + \Delta t_{\text{exec}})$	$1.5 * \Delta t_{\text{infer}} + 0.5 * \Delta t_{\text{exec}}$	–
Async	Δt_{exec}	$\mathcal{U}(\Delta t_{\text{infer}}, \Delta t_{\text{infer}} + \Delta t_{\text{exec}})$	$\Delta t_{\text{infer}} + 0.5 * \Delta t_{\text{exec}}$	$\lceil \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rceil$

Synchronous Inference. The system operates synchronously by default, as shown in Figure 2(a). After completing execution of the preceding chunk at time t, the client sends the observation $\mathbf{o}_t$ to the server to request a new inference. After the inference latency, the server returns the predicted chunk. During this period, the robot controller pauses and resumes only when the new actions arrive at $t + \Delta t_{\text{infer}}$. To achieve uninterrupted execution, the condition $\Delta t_{\text{infer}} < \Delta t_{\text{ctrl}}$ (*i.e.*, $d = 0$) should hold, meaning the next chunk is available within a single control step. In practice, however, this requirement is hardly satisfied, resulting in non-smooth trajectories and degraded task performance [4, 89].

Asynchronous Inference. A natural strategy to tackle inter-chunk pauses is asynchronous inference [82]. The core idea is to initiate inference of the next chunk before the current chunk is fully executed, as depicted in Figure 2(b). Specifically, once inference is triggered at time t, the robot continues executing the remaining actions in the ongoing chunk. By time $t + \Delta t_{\text{infer}}$, when the final action is completed, the newly predicted chunk is expected to be available, thereby enabling seamless execution without halt.

However, asynchronous execution incurs the problem of perception-execution gap [49, 104]. The observation is captured at time t, but when the new chunk becomes available, the environment and the robot state may have changed due to actions executed during the interval $(t, t + \Delta t_{\text{infer}})$. A naive strategy of discarding the first d delayed actions in the new chunk and switching to the remaining ones $[d, d + s)$ can lead to unstable and discontinuous motion, and this issue becomes increasingly severe as the delay d grows [4, 89]. Recent approaches mitigate inter-chunk discontinuity by incorporating the overlapping actions ($[s, d + s)$ in previous chunk) as part of the model input [4, 5, 94, 58, 9]. This paper follows RTC [4, 5], where the overlapping actions are treated as prefix conditions during action generation, guiding the new actions to transition smoothly.

Smoothness *vs.* Reaction. Existing real-time VLAs primarily focus on improving inter-chunk smoothness. Nevertheless, they often overlook or misunderstand another fundamental aspect of real-time performance: **reaction**. In this work, we revisit the notion of reaction in action chunking policy inference and provide a systematic analysis.

Reaction time Δt_{react} is defined as the interval between the occurrence of a sudden event and the response produced by the robot. We summarize the reaction characteristics of synchronous and asynchronous clients in Table 1. A key insight is that reaction time is susceptible to the dual influence of inference latency and frequency. Given that policy inference is performed periodically, with consecutive inference triggers at time t and t' in Figure 2. When a new event occurs, the system can only respond after the next inference cycle is completed. Therefore, the lower bound of Δt_{react} is Δt_{infer}, corresponding to the case where the event happens just before an inference starts. In the worst case, the event occurs immediately after inference begins; the reaction will then only be reflected at $t' + \Delta t_{\text{infer}}$, shaping an upper bound equal to Δt_{infer} plus the inference interval.

As an event occurs stochastically in the physical world, it is equally likely to occur at any point within an inference interval. The reaction time can then be modeled as *following a uniform distribution*, denoted by D_{react}. An important finding from the expectation $\mathbb{E}[\Delta t_{\text{react}}]$ in Table 1 is that the gain by upgrading from synchronous to asynchronous inference is potentially limited, as it reduces the expected reaction time by only $0.5 * \Delta t_{\text{infer}}$.

Reducing the execution horizon s is an intuitive idea to increase the inference frequency. In the asynchronous setting, s has a minimum value $s_{\text{min}} := \lceil \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rceil$ to guarantee that the inference interval exceeds the latency (*i.e.*, $\Delta t_{\text{exec}} \geq \Delta t_{\text{infer}}$). Under this configuration, inference is triggered every s_{min} control steps, achieving optimal reaction performance [104].

Time to First Action. As observed in Table 1, reaction time heavily depends on inference latency. More importantly, if actions are not generated simultaneously, responsiveness is determined solely by how quickly the system can produce the *first* action. Since the robot does not need the entire action chunk to begin moving, later actions, while essential for task accuracy, do not directly affect immediate

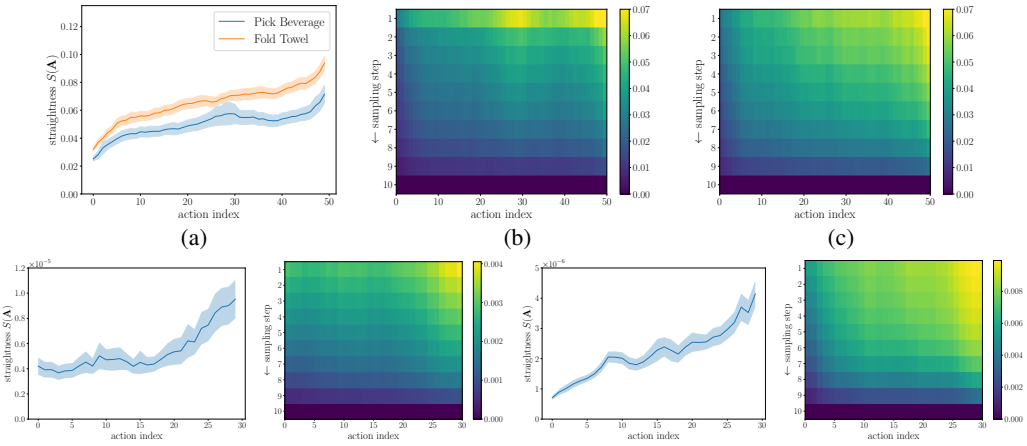


Figure 3: Visualizations of (a) straightness $S(\mathbf{A})$ of the denoising path during sampling of the action chunk, and (b,c) differences between the intermediate clean action estimates $\tilde{\mathbf{A}}_t^{\tau\to 0}$ at each sampling timestep τ and the final output $\mathbf{A}_t^0$, using $\pi_{0.5}$ on Pick Beverage and Fold Towel tasks, respectively. To assess generalizability across models and datasets, we further evaluate X-VLA on LIBERO (d,e) and CALVIN (f,g). Shaded regions in (a,d,f) indicate ± 2 SEM over 200 random samples.

responsiveness. Therefore, we introduce *Time to First Action (TTFA)* as a more precise metric for measuring reactivity, analogous to Time to First Token (TTFT) in large language models [32, 121]. TTFA explicitly captures the earliest moment at which the robot can initiate movement, making it the true bottleneck of reaction speed. This paper presents a novel asynchronous pipeline that jointly minimizes TTFA and increases inference frequency, leading to substantially improved reaction capability in action chunking policies.

3 Methodology

3.1 Preliminaries

We adopt the widely used flow-based VLA structure [3, 35, 2, 120]. The model consists of a VLM backbone and an action expert (AE), learning a velocity field that transports a noise sample to the target action chunk using conditional flow matching [51, 52]. Training follows the optimal transport formulation [91, 56], which assumes a linear interpolation path between Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the ground-truth actions $\hat{\mathbf{A}}_t$: $\mathbf{A}_t^\tau = \tau\boldsymbol{\epsilon} + (1-\tau)\hat{\mathbf{A}}_t$, where $\tau \in (0,1)$ is the continuous timestep of the flow. The objective is to regress the velocity field along this path with network v_θ:

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau\sim\mathcal{U}(0,1)} \left\| v_\theta(\mathbf{o}_t, \mathbf{A}_t^\tau, \tau) - (\boldsymbol{\epsilon} - \hat{\mathbf{A}}_t) \right\|^2. \tag{1}$$

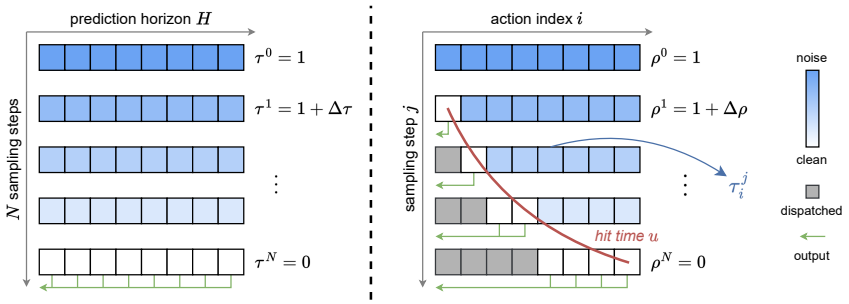
During inference, actions are generated by initializing from Gaussian noise $\mathbf{A}_t^1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ at $\tau = 1$, and progressively integrating the learned velocity field toward $\tau = 0$ using an ODE solver such as the Euler method:

$$\mathbf{A}_t^{\tau+\Delta\tau} = \mathbf{A}_t^\tau + v_\theta(\mathbf{o}_t, \mathbf{A}_t^\tau, \tau)\Delta\tau, \tag{2}$$

where $\Delta\tau = -1/N$ is related to the number of sampling steps N, a typical value of 10 in practice.

3.2 Pilot Study on Action Chunk Sampling

Existing flow-based VLAs treat the entire action chunk as an indivisible unit and apply a constant timestep schedule across all action indexes. As a result, every action within the chunk undergoes the same number of denoising steps during inference. The immediate next action $\mathbf{A}_t$, which is urgently required for execution, is therefore forced to share the same schedule as the most distant future action $\mathbf{A}_{t+H-1}$. Consequently, the entire multi-step denoising procedure has to be completed before any individual action can be issued, which constitutes a dominant bottleneck of the overall latency [109].



(a) Constant Schedule (b) Horizon-Aware Schedule

Figure 4: Illustration of (a) constant timestep schedule used in conventional flow sampling and (b) Horizon-Aware Schedule (HAS) used in FASTER that allocates adaptive hit times across the action chunk and accelerates the sampling of early actions, enabling streaming output.

Nevertheless, action chunks exhibit an inherent temporal structure. Given current observations and proprioceptive states, early-stage actions are subject to stronger causal constraints and thus lie in a substantially narrower search space compared to future actions. Intuitively, this makes short-term predictions easier and more certain. Furthermore, when asynchronous methods incorporate action prefixes as input [5, 58, 89], these extra priors provide additional conditioning that constrains subsequent predictions. This further reduces the uncertainty of the immediate actions and lowers the complexity of generation.

We validate this hypothesis through a quantitative analysis of the sampling dynamics in flow-based VLAs. Specifically, we adopt the *straightness* metric [56], which is defined for any continuously differentiable process $\mathbf{Z} = \{Z_\tau\}$ evolving from Z_0 to Z_1 as $S(\mathbf{Z}) = \int_0^1 \mathbb{E}\left[\|(Z_1 - Z_0) - \dot{Z}_\tau\|^2\right] \mathrm{d}\tau$, where $\dot{Z}_\tau = \frac{\mathrm{d}}{\mathrm{d}\tau} Z_\tau$ denotes the instantaneous velocity at time τ. In our context, the VLA denoising process is discretized and the straightness can be formulated as:

$$S(\mathbf{A}) = \sum_{\tau=0}^{1} \mathbb{E}_t \left[\left\| (\mathbf{A}_t^1 - \mathbf{A}_t^0) - v_\theta(\mathbf{o}_t, \mathbf{A}_t^\tau, \tau) \right\|^2 \right] \cdot (-\Delta\tau), \tag{3}$$

where $\mathbf{A}_t^0$ represents the final actions obtained via Equation (2). A value of $S(\mathbf{A}) = 0$ indicates a perfectly straight path. Smaller $S(\mathbf{A})$ corresponds to paths closer to linear interpolation, which in turn can be accurately integrated with fewer steps [56]. We also investigate the estimated clean actions $\tilde{\mathbf{A}}_t^{\tau \to 0}$ at each denoising step $\{1, 1 + \Delta\tau, \ldots, -\Delta\tau\}$, obtained via the following extrapolation:

$$\tilde{\mathbf{A}}_t^{\tau \to 0} = \mathbf{A}_t^\tau - v_\theta(\mathbf{o}_t, \mathbf{A}_t^\tau, \tau)\tau. \tag{4}$$

We measure their deviation from the final output $\mathbf{A}_t^0$ using ℓ_2 norm $\|\tilde{\mathbf{A}}_t^{\tau \to 0} - \mathbf{A}_t^0\|_2$. During sampling, this deviation is expected to decrease and reaches zero at the final step. A smaller deviation suggests that the model provides a more accurate estimate of the results at current timestep.

We conduct a pilot study by fine-tuning a pretrained $\pi_{0.5}$ model on our real-world robotic tasks. As visualized in Figure 3, we can find that both the straightness metric and the estimate deviation exhibit non-uniformity across the temporal dimension (action index) of the action chunk. In particular, early actions (approximately the first 1–10 frames) demonstrate lower straightness values and smaller variations in $\tilde{\mathbf{A}}_t^{\tau \to 0}$ throughout the sampling iterations. This trend is also observed for X-VLA on two simulation datasets LIBERO and CALVIN, providing strong evidence supporting our hypothesis across models and datasets.

3.3 FASTER

Motivated by the insight that near-term actions within a chunk are easier to generate under flow matching, we propose to prioritize the sampling of these latency-critical actions with a Horizon-Aware Schedule (HAS).

Horizon-Aware Schedule. Unlike conventional flow-based VLAs, which employ a constant time schedule across the entire chunk (Figure 4(a)), we design a horizon-aware time allocation mechanism that accelerates the denoising of near-term actions during inference, while allowing later-horizon actions to follow a comparatively slower schedule, mimicking the straightness plot in Figure 3(a).

Inspired by Diffusion Forcing [10], the flow matching timestep is made index-dependent and represented as a vector $\boldsymbol{\tau} = \{\tau_i\}$, where $i \in [0, H-1]$ denotes the action index. As illustrated in Figure 4(b), we use $\rho^j \in [0, 1]$ to represent the global sampling progress at j-th step, which still iterates from 1 to 0 following the standard flow matching procedure. Each action reaches completion at a distinct *hit time* u_i determined by its index:

$$u_i = (1 - (i/(H-1))^\alpha) \cdot u_0 \quad i \in [1, H-1], \tag{5}$$

where the predefined u_0 specifies the global timestep at which the first action is finalized. The hyperparameter $\alpha \in (0, 1]$ controls how the hit times vary across the action index. When $\alpha = 1$, the hit times decrease uniformly from u_0 to 0. When $\alpha < 1$, early actions reach their hit times rapidly, while future actions have hit times closer to 0. This design adaptively allocates more denoising steps to future actions due to compounding uncertainty, preserving generation quality and minimizing deviation from the original schedule inherited from pretraining.

Given the global timestep ρ^j, the local time schedule for each action index is formulated as:

$$\tau_i^j = \max\left(0, (\rho^j - u_i)/(1 - u_i)\right). \tag{6}$$

Under this schedule, once ρ^j reaches u_0, the first action is fully denoised and can be dispatched immediately, while the remaining actions continue refining. We set $u_0 = (N-1)/N$, thus guaranteeing that the first action is ready with a single sampling step. With ρ^j further decreasing, subsequent actions are progressively completed when $\rho^j \leq u_i$.

Fine-tuning with Mixed Schedule. Directly fine-tuning a pretrained VLA using HAS may introduce two challenges. First, existing pretrained models are optimized under a constant timestep schedule, thus naively switching to an index-dependent schedule can enlarge the fine-tuning gap, in addition to the distribution shift. Second, when randomly selecting $\rho \sim \mathcal{U}(0, 1)$, there exists a high probability that the corresponding local timestep for near-term actions becomes zero. This could collapse the learning of these actions since the inputs are constantly ground-truth and potentially deteriorate rollout performance.

To address these issues, we introduce a mixed scheduling strategy to augment fine-tuning. Concretely, given a mixing probability p, each action sample in a training batch utilizes HAS with probability p, and retains the original constant schedule with probability $1 - p$. The mixed schedule forces the model to learn the flow matching velocity field under both timestep parameterizations, thereby improving robustness to the schedule variation across the action horizon. It is worth noting that the proposed scheduling methodology can be readily incorporated into the standard fine-tuning pipeline of flow-based VLAs, without any architecture modifications or additional training cost.

Synergization with Action Conditioning. HAS naturally synergizes with the action conditioning technique proposed in Training-time RTC [5], which conditions both training and sampling on the action prefixes by treating them as fully denoised. Under our adaptive schedule, early actions frequently receive timesteps close to zero, while future actions are assigned progressively larger timesteps. This pattern is inherently consistent with the principle of action conditioning. Moreover, it encourages the model to learn a more structured mapping between timestep values and the degree of noise interpolation relative to the target actions, thereby strengthening its understanding of temporally conditioned generation. Action conditioning is integrated into HAS with an offset on the index, and we provide a detailed description in Appendix C.1.

Streaming Client-Server Interface. With FASTER, actions are generated progressively during the denoising iterations. To exploit this property in real-world robotic systems, we implement a streaming client-server interface. On the server side, newly finalized actions from the policy are dispatched immediately, while the model proceeds to generate the remaining actions concurrently. On the client side, the controller continuously listens for incoming packets and appends received actions to the robot's action buffer for execution, without waiting for the entire chunk to complete. As long as the action acquisition rate exceeds the robot control frequency—which is achievable even on consumer-level GPUs—the robot can operate without interruption (shown in Appendix C.3).

Enhancing Reaction Capability. We analyze how FASTER improves reaction time by tightening both its lower and upper bounds. For conventional flow-based VLAs with N sampling steps, TTFA is approximately $\Delta t_{\text{VLM}} + N\Delta t_{\text{AE}}$, where Δt_{VLM} and Δt_{AE} denote the inference time of the VLM and AE, respectively. Equipped with FASTER, TTFA is shortened to $\Delta t_{\text{VLM}} + \Delta t_{\text{AE}}$, as the first action requires only a single AE sampling step.

Table 2: Comparison of reaction capability on RTX 4090 and RTX 4060 GPUs. TTFA is measured empirically, whereas expected reaction times are computed under the timing model. "*Speedup*" denotes the relative gain of our method over the asynchronous baseline.

Model	Method	RTX 4090			RTX 4060		
		TTFA ↓	$s_{\min}$ ↓	$\mathbb{E}[\Delta t_{\text{react}}]$ ↓	TTFA ↓	$s_{\min}$ ↓	$\mathbb{E}[\Delta t_{\text{react}}]$ ↓
$\pi_{0.5}$	Sync	$80.0_{\pm 1.6}$ms	3	170.0ms	$303.3_{\pm 0.8}$ms	10	621.6ms
	Async	$80.0_{\pm 1.6}$ms	3	130.0ms	$303.3_{\pm 0.8}$ms	10	470.0ms
	FASTER	**62.1**$_{\pm 3.1}$ms	3	**112.1**ms	**238.6**$_{\pm 1.9}$ms	**8**	**371.9**ms
	Speedup	**1.29**×	**–**	**1.16**×	**1.27**×	**1.25**×	**1.26**×
X-VLA	Sync	$113.7_{\pm 0.8}$ms	4	237.2ms	$399.5_{\pm 8.5}$ms	12	799.2ms
	Async	$113.7_{\pm 0.8}$ms	4	180.4ms	$399.5_{\pm 8.5}$ms	12	599.5ms
	FASTER	**44.8**$_{\pm 0.3}$ms	**2**	**78.1**ms	**129.2**$_{\pm 2.4}$ms	**6**	**229.2**ms
	Speedup	**2.54**×	**2**×	**2.31**×	**3.09**×	**2**×	**2.62**×

Responsiveness can be further improved by increasing the inference frequency, *i.e.*, reducing the execution horizon s, whose minimum value $s_{\min}$ must satisfy $\Delta t_{\text{exec}} \geq \Delta t_{\text{infer}}$. When a high frequency is desired, a relatively small s is selected (*e.g.* 4 out of $H = 50$), implying that only a small portion of the chunk is useful while the other actions are discarded. Our method enables an early-stopping strategy: once all actions within the execution horizon are finalized, the remaining sampling steps can be skipped. This avoids completing all AE iterations and effectively reduces the overall latency. Consequently, we can apply a smaller feasible $s_{\min}$ compared with conventional asynchronous inference, tightening the upper bound of Δt_{react} without sacrificing execution smoothness.

4 Experiments

We focus primarily on real-world experiments, while simulation benchmarks and ablation studies are provided in Appendix E.4 and E.5 due to space limitations.

4.1 Experimental Analysis on Reaction Speed

Experimental Setup. To investigate improvements in reaction capability, we first compare the inference latency—measured by TTFA—and the expected reaction time of our method against the synchronous and asynchronous baselines. Since prior real-time VLA approaches [4, 5, 89] follow the same naive asynchronous paradigm in these aspects, we do not report them separately. Experiments are conducted on two hardware platforms: a high-performance RTX 4090 GPU, and a consumer-grade RTX 4060 GPU. We utilize two representative flow-based VLAs, $\pi_{0.5}$ [35] and X-VLA [120], and adhere to their default configurations. The robot control frequency is set to $f = 30$Hz, corresponding to a control period $\Delta t_{\text{ctrl}} = 33.3$ms. To reflect the optimal achievable reaction speed, we set the execution horizon to $s_{\min}$, thereby maximizing the inference frequency.

Results. As detailed in Table 2, our method manifests substantial acceleration in reaction performance across all scenarios. As the design of X-VLA incurs a higher computational cost in its action expert, FASTER delivers particularly significant gains, achieving a 3× boost in TTFA on RTX 4060. Notably, the early-stopping strategy in our approach effectively decreases the feasible inference interval by a smaller $s_{\min}$, which minimizes the inference-execution cycle and contributes to additional enhancements in responsiveness.

As highlighted in Section 2, reaction time should be treated as a random variable, since event occurrences are inherently stochastic. Accordingly, Table 3 presents a probabilistic analysis that more faithfully reflects real-world conditions, measuring the probability that one method attains a faster reaction time than another. FASTER surpasses both baselines by a clear margin, with a larger advantage in the more resource-constrained scenario. Notably, on X-VLA, our method is deterministically superior: its upper bound of reaction time is lower than the baselines' lower bound, establishing a strict performance dominance.

4.2 Real-world Experiments

Experimental Setup. To examine the real-robot reaction capability in a highly dynamic environment, we design a task that demands both rapid response and accurate motion execution. Specifically, we

Table 3: Pairwise comparison of reaction speed from a probabilistic perspective on RTX 4090 and RTX 4060 GPUs. "*vs.* Sync" denotes the probability that an independently sampled reaction time from the given method is shorter than one from Sync under the timing model, and "*vs.* Async" is defined analogously.

Model	Method	RTX 4090		RTX 4060	
		vs. Sync	*vs.* Async	*vs.* Sync	*vs.* Async
$\pi_{0.5}$	Async	0.72	-	0.74	-
	FASTER	**0.81**	**0.66**	**0.88**	**0.77**
X-VLA	Async	0.73	-	0.75	-
	FASTER	**1.00**	**1.00**	**1.00**	**1.00**

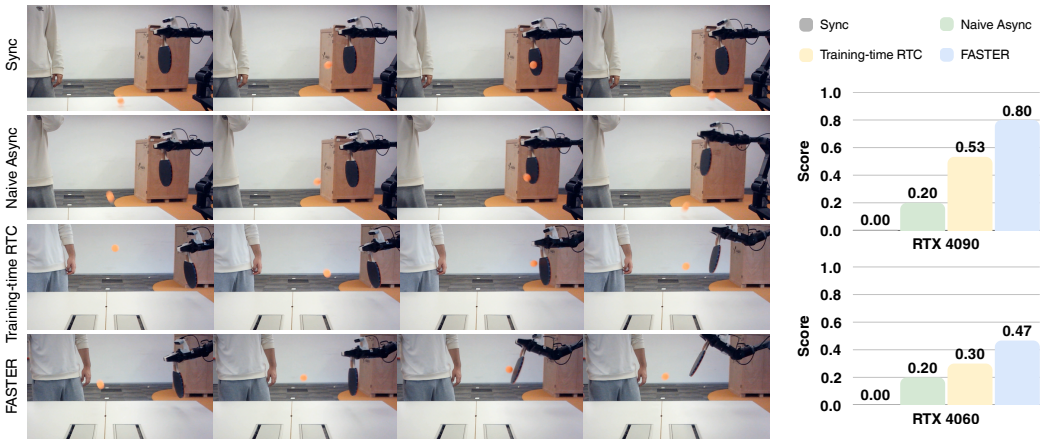


Figure 5: Comparison of real-world reaction speed on the table tennis task. **Left**: Visualization of rollouts on RTX 4090, the third column corresponds to the contact moment, and the interval between each image in a row is 166.7ms (5 frames). **Right**: Quantitative completion scores on two GPUs.

train VLA to play table tennis using a racket mounted on a 6-DoF Piper robot arm in the AgileX Cobot Magic platform. We collect approximately 14 minutes of demonstration data via human teleoperation to fine-tune the $\pi_{0.5}$ models. In addition, we include two tasks that place less emphasis on real-time reaction: (1) "Pick Beverage": picking up a beverage and placing it in a basket, focusing on object and localization generalization; (2) "Fold Towel": folding a towel twice with dual arms, representing deformable object manipulation. These tasks assess whether accelerated sampling preserves manipulation performance beyond latency-sensitive settings.

We evaluate Table Tennis, Pick Beverage, and Fold Towel over 15, 35, and 10 trials per method, respectively. For Table Tennis, a miss receives 0 points, while a return landing before or beyond the marked table midpoint receives 0.5 or 1 point. Pick Beverage is scored on grasping and placement, while Fold Towel is scored on four stages: initial grasping, forward folding, regrasping, and backward folding. Each stage receives 0, 0.5, or 1 point according to completion quality, and scores are averaged across stages and trials. Details are provided in Appendix D.1.

Results. We compare synchronous inference ("Sync"), naive asynchronous inference ("Naive Async"), and the state-of-the-art Training-time RTC [5] with our FASTER. Representative rollouts are visualized on the left of Figure 5, with quantitative results presented in the right panels. We observe that Sync fails to respond to incoming balls in our trials due to its drastically slow reaction speed. Naive Async and Training-time RTC share the same temporal pipeline and thus similar reaction capability. However, Training-time RTC improves the smoothness of racket swing through its action conditioning mechanism, leading to moderately higher scores.

Our method exhibits noticeably faster reaction than all baselines, which can be intuitively observed from the racket angle at the moment of ball contact. To successfully return the ball, the robot should begin adjusting its arm posture in advance to reach an appropriate hitting position with sufficient swing velocity. If the reaction is delayed, there is insufficient time left to move, leading to suboptimal contact angles, as seen in the baselines. In contrast, faster reactions allow the robot to initiate motion earlier, providing adequate time to rotate the racket and build up swing speed, resulting in a more powerful and controlled hit. Therefore, the racket angle serves as a clear and visually interpretable

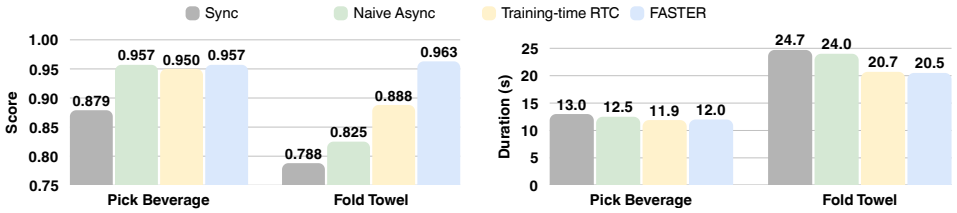


Figure 6: Comparison of real-world performance and completion duration on two additional tasks.

indicator of reaction speed. Additional behavioral timing measurements in Table 10 show a mean release-to-motion interval of 683ms for FASTER, compared with 855ms for Training-time RTC and 916ms for Naive Async. Overall, FASTER achieves the best performance across both hardware settings. The advantage is particularly pronounced on RTX 4060, where inference runs at merely 3Hz, highlighting the compounded benefits of reduced inference latency and increased frequency. More analysis is provided in Appendix E.2.

We report the average completion score and duration for the additional tasks in Figure 6. The results show that asynchronous methods outperform Sync by a clear gap, and FASTER achieves better or comparable scores across both tasks. This highlights that task performance is not determined solely by action accuracy, but also by the real-time interaction with the physical world. Though our method may slightly compromise action prediction due to accelerated sampling, it strikes a more effective balance between responsiveness and accuracy. Consistent with previous findings [5, 89], the synchronous method exhibits the longest task duration due to frequent inter-chunk pauses. In contrast, Training-time RTC and FASTER effectively reduce completion time, yielding greater efficiency for downstream applications.

5 Related Work

Efficient and Real-Time VLAs. Existing approaches improve VLA inference efficiency through smaller backbones [120, 100], model compression [117, 109], and visual token pruning [111, 60]. Asynchronous execution removes inter-chunk pauses [94, 89], while RTC [4] and Training-time RTC [5] improve trajectory continuity through action conditioning. FASTER complements these approaches by reducing the sampling steps before the first executable action becomes available and enabling more frequent replanning through early stopping.

Diffusion Acceleration in VLAs. Diffusion distillation [55, 63] and few-step generation methods [13, 81] reduce sampling steps through additional training stages or modified inputs and objectives. FASTER retains the standard flow-matching objective and pretrained architecture, allocating fewer steps to immediate actions and progressively more refinement to later ones. Streaming diffusion policies [20, 15, 31] also produce immediate actions in one step, but their per-step observation updates would require repeated VLM processing when directly applied to VLAs. FASTER instead uses one VLM prefill per chunk and streams actions generated from the cached representation.

6 Conclusion

In this paper, we revisit the responsiveness of action-chunking VLA policies and identify the constant timestep schedule in flow-based VLAs as a key bottleneck for real-time control. To address this limitation, we propose FASTER, which employs a Horizon-Aware Schedule to adaptively accelerate action sampling, enabling single-step generation of the immediate action while preserving competitive manipulation performance. Combined with a streaming client-server interface and early stopping, FASTER substantially reduces time to first action (TTFA) and enables more responsive closed-loop control. Real-robot experiments confirm that FASTER offers a robust, general, and plug-and-play path toward real-time embodied intelligence, particularly on resource-constrained devices.

Despite these advantages, several limitations remain. Accelerated sampling introduces a trade-off between action fidelity and responsiveness, which may affect performance in precision-sensitive bottlenecks. Moreover, the benefits may be less pronounced when latency is not a primary constraint. Future work could explore uncertainty- or task-phase-aware schedules and more controlled evaluations to better characterize the accuracy-responsiveness trade-off.

Acknowledgments and Disclosure of Funding

This work is supported by the National Natural Science Foundation of China (No. 62422606, 62441615), Hong Kong Research Grant Council General Research Fund (No. 17213925), and Shenzhen Loop Area Institute Fund (No. FPF10120250006). There is no competing interest.

References

[1] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[2] 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.

[3] 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. In *RSS*, 2025.

[4] Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *NeurIPS*, 2025.

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

[6] Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Xindong He, Xu Huang, et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. In *IROS*, 2025.

[7] Paweł Budzianowski, Wesley Maa, Matthew Freed, Jingxiang Mo, Winston Hsiao, Aaron Xie, Tomasz Młoduchowski, Viraj Tipnis, and Benjamin Bolte. Edgevla: Efficient vision-language-action models. *arXiv preprint arXiv:2507.14049*, 2025.

[8] Remi Cadene, Simon Alibert, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, Mustafa Shukor, Jess Moss, Alexander Soare, Dana Aubakirova, Quentin Lhoest, Quentin Gallouédec, and Thomas Wolf. Lerobot: An open-source library for end-to-end robot learning. In *ICLR*, 2026.

[9] Rui Cai, Jun Guo, Xinze He, Piaopiao Jin, Jie Li, Bingxuan Lin, Futeng Liu, Wei Liu, Fei Ma, Kun Ma, et al. Xiaomi-robotics-0: An open-sourced vision-language-action model with real-time execution. *arXiv preprint arXiv:2602.12684*, 2026.

[10] 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.

[11] 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 *ICML*, 2025.

[12] Jiahong Chen, Jing Wang, Long Chen, Chuwei Cai, and Jinghui Lu. Nanovla: Routing decoupled vision-language understanding for nano-sized generalist robotic policies. *arXiv preprint arXiv:2510.25122*, 2025.

[13] Yang Chen, Xiaoguang Ma, and Bin Zhao. Mean-flow based one-step vision-language-action. *arXiv preprint arXiv:2603.01469*, 2026.

[14] Yuxuan Chen, Yixin Han, Yize Huang, and Xiao Li. Rlrc: Reinforcement learning-based recovery for compressed vision-language-action models. *IEEE Robotics and Automation Letters*, 2026.

[15] Zhuoqun Chen, Xiu Yuan, Tongzhou Mu, and Hao Su. Responsive noise-relaying diffusion policy: Responsive and efficient visuomotor control. *TMLR*, 2025.

[16] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor policy learning via action diffusion. In *RSS*, 2023.

[17] Yuntao Dai, Hang Gu, Teng Wang, Qianyu Cheng, Yifei Zheng, Zhiyong Qiu, Lei Gong, Wenqi Lou, and Xuehai Zhou. Actionflow: A pipelined action acceleration for vision language models on edge. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference*, 2026.

[18] Haoran Ding, Noémie Jaquier, Jan Peters, and Leonel Rozo. Fast and robust visuomotor riemannian flow matching policy. *IEEE Transactions on robotics*, 2025.

[19] Zhenchen Dong, Jinna Fu, Jiaming Wu, Shengyuan Yu, Fulin Chen, and Yide Liu. Hybridflow: A two-step generative policy for robotic manipulation. *arXiv preprint arXiv:2602.13718*, 2026.

[20] Yufei Duan, Hang Yin, and Danica Kragic. Real-time iteration scheme for diffusion policy. In *IROS*, pages 11758–11764, 2025.

[21] Xianzhe Fan, Shengliang Deng, Xiaoyang Wu, Yuxiang Lu, Zhuoling Li, Mi Yan, Yujia Zhang, Zhizheng Zhang, He Wang, and Hengshuang Zhao. Any3d-vla: Enhancing vla robustness via diverse point clouds. In *ICML*, 2026.

[22] Hengyu Fang, Yijiang Liu, Yuan Du, Li Du, and Huanrui Yang. Sqap-vla: A synergistic quantization-aware pruning framework for high-performance vision-language-action models. *arXiv preprint arXiv:2509.09090*, 2025.

[23] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *ICLR*, 2025.

[24] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile ALOHA: Learning bimanual mobile manipulation using low-cost whole-body teleoperation. In *CoRL*, 2024.

[25] Juntao Gao, Feiyang Ye, Jing Zhang, and Wenjing Qian. Compressor-vla: Instruction-guided visual token compression for efficient robotic manipulation. *arXiv preprint arXiv:2511.18950*, 2025.

[26] Yuxuan Gao, Yedong Shen, Shiqi Zhang, Wenhao Yu, Yifan Duan, Jiajia Wu, Jiajun Deng, Yanyong Zhang, et al. Drift-based policy optimization: Native one-step policy learning for online robot control. In *ACM MM*, 2026.

[27] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *NeurIPS*, 2025.

[28] Weifan Guan, Qinghao Hu, Aosheng Li, and Jian Cheng. Efficient vision-language-action models for embodied manipulation: A systematic survey. *arXiv preprint arXiv:2510.17111*, 2025.

[29] Xiaoshuai Hao, Lei Zhou, Zhijian Huang, Zhiwen Hou, Yingbo Tang, Lingfeng Zhang, Guang Li, Zheng Lu, Shuhuai Ren, Xianhui Meng, et al. Mimo-embodied: X-embodied foundation model technical report. *arXiv preprint arXiv:2511.16518*, 2025.

[30] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[31] Sigmund H Høeg, Yilun Du, and Olav Egeland. Fast policy synthesis with variable noise diffusion models. In *ICRA*, 2025.

[32] Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, et al. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.

[33] Yuting Huang, Leilei Ding, Zhipeng Tang, Zenghuan Zhu, Jiajun Deng, Xinrui Lin, Shuo Liu, Haojie Ren, Jianmin Ji, and Yanyong Zhang. Environment-aware adaptive pruning with interleaved inference orchestration for vision-language-action models. *arXiv preprint arXiv:2602.00780*, 2026.

[34] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

[35] 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.

[36] Jason Jabbour, Dong-Ki Kim, Max Smith, Jay Patrikar, Radhika Ghosal, Youhui Wang, Ali Agha, Vijay Janapa Reddi, and Shayegan Omidshafiei. Don't run with scissors: Pruning breaks vla models but they can be recovered. *arXiv preprint arXiv:2510.08464*, 2025.

[37] Boseong Jeon, Yunho Choi, and Taehan Kim. Shallow-π: Knowledge distillation for flow-based vlas. *arXiv preprint arXiv:2601.20262*, 2026.

[38] Bofang Jia, Pengxiang Ding, Can Cui, Mingyang Sun, Pengfang Qian, Siteng Huang, Zhaoxin Fan, and Donglin Wang. Score and distribution matching policy: Advanced accelerated visuomotor policies via matched distillation. *arXiv preprint arXiv:2412.09265*, 2024.

[39] Jindou Jia, Gen Li, Xiangyu Chen, Tuo An, Yuxuan Hu, Jingliang Li, Xinying Guo, and Jianfei Yang. Action-to-action flow matching. *arXiv preprint arXiv:2602.07322*, 2026.

[40] Titong Jiang, Xuefeng Jiang, Yuan Ma, Xin Wen, Bailin Li, Kun Zhan, Peng Jia, Yahui Liu, Sheng Sun, and Xianpeng Lang. The better you learn, the smarter you prune: Towards efficient vision-language-action models via differentiable token pruning. *arXiv preprint arXiv:2509.12594*, 2025.

[41] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *RSS*, 2024.

[42] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *RSS*, 2025.

[43] 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. In *CoRL*, 2024.

[44] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. LLaVA-onevision: Easy visual task transfer. *Transactions on Machine Learning Research*, 2025.

[45] Shaolong Li, Lichao Sun, and Yongchao Chen. One-step flow policy: Self-distillation for fast visuomotor policies. *arXiv preprint arXiv:2603.12480*, 2026.

[46] Wei Li, Renshan Zhang, Rui Shao, Zhijian Fang, Kaiwen Zhou, Zhuotao Tian, and Liqiang Nie. Semanticvla: Semantic-aligned sparsification and enhancement for efficient robotic manipulation. In *AAAI*, 2026.

[47] Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shu-Tao Xia, Zhi Wang, and Wenwu Zhu. Sp-vla: A joint model scheduling and token pruning approach for vla model acceleration. In *ICLR*, 2026.

[48] Zhuoling Li, Xiaoyang Wu, Zhenhua Xu, and Hengshuang Zhao. Train once, deploy anywhere: Realize data-efficient dynamic object manipulation. *arXiv preprint arXiv:2508.14042*, 2025.

[49] Aileen Liao, Dong-Ki Kim, Max Olan Smith, Ali-akbar Agha-mohammadi, and Shayegan Omidshafiei. Delay-aware diffusion policy: Bridging the observation-execution gap in dynamic tasks. *arXiv preprint arXiv:2512.07697*, 2025.

[50] Tao Lin, Yilei Zhong, Yuxin Du, Jingjing Zhang, Jiting Liu, Yinxinyu Chen, Encheng Gu, Ziyan Liu, Hongyi Cai, Yanwen Zou, et al. Evo-1: Lightweight vision-language-action model with preserved semantic alignment. *arXiv preprint arXiv:2511.04555*, 2025.

[51] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*, 2023.

[52] Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky TQ Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

[53] 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*, pages 44776–44791, 2023.

[54] 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*, pages 40085–40110, 2024.

[55] Songming Liu, Bangguo Li, Kai Ma, Lingxuan Wu, Hengkai Tan, Xiao Ouyang, Hang Su, and Jun Zhu. RDT2: Exploring the scaling limit of umi data towards zero-shot cross-embodiment generalization. *arXiv preprint arXiv:2602.03310*, 2026.

[56] Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.

[57] Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Max Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via guided test-time sampling. In *ICLR*, 2025.

[58] Yufeng Liu, Hang Yu, Juntu Zhao, Bocheng Li, Di Zhang, Mingzhu Li, Wenxuan Wu, Yingdong Hu, Junyuan Xie, Junliang Guo, Dequan Wang, and Yang Gao. Learning Native Continuation for Action Chunking Flow Policies. In *Proceedings of Robotics: Science and Systems*, 2026.

[59] Zhe Liu, Runhui Huang, Rui Yang, Siming Yan, Zining Wang, Lu Hou, Di Lin, Xiang Bai, and Hengshuang Zhao. Drivepi: Spatial-aware 4d mllm for unified autonomous driving understanding, perception, prediction and planning. In *CVPR*, 2026.

[60] Ziyan Liu, Yeqiu Chen, Hongyi Cai, Tao Lin, Shuo Yang, Zheng Liu, and Bo Zhao. Vla-pruner: Temporal-aware dual-level visual token pruning for efficient vision-language-action inference. *arXiv preprint arXiv:2511.16449*, 2025.

[61] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, 22(4):730–751, 2025.

[62] Guanxing Lu, Zifeng Gao, Tianxing Chen, Wenxun Dai, Ziwei Wang, Wenbo Ding, and Yansong Tang. Manicm: Real-time 3d diffusion policy via consistency model for robotic manipulation. *arXiv preprint arXiv:2406.01586*, 2024.

[63] Wuyang Luan, Junhui Li, Weiguang Zhao, Wenjian Zhang, Tieru Wu, and Rui Ma. Snapflow: One-step action generation for flow-matching vlas via progressive self-distillation. *arXiv preprint arXiv:2604.05656*, 2026.

[64] Xiaoyu Ma, Zhengqing Yuan, Zheyuan Zhang, Kaiwen Shi, Lichao Sun, and Yanfang Ye. Blurr: A boosted low-resource inference for vision-language-action model. In *Companion Proceedings of the ACM Web Conference*, 2026.

[65] Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision–language–action models for embodied ai. *IEEE Transactions on Neural Networks and Learning Systems*, 2026.

[66] Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.

[67] Michael Matthews, Michael Beukman, Chris Lu, and Jakob Nicolaus Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. In *ICLR*, 2025.

[68] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *RAL*, 2022.

[69] Chaojun Ni, Cheng Chen, Xiaofeng Wang, Zheng Zhu, Wenzhao Zheng, Boyuan Wang, Tianrun Chen, Guosheng Zhao, Haoyun Li, Zhehao Dong, et al. Swiftvla: Unlocking spatiotemporal dynamics for lightweight vla models at minimal overhead. *arXiv preprint arXiv:2512.00903*, 2025.

[70] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. In *ICRA*, 2024.

[71] Seongmin Park, Hyungmin Kim, Wonseok Jeon, Juyoung Yang, Byeongwook Jeon, Yoonseon Oh, and Jungwook Choi. Quantization-aware imitation-learning for resource-efficient robotic control. *arXiv preprint arXiv:2412.01034*, 2024.

[72] Seongmin Park, Hyungmin Kim, Sangwoo Kim, Wonseok Jeon, Juyoung Yang, Byeongwook Jeon, Yoonseon Oh, and Jungwook Choi. Saliency-aware quantized imitation learning for efficient robotic control. In *ICCV*, pages 13140–13150, 2025.

[73] Xiaohuan Pei, Yuxing Chen, Siyu Xu, Yunke Wang, Yuheng Shi, and Chang Xu. Action-aware dynamic pruning for efficient vision-language-action manipulation. In *ICLR*, 2026.

[74] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

[75] Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-action flow policies. In *CoRL*, 2025.

[76] Tommoro Robotics, Jesoon Kang, Taegeon Park, Jisu An, Soo Min Kimm, Jaejoon Kim, Jinu Pahk, Byungju Kim, Junseok Lee, Namheon Baek, et al. Habilis-$beta$: A fast-motion and long-lasting on-device vision-language-action model. *arXiv preprint arXiv:2602.18813*, 2026.

[77] Ranjan Sapkota, Yang Cao, Konstantinos I Roumeliotis, and Manoj Karkee. Vision-language-action (vla) models: Concepts, progress, applications and challenges. *arXiv preprint arXiv:2505.04769*, 2025.

[78] Rui Shao, Wei Li, Lingsen Zhang, Renshan Zhang, Zhiyang Liu, Ran Chen, and Liqiang Nie. Large vlm-based vision-language-action models for robotic manipulation: A survey. *arXiv preprint arXiv:2508.13073*, 2025.

[79] Juyi Sheng, Ziyi Wang, Peiming Li, and Mengyuan Liu. Mp1: Meanflow tames policy learning in 1-step for robotic manipulation. In *AAAI*, 2026.

[80] Modi Shi, Li Chen, Jin Chen, Yuxiang Lu, Chiming Liu, Guanghui Ren, Ping Luo, Di Huang, Maoqing Yao, and Hongyang Li. Is diversity all you need for scalable robotic manipulation? *arXiv preprint arXiv:2507.06219*, 2025.

[81] 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.

[82] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[83] Andreas Sochopoulos, Nikolay Malkin, Nikolaos Tsagkas, Joao Moura, Michael Gienger, and Sethu Vijayakumar. Fast flow-based visuomotor policies via conditional optimal transport couplings. In *CoRL*, 2025.

[84] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

[85] Wenxuan Song, Jiayi Chen, Pengxiang Ding, Yuxin Huang, Han Zhao, Donglin Wang, and Haoang Li. Ceed-vla: Consistency vision-language-action model with early-exit decoding. In *ECCV*, 2026.

[86] Wenxuan Song, Jiayi Chen, Pengxiang Ding, Han Zhao, Wei Zhao, Zhide Zhong, Zongyuan Ge, Jun Ma, and Haoang Li. Accelerating vision-language-action model integrated with action chunking via parallel decoding. In *IROS*, 2025.

[87] Mingzhen Sun, Weining Wang, Gen Li, Jiawei Liu, Jiahui Sun, Wanquan Feng, Shanshan Lao, SiYu Zhou, Qian He, and Jing Liu. Ar-diffusion: Asynchronous video generation with auto-regressive diffusion. In *CVPR*, pages 7364–7373, 2025.

[88] Amir Taherin, Juyi Lin, Arash Akbari, Arman Akbari, Pu Zhao, Weiwei Chen, David Kaeli, and Yanzhi Wang. Cross-platform scaling of vision-language-action models from edge to cloud gpus. In *Proceedings of the Great Lakes Symposium on VLSI 2026*, 2026.

[89] 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.

[90] Gemini Robotics Team, Abbas Abdolmaleki, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Ashwin Balakrishna, Nathan Batchelor, Alex Bewley, Jeff Bingham, et al. Gemini Robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer. *arXiv preprint arXiv:2510.03342*, 2025.

[91] Alexander Tong, Kilian FATRAS, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *TMLR*, 2024.

[92] Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. BridgeData v2: A dataset for robot learning at scale. In *CoRL*, 2023.

[93] Hanzhen Wang, Jiaming Xu, Jiayi Pan, Yongkang Zhou, and Guohao Dai. Specprune-vla: Accelerating vision-language-action models via action-aware self-speculative pruning. In *ICML*, 2026.

[94] Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Rao Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. In *ICLR*, 2026.

[95] Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. Bitvla: 1-bit vision-language-action models for robotics manipulation. *arXiv preprint arXiv:2506.07530*, 2025.

[96] Songsheng Wang, Rucheng Yu, Zhihang Yuan, Chao Yu, Feng Gao, Yu Wang, and Derek F Wong. Spec-vla: speculative decoding for vision-language-action models with relaxed acceptance. In *EMNLP*, pages 26916–26928, 2025.

[97] Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, Zirui Ge, Xinyang Tong, Wenxuan Song, Han Zhao, Wei Zhao, Pengxu Hou, et al. Vla-adapter: An effective paradigm for tiny-scale vision-language-action model. In *AAAI*, 2026.

[98] 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 *ICML*, 2025.

[99] Yujie Wei, Jiahan Fan, Jiyu Guo, Ruichen Zhen, Rui Shao, Xiu Su, Zeke Xie, and Shuo Yang. Learning to accelerate vision-language-action models through adaptive visual token caching. *arXiv preprint arXiv:2602.00686*, 2026.

[100] 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. *RAL*, 2025.

[101] Justin Williams, Kishor Datta Gupta, Roy George, and Mrinmoy Sarkar. Lite vla: Efficient vision-language-action control on cpu-bound edge robots. *arXiv preprint arXiv:2511.05642*, 2025.

[102] Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, et al. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

[103] Yiming Wu, Huan Wang, Zhenghao Chen, Jianxin Pang, and Dong Xu. On-device diffusion transformer policy for efficient robot manipulation. In *ICCV*, pages 14073–14083, 2025.

[104] Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.

[105] Zheng Xiong, Kang Li, Zilin Wang, Matthew Jackson, Jakob Foerster, and Shimon Whiteson. Hypervla: Efficient inference in vision-language-action models via hypernetworks. *arXiv preprint arXiv:2510.04898*, 2025.

[106] Chongyang Xu, Yixian Zou, Ziliang Feng, Fanman Meng, and Shuaicheng Liu. Ada3drift: Adaptive training-time drifting for one-step 3d visuomotor robotic manipulation. *arXiv preprint arXiv:2603.11984*, 2026.

[107] Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Towards efficient vision-language-action model via adaptive token caching in robotic manipulation. In *NeurIPS*, 2025.

[108] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[109] 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.

[110] Wencheng Ye, Tianshi Wang, Lei Zhu, Fengling Li, and Guoli Yang. Actdistill: General action-guided self-derived distillation for efficient vision-language-action models. *arXiv preprint arXiv:2511.18082*, 2025.

[111] Yifan Ye, Jiaqi Ma, Jun Cen, and Zhihe Lu. Token expand-merge: Training-free token compression for vision-language-action models. *arXiv preprint arXiv:2512.09927*, 2025.

[112] Wenda Yu, Tianshi Wang, Fengling Li, Jingjing Li, and Lei Zhu. Ac^2-vla: Action-context-aware adaptive computation in vision-language-action models for efficient robotic manipulation. In *IJCAI*, 2026.

[113] Zhaoshu Yu, Bo Wang, Pengpeng Zeng, Haonan Zhang, Ji Zhang, Zheng Wang, Lianli Gao, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on efficient vision-language-action models. *IEEE TPAMI*, 2026.

[114] 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.

[115] Dapeng Zhang, Jing Sun, Chenghui Hu, Xiaoyan Wu, Zhenlong Yuan, Rui Zhou, Fei Shen, and Qingguo Zhou. Pure vision language action (vla) models: A comprehensive survey. *arXiv preprint arXiv:2509.19012*, 2025.

[116] Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In *AAAI*, pages 14754–14762, 2025.

[117] Rongyu Zhang, Menghang Dong, Yuan Zhang, Liang Heng, Xiaowei Chi, Gaole Dai, Li Du, 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.

[118] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *RSS*, 2023.

[119] Yongsheng Zhao, Lei Zhao, Baoping Cheng, Gongxin Yao, Xuanzhang Wen, and Han Gao. Vla-rail: A real-time asynchronous inference linker for vla models and robots. *arXiv preprint arXiv:2512.24673*, 2025.

[120] Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *ICLR*, 2026.

[121] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.

[122] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*, pages 2165–2183, 2023.

Appendix

We provide the following contents in the appendix:

- Section A reviews related work on VLA models, real-time VLAs, diffusion acceleration approaches in VLAs, and discusses closely related directions.
- Section B provides a fine-grained analysis of the asynchronous inference pipeline and clarifies the relationship between inference latency, delay, and minimal execution horizon.
- Section C gives additional methodological details, including the integration with action conditioning, the streaming client-server interface, and pseudo-code for training and inference.
- Section D summarizes implementation details for real-world experiments, simulation benchmarks, policy training, and robot deployment.
- Section E reports additional experimental results, including reaction analysis, extra real-world results, simulation benchmark results, and ablation studies.
- Section F discusses the limitations of this work and potential directions for future research.
- Section G discusses potential positive and negative societal impacts.

A Related Work

Vision-Language-Action Models. Vision-Language-Action (VLA) models [43, 3, 2, 59, 29] extend large-scale vision-language pretraining from Vision-Language Models (VLMs) [1, 108, 44] to embodied action learning, and have demonstrated impressive performance in robotic manipulation. By pretraining on large-scale vision-text-action corpora [6, 70, 92, 41], VLAs enable robots to map multimodal observations and language instructions directly to low-level motor commands, facilitating dexterous manipulation across a wide range of tasks and promoting generalization to diverse and complicated environments [80, 35, 34, 21, 90].

Early approaches such as RT-2 [122] and OpenVLA [43] discretize robot actions into tokens, making them compatible with the auto-regressive objective of VLMs. Subsequent effort explores diffusion- or flow-matching-based action generation [16], adopting continuous action representations to model the multimodal distribution. Represented by methods including π_0 [3] and GR00T [2], these approaches incorporate a dedicated action expert alongside the VLM backbone, generating high-quality actions conditioned on vision-language features.

Real-Time VLAs. In contrast to VLMs operating purely in cyberspace, VLAs interact with the physical world and are therefore highly sensitive to real-time interaction [48, 104]. Consequently, improving the efficiency of VLAs has become an active research topic [113, 28]. A straightforward strategy is to reduce model inference latency, with representative directions including:

- Replacing the VLM backbone (typically $3 \sim 7$B parameters) with smaller models consisting of fewer than 1B parameters [120, 100, 69, 50, 12, 105, 75, 82, 54, 97, 7];
- Compressing the LLM backbone through mechanisms such as layer selection and early exiting [117, 109, 82, 14, 114, 37, 110, 36, 85, 2, 112];
- Accelerating action decoding, mainly for auto-regressive VLAs [96, 85, 42, 86, 74];
- Pruning visual tokens, as multi-view image inputs account for a large proportion of tokens while often introducing perceptual redundancy [111, 60, 25, 46, 73, 22, 93, 40, 47, 107, 33, 99, 112];
- Applying low-level inference optimizations or quantization techniques [17, 64, 101, 66, 88, 22, 103, 14, 95, 72, 71].

Another line of work seeks to eliminate inter-chunk pauses introduced by standard action chunking and synchronous inference paradigm. By introducing asynchronous execution, VLA models can generate the next action chunk concurrently while the current one is being executed, resulting in non-stop trajectories [82, 119, 104, 9, 58]. However, naively switching between chunks may cause abrupt multimodal transitions and jerky motions, a phenomenon known as inter-chunk discontinuity [4, 49]. To mitigate this issue, RTC [4] inpaints the next action chunk conditioned on the current chunk, while Training-time RTC [5], REMAC [94], and VLASH [89] condition the model on the predicted actions.

Different from prior work, FASTER is the first real-time VLA that explicitly targets responsiveness by accelerating the sampling of immediate actions. Notably, it requires no architectural modifications, rendering it orthogonal and complementary to aforementioned efficient VLA techniques, such as LLM compression and token pruning.

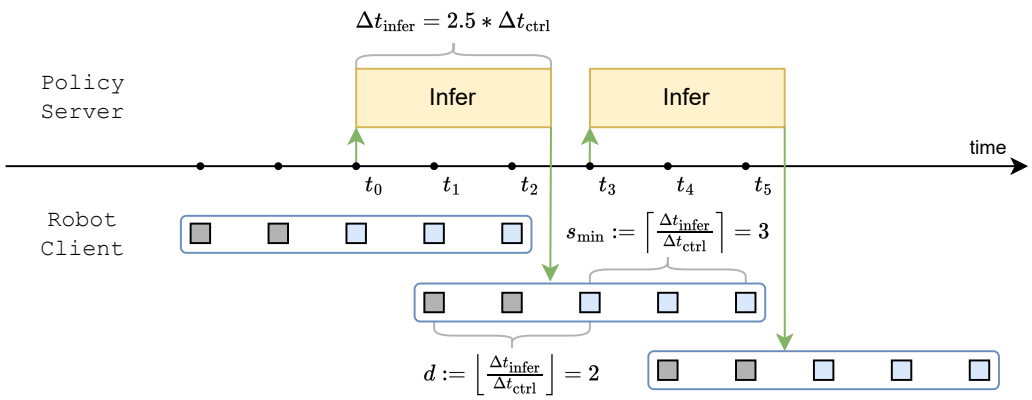


Figure 7: Temporal pipeline of asynchronous inference at a fine-grained level. Suppose the inference latency Δt_{infer} is 2.5 times the controller period Δt_{ctrl}, resulting in an inference delay of $d = 2$ and a minimal execution horizon of $s_{\min} = 3$.

Diffusion Acceleration in VLAs. A closely related line of work aims to distill multi-step diffusion or flow matching computation in policies into one-step models, or to train one-step models directly. We focus on the VLA context rather than conventional diffusion policies [79, 83, 116, 11, 38, 18, 98, 62, 26, 19, 45, 106, 39]. Distillation-based methods [55, 63, 76] typically involve two stages: first training a standard multi-step model and then distilling it into a one-step model, which significantly increases training cost. Direct one-step training [13, 81] often requires model modifications, such as additional inputs in Shortcut Model [23] or alternative objectives in MeanFlow [27]. These changes make it difficult to fine-tune pretrained VLAs while preserving their strong capabilities. In contrast, FASTER requires no architectural modifications and can be seamlessly integrated into the standard fine-tuning pipeline of flow-based VLAs with the same optimization objective.

Another related direction explores streaming policies, where the immediate action is also produced within a single sampling step [20, 15, 31]. However, these approaches update the observation input at every step, which introduces a substantial inference burden when applied to VLAs. Updating newly acquired visual observations requires a forward pass through the expensive VLM backbone at each step, making real-time VLA deployment impractical. In contrast, our method is specifically designed for VLAs: the VLM backbone is forwarded only once per action chunk, thereby avoiding the primary computational bottleneck in VLA inference. The immediate action is generated through one-step sampling of the action expert after VLM prefilling, while subsequent actions are progressively completed through iterations of the action expert based on the same cached VLM representations.

B Details of Asynchronous Inference Pipeline

Here we provide a fine-grained illustration of the asynchronous inference pipeline. Since the robot controller typically operates at a fixed frequency, all operations occur with an interval of control period Δt_{ctrl}. As shown in Figure 7, the inference latency generally does not align with the controller timesteps in real-world robotic systems. Therefore, if inference starts at time t_0, the predicted actions can only be executed at time $t_0 + \lceil \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rceil \cdot \Delta t_{\text{ctrl}}$. However, the next action is expected to be available at $t_0 + \Delta t_{\text{ctrl}}$ (one control step later), resulting in a delay of d actions:

$$d := \lceil \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rceil - 1 = \lfloor \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rfloor. \tag{7}$$

Note that Δt_{infer} almost never lies exactly on a control-period boundary. Once the current inference finishes, the next inference can only be triggered at the subsequent controller timestep (*e.g.*, t_3 in Figure 7). Consequently, the inference interval is at least $\lceil \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rceil \cdot \Delta t_{\text{ctrl}}$. As discussed in Table 1, the inference interval is related to the execution duration Δt_{exec}, and the minimum feasible execution horizon is therefore $s_{\min} := \lceil \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rceil$.

Table 4: Illustration of inference latency *vs.* sequential execution. "Index" denotes the index of valid actions in the chunk, starting from 0. "Time Req." is the time at which an action is required by the robot controller, while "Time Rec." is the time at which an action is received from the policy server via the streaming interface, reported as mean ± standard deviation over 20 trials. All timestamps are measured relative to the start of inference.

GPU	$\pi_{0.5}$			X-VLA		
	Index	Time Req.	Time Rec.	Index	Time Req.	Time Rec.
RTX 4090	1	66.7ms	$62.1_{\pm 3.1}$ms	1	66.7ms	$44.8_{\pm 0.3}$ms
	2	100.0ms	$70.2_{\pm 3.1}$ms	2	100.0ms	$52.0_{\pm 0.1}$ms
	3	133.3ms	$77.1_{\pm 3.2}$ms	3	133.3ms	$59.6_{\pm 0.3}$ms
RTX 4060	7	266.7ms	$238.6_{\pm 1.9}$ms	3	133.3ms	$129.2_{\pm 2.4}$ms
	8	300.0ms	$253.3_{\pm 2.5}$ms	4	166.7ms	$159.0_{\pm 3.6}$ms
	9	333.3ms	$266.9_{\pm 2.6}$ms	5	200.0ms	$186.8_{\pm 4.3}$ms

C Additional Methodological Details

C.1 Horizon-Aware Schedule with Action Conditioning

We provide details on how the Action Conditioning strategy [5] is integrated into our Horizon-Aware Schedule. It is also worth noting that action prefix unimodalizes the distribution for the immediate next action. Because the subsequent action is heavily constrained by the previous overlapping chunk, it generally does not demand multi-step expressivity.

In the main paper, we use the superscript j in ρ^j and τ_i^j to denote the j-th sampling step. Here we omit j, as the following description applies to both training and sampling. Specifically, given an action prefix of length $d < H$, we set the local flow matching timesteps of the prefix actions to zero[3] (*i.e.*, $\tau_i \equiv 0$ for $i < d$) and apply an offset $-d$ to the action indices. The hit time formulation is accordingly updated from Equation (5) to:

$$\mathbf{u} = \{u_i\} = \left(1 - \left(\frac{i-d}{\max(H-1-d, 1)}\right)^{\alpha}\right) \cdot u_d, \quad i \in [d+1, H-1], \tag{8}$$

where u_d replaces u_0 as the predefined global timestep at which the *first valid action* (now the action at index d) is finalized. Consequently, the immediate action still reaches its hit time $u_d = (N-1)/N$ with a single sampling step, ensuring low latency while preserving the temporal consistency of flow matching. During inference, we simply set d to the real inference delay, where the action prefix corresponds to the overlapping actions $[s, s+d-1]$ from the previous chunk.

During training, d is sampled uniformly from 0 to d_{max} to simulate varying Time to First Action (TTFA) on distinct devices. An action prefix mask $\mathbf{m}$ is introduced to mask the loss function:

$$\mathbf{m} = \{m_i\} = \mathbf{1}(i \geq d), \quad i \in [0, H-1], \tag{9}$$

where $\mathbf{1}(\cdot)$ denotes the indicator function. The overall learning objective is formulated as:

$$\boldsymbol{\tau} = \{\tau_i\} = \begin{cases} 0, & i < d, \\ \max\left(0, (\rho - u_i)/(1-u_i)\right), & i \geq d, \end{cases} \tag{10}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\rho \sim \mathcal{U}(0,1),\, d \sim \mathcal{U}\{0, d_{\text{max}}\}} \frac{\left\| \mathbf{m} \odot \left(v_\theta(\mathbf{o}_t, \mathbf{A}_t^{\boldsymbol{\tau}}, \boldsymbol{\tau}) - (\boldsymbol{\epsilon} - \hat{\mathbf{A}}_t) \right) \right\|^2}{\|\mathbf{m}\|_1}, \tag{11}$$

where $\odot$ is element-wise multiplication and the ℓ_1 norm $\|\mathbf{m}\|_1$ denotes the number of elements equal to one. Therefore, the loss is computed only over the suffix actions.

C.2 Horizon-Aware Schedule Implementation

In the original implementations of flow-based VLAs, the flow-matching timestep is a batch-wise scalar and its conditioning is shared across the action horizon. HAS replaces this scalar with an

[3]The notation of timestep is opposite in Training-time RTC [5], so they set the timesteps to one in their formulation.

index-dependent $[B, H]$ timestep vector constructed according to Equation (6), where each action token is assigned its own local timestep.

For $\pi_{0.5}$, each local timestep is processed by the existing timestep encoder. The resulting $[B, H, D]$ embedding is supplied to adaptive RMSNorm token-wise, rather than broadcasting one $[B, D]$ conditioning vector to all action tokens.

For X-VLA, we also apply the existing timestep encoder to the local timestep values. These per-action embeddings are concatenated with the corresponding action tokens, replacing the original repetition of a single timestep embedding across the horizon.

These changes require adapting the construction logic of timestep embedding to operate per action index, but all layers and parameters remain unchanged, allowing the pretrained weights to be loaded directly.

C.3 Streaming Client-Server Interface

Unlike conventional action chunking pipelines that transmit an entire action chunk as a single large payload, the streaming client-server interface in FASTER employs a progressive streaming mechanism, decomposing the transmission into a sequence of high-frequency, smaller packets. While this mechanism increases the total completion time for a full chunk due to network communication (particularly for the final action), it aligns naturally with the sequential characteristic of robotic execution.

Suppose the i-th action $\boldsymbol{a}_i$ is executed at controller time t. The system only requires the $(i+1)$-th action $\boldsymbol{a}_{i+1}$ to be available at $t + \Delta t_{\text{ctrl}}$. As long as the latency for acquiring $\boldsymbol{a}_{i+1}$ remains below the control period Δt_{ctrl}, the controller avoids pipeline stalls. We provide detailed illustrations in Table 4, including latency measurements for two VLAs on RTX 4090 and RTX 4060 GPUs, respectively. Under this paradigm, the network latency incurred by subsequent actions is effectively masked by the execution time of preceding actions. Consequently, the marginal delay for actions at the end of a chunk becomes functionally negligible and imperceptible during real-time execution.

C.4 Algorithms

The pseudo-code for VLA policy training and inference in FASTER are provided in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 FASTER policy training

Input: Training dataset $\mathcal{D}$, pretrained VLA model v_θ, prediction horizon H, HAS parameters α, u_d, mixing probability p, max prefix length d_{max}
Output: 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 $\rho \sim \mathcal{U}(0,1)$, prefix length $d \sim \mathcal{U}\{0, d_{\text{max}}\}$, schedule type $z \sim \text{Bernoulli}(p)$
  Compute action prefix mask $\mathbf{m}$:

$$\mathbf{m} = \{m_i\} = \mathbf{1}(i \geq d), \quad i \in [0, H-1],$$

  if $z = 1$ **then** ▷ Use Horizon-Aware Schedule
    Compute hit times $\mathbf{u}$:

$$\mathbf{u} = \{u_i\} = \left(1 - \left(\frac{i-d}{\max(H-1-d, 1)}\right)^{\alpha}\right) \cdot u_d, \quad i \in [d+1, H-1],$$

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

$$\boldsymbol{\tau} = \{\tau_i\} = \begin{cases} 0, & i < d, \\ \max\left(0, (\rho - u_i)/(1-u_i)\right), & i \geq d, \end{cases}$$

  else ▷ Use Constant Schedule with Action Conditioning
    Set local timesteps $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = \{\tau_i\} = \begin{cases} 0, & i < d, \\ \rho, & i \geq d, \end{cases}$$

  end if
  Construct noisy actions:

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

  Compute masked loss:

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

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

Algorithm 2 FASTER policy inference

Input: Fine-tuned VLA model v'_θ, sampling steps N, prediction horizon H, HAS parameters α, u_d, observation $\mathbf{o}_t$, delay d, action prefix $\mathbf{A}^{\text{pre}}$, execution horizon s

Initialize $\mathbf{A}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Compute hit times $\mathbf{u}$

$$\mathbf{u} = \{u_i\} = \left(1 - \left(\frac{i-d}{\max(H-1-d, 1)}\right)^{\alpha}\right) \cdot u_d, \quad i \in [d+1, H-1],$$

Forward VLM backbone with $\mathbf{o}_t$

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

  Set global timestep $\rho^j = (N - j + 1)/N$

  Compute local timesteps $\boldsymbol{\tau}^j$ w.r.t. ρ^j:

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

  Similarly, compute next local timesteps $\boldsymbol{\tau}^{j+1}$ w.r.t. ρ^{j+1} ▷ $\rho^{N+1} := 0$

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

  Predict velocity $\mathbf{v}^j = v_\theta(\mathbf{o}_t, \mathbf{A}_t, \boldsymbol{\tau}^j)$ ▷ Forward AE only

  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 $\boldsymbol{a}_{t+i}$ to 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



Figure 8: AgileX Cobot Magic robotic platform with Piper arms.

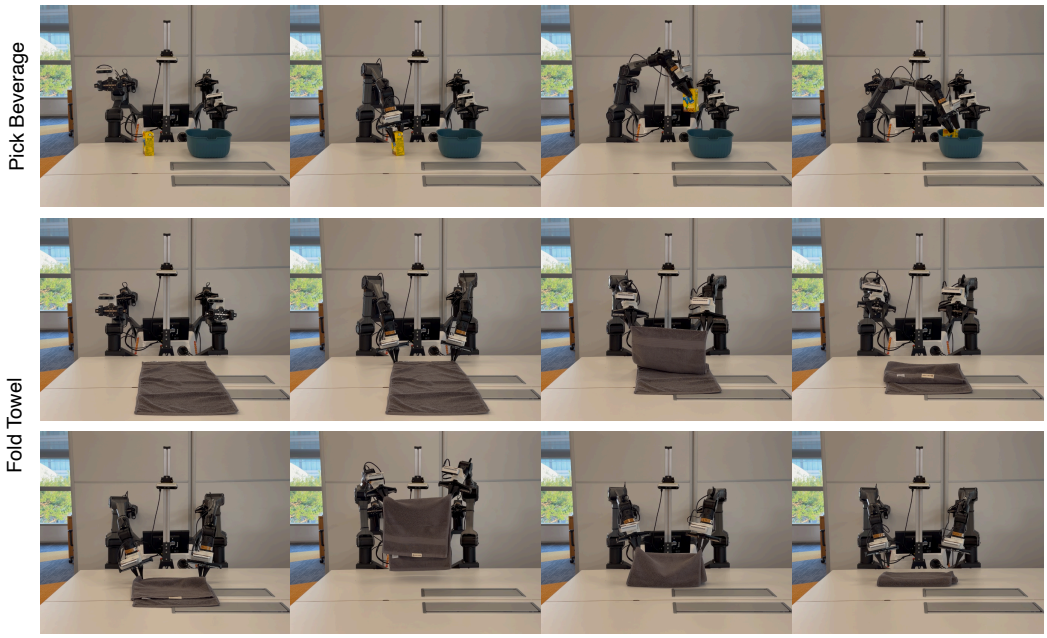


Figure 9: Visualization of Pick Beverage and Fold Towel tasks.

D Implementation Details

D.1 Real-world Experiments

Hardware Setup. We use the AgileX Cobot Magic robotic platform shown in Figure 8, designed in the Mobile Aloha style [118, 24]. It is equipped with four 6-DoF Piper robot arms: two leader arms for human teleoperation and two follower arms for data collection and policy inference. A multi-camera setup is used, consisting of one front camera (RealSense D455) and two wrist cameras (RealSense D435), each mounted on a follower arm.

Tasks. We evaluate three real-robot tasks: "Table Tennis", "Pick Beverage", and "Fold Towel". The visualization of Table Tennis task is already provided in the main paper, while illustrations of the other two tasks are shown in Figure 9. We collect 335 demonstration episodes (approximately 14 minutes) for Table Tennis task and 150 episodes for each of the other two tasks via human teleoperation. All data is recorded at 30 FPS.

In Table Tennis task, the human player and robot are positioned on opposite sides of the table. A marked line is located at the midpoint of the table, approximately 80cm from the base of the robotic arm.

The language instructions for the tasks are as follows:

- Table Tennis: "Hit the table tennis ball to the opponent."
- Pick Beverage: "Pick up the beverage and put it in the plastic basket."
- Fold Towel: "Fold the towel."

Evaluation. The tasks are evaluated with a fixed number of rollouts: 15 trials for Table Tennis, 35 trials for Pick Beverage, and 10 trials for Fold Towel. For Pick Beverage and Fold Towel, we follow predefined test cases with fixed object types, positions, and orientations to ensure fair comparison. For Table Tennis, we regulate the ball speed to the best of our ability.

We define fine-grained evaluation metrics to more precisely assess the real-world performance of the models under a limited number of rollouts. Results are reported by averaging scores across sub-steps and rollouts. The scoring criteria are as follows:

1. Table Tennis
    - Step 1: Hitting the table tennis ball with the racket
        - 0 point: The robot misses the ball.
        - 0.5 point: The robot returns the ball, but the ball travels only a short distance and lands on the table before the marked line.
        - 1 point: The robot returns the ball, and the ball travels a long distance and lands on the table beyond the marked line.
2. Pick Beverage
    - Step 1: Grasping the beverage
        - 0 point: The robot fails to grasp the object.
        - 0.5 point: The robot grasps the object after multiple (up to five) attempts and recovers from errors.
        - 1 point: The robot successfully grasps the object on the first attempt.
    - Step 2: Placing the beverage into the basket
        - 0 point: The object is not placed into the basket or is dropped.
        - 0.5 point: The robot places the object into the basket but causes a collision.
        - 1 point: The robot successfully places the object into the basket on the first attempt.
3. Fold Towel
    - Step 1: Grasping the towel
        - 0 point: The robot fails to grasp both sides of the towel.
        - 0.5 point: The robot grasps both sides after multiple (up to five) attempts and recovers from errors.
        - 1 point: The robot successfully grasps both sides on the first attempt.
    - Step 2: Forward folding the towel
        - 0 point: The robot fails to fold the towel forward (*e.g.*, slippage from the grippers).
        - 0.5 point: The towel is folded forward but not well aligned.
        - 1 point: The robot folds the towel forward perfectly.
    - Step 3: Grasping the towel again
        - 0 point: The robot fails to grasp both sides of the towel.
        - 0.5 point: The robot grasps both sides after multiple (up to five) attempts and recovers from errors.
        - 1 point: The robot successfully grasps both sides on the first attempt.
    - Step 4: Backward folding the towel
        - 0 point: The robot fails to fold the towel backward (*e.g.*, slippage from the grippers).
        - 0.5 point: The towel is folded backward but not well aligned.
        - 1 point: The robot folds the towel backward perfectly.

Table 5: Hyperparameter settings for fine-tuning VLA models.

Model	$\pi_{0.5}$		X-VLA	
Task	AgileX	Simulation	AgileX	Simulation
Prediction Horizon H	50	10	30	30
Action Space	Relative Joint	Delta EEF	Abs EE6D	Abs EE6D
Global Batch Size	128	256	128	128
Training Steps	50k	30k	50k	60k
Optimizer	AdamW	AdamW	AdamW	AdamW
Weight Decay	0	0	0	0
Betas	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)
Base LR	2.5e-5	5e-5	1e-4	1e-4
LR Scheduler	Cosine Decay	Cosine Decay	Constant	Constant
Warmup Steps	1k	10k	2k	2k
Grad Norm Clip	1.0	1.0	1.0	1.0
EMA Decay	0.99	0.999	N/A	N/A

Table 6: Settings of inference delay d and execution horizon s on RTX 4090 and RTX 4060 GPUs. "Async" refers to Naive Async and Training-time RTC.

Task	Method	RTX 4090		RTX 4060	
		d	s	d	s
Table Tennis	Sync	4	5	10	11
	Async	4	5	10	11
	FASTER	3	4	8	10
Pick Beverage, Fold Towel	Sync	4	50	10	50
	Async	4	46	10	40
	FASTER	3	47	8	42

D.2 Simulation Benchmarks

LIBERO. The LIBERO benchmark [53] consists of four task suites: Spatial, Object, Goal, and 10 (Long), each targeting distinct aspects of embodied capabilities. Following prior work [35, 120], we train a single policy across all four suites. For training $\pi_{0.5}$, we use the dataset provided by OpenVLA[4] [43] and convert it to the LeRobot format [8] using the script in `openpi`. While for training X-VLA, we use the HDF5-format dataset provided by X-VLA authors[5], which aligns with the EE6D action space. Each suite contains 10 tasks, and every task is evaluated over 50 trials.

CALVIN. The CALVIN benchmark [68] comprises 34 tasks with unconstrained language instructions spanning diverse manipulation skills. We adopt the widely used ABC→D evaluation setting and use the LeRobot-format dataset[6]. Each model is evaluated on 1,000 unique instruction chains, where each chain consists of five consecutive tasks. Performance is measured by the average number of successfully completed tasks per chain.

Kinetix. The Kinetix benchmark[7] is proposed by RTC [4] and contains 12 dynamic tasks in the Kinetix simulation environment [67]. Since these tasks involve dynamic motions with force-based control, they are suitable for evaluating asynchronous execution methods under varying inference delays. However, the policy used in this benchmark is a simple 4-layer MLP rather than a VLA model, so it is not the primary focus of our study.

Following prior work [4, 5], the policy uses a prediction horizon of $H = 8$. During training, the simulated delay is randomly sampled from 0 to 4 with exponentially decreasing weights. We resume

[4]HuggingFace: `openvla/modified_libero_rlds`
[5]HuggingFace: `2toINF/Libero-XVLA-format`
[6]HuggingFace: `InternRobotics/InternData-Calvin_ABC`
[7]GitHub: `Physical-Intelligence/real-time-chunking-kinetix`

Table 7: Detailed comparison of reaction capability on RTX 4090 and RTX 4060 GPUs. The inference latency Δt_{infer} corresponds to TTFA, and execution duration Δt_{exec} is computed as $s_{\min} \cdot \Delta t_{\text{ctrl}}$. The distribution of reaction time D_{react} is $\mathcal{U}(\Delta t_{\text{infer}}, 2 * \Delta t_{\text{infer}} + \Delta t_{\text{exec}})$ for Sync, and $\mathcal{U}(\Delta t_{\text{infer}}, \Delta t_{\text{infer}} + \Delta t_{\text{exec}})$ for Async and FASTER.

Model	Method	RTX 4090			RTX 4060		
		Δt_{infer}	Δt_{exec}	D_{react}	Δt_{infer}	Δt_{exec}	D_{react}
$\pi_{0.5}$	Sync	80.0ms	100.0ms	$\mathcal{U}(80.0, 260.0)$	303.3ms	333.3ms	$\mathcal{U}(303.3, 939.9)$
	Async	80.0ms	100.0ms	$\mathcal{U}(80.0, 180.0)$	303.3ms	333.3ms	$\mathcal{U}(303.3, 636.6)$
	FASTER	**62.1**ms	100.0ms	$\mathcal{U}(62.1, 162.1)$	**238.6**ms	**266.7**ms	$\mathcal{U}(238.6, 505.3)$
X-VLA	Sync	113.7ms	133.3ms	$\mathcal{U}(113.7, 360.7)$	399.5ms	400.0ms	$\mathcal{U}(399.5, 1199.0)$
	Async	113.7ms	133.3ms	$\mathcal{U}(113.7, 247.0)$	399.5ms	400.0ms	$\mathcal{U}(399.5, 799.5)$
	FASTER	**44.8**ms	**66.7**ms	$\mathcal{U}(44.8, 111.5)$	**129.2**ms	**200.0**ms	$\mathcal{U}(129.2, 329.2)$

training from the 24th epoch and fine-tune the policy for 8 epochs using our mixed schedule. The number of sampling steps is set to 5, and accordingly we set $u_d = 0.8$. Performance is evaluated over 2,048 rollouts, and we report the average success rates.

D.3 Training Setup

We use the official codebases of $\pi_{0.5}$[8] and X-VLA[9] to fine-tune the VLA models. We follow their default or recommended configurations, with key hyperparameters listed in Table 5. Our real-robot dataset is recorded in the absolute joint space. Actions are converted to the relative joint space by computing offsets from the proprioceptive states, and to the absolute EE6D space using forward kinematics. All models are trained on 8 NVIDIA A800 GPUs with 80GB VRAM from cloud provider, requiring 1-2 days for $\pi_{0.5}$ and 6-7 hours for X-VLA.

Regarding the hyperparameters of our proposed Horizon-Aware Schedule, we use $\alpha = 0.6$ to control the hit times and $p = 0.5$ for the mixed schedule by default. When training X-VLA on LIBERO and CALVIN, we set α to 0.7; ablation studies are provided in Section E.5. Since both VLAs use 10 sampling steps for flow matching, we set $u_0 = 0.9$ to ensure that the immediate action requires only a single step, as described in the main paper.

Although the prediction horizon is 50 or 30, we set the maximum prefix length $d_{\max}$ to 10 during fine-tuning real-world tasks, which simulate TTFA up to 333.3ms. This design is motivated by practical considerations: inference delays exceeding 333.3ms can render real-time reactive control functionally infeasible for certain tasks. By limiting the maximum prefix length, we prevent the model from wasting capacity on physically unrealistic scenarios. This range already provides sufficient coverage even for an RTX 4060 GPU, thereby ensuring better sample efficiency and smoother flow matching in the critical low-latency regime.

D.4 Robot Deployment Setup

As mentioned in the main paper, the ROS system on the robot runs at a fixed 30Hz frequency, *i.e.*, with a control period $\Delta t_{\text{ctrl}} = 33.3$ms. The robot is connected to policy server via LAN and communicates through websocket protocol. We define the inference delay as $d := \lfloor \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rfloor$ and the minimum execution horizon as $s_{\min} := \lceil \Delta t_{\text{infer}} / \Delta t_{\text{ctrl}} \rceil$. To account for additional overhead from network communication, local processing, memory I/O, and other system costs, we set these values slightly larger in real-robot deployment, as listed in Table 6.

For the table tennis task, we set the execution horizon to $s_{\min}$ to achieve optimal reaction capability. For other tasks that do not heavily depend on high-frequency control, we set the execution horizon to the number of valid actions (H for Sync, $H - d$ for Async and FASTER) to ensure smooth action trajectories.

[8]Github: `Physical-Intelligence/openpi`
[9]Github: `2toinf/X-VLA`

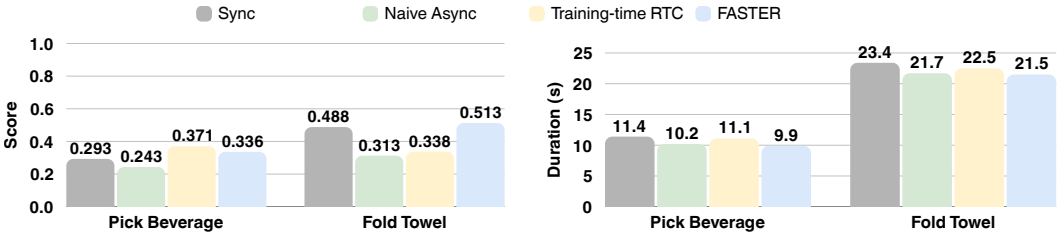


Figure 10: Comparison of real-world performance and task completion duration on two real-world tasks using X-VLA. Note that the duration is computed only from successful rollouts, and therefore is not directly comparable to the results in Figure 6.

Table 8: Performance statistics for the Table Tennis task. The 96% confidence intervals are computed using bootstrap resampling.

Model	Method	RTX 4090		RTX 4060	
		Mean	96% CI	Mean	96% CI
$\pi_{0.5}$	Sync	0.000	(0.000, 0.000)	0.000	(0.000, 0.000)
	Naive Async	0.200	(0.075, 0.325)	0.200	(0.067, 0.333)
	Training-time RTC	0.533	(0.300, 0.767)	0.300	(0.133, 0.500)
	FASTER	0.800	(0.667, 0.933)	0.467	(0.333, 0.567)

E Additional Experimental Results

E.1 Additional Analysis on Reaction Speed

We provide Table 7 to complement the reaction time details in Table 2. The probabilities reported in Table 3 are also derived from the distributions D_{react}, defined as the probability that one independently sampled Δt_{react} from a method is smaller than the other method. As discussed in the main paper, FASTER is deterministically superior on X-VLA, since its upper bound of reaction time is lower than the baselines' lower bound: 111.5ms *vs.* 113.7ms on RTX 4090, and 329.2ms *vs.* 399.5ms on RTX 4060.

E.2 Additional Reaction Time Analysis

Table Tennis: Release-to-Motion Interval. We measure the interval from ball release to robot-motion onset through frame-by-frame annotation of videos recorded at 120 FPS. Experiments are conducted on RTX 4060 using all evaluation rollouts; Sync is excluded from this comparison because it produces no successful returns.

Since ball launch is an external event, the first corresponding robot motion can occur only after a post-release observation and the first executable action has been delivered. This interval is a behavioral timing proxy rather than a direct measurement of system latency. It may also include policy-dependent waiting while the robot estimates the ball trajectory and selects an appropriate interception time, so it is not an exact measurement of system-level latency. Nevertheless, reaction latency consumes part of the available time before interception: a slower response leaves less time to rotate the racket and build up swing speed. Under the same event definition and annotation protocol, this metric provides a consistent basis for relative comparison, though it does not eliminate variation in ball trajectories.

As shown in Table 10, FASTER achieves the lowest mean release-to-motion interval and standard deviation. The aligned supplementary video provides a qualitative comparison under similar ball motion.

Pick Beverage: Recovery from Perturbations. We further evaluate responses to controlled perturbations using $\pi_{0.5}$ on RTX 4090. At a fixed pre-grasp stage, the beverage is moved from an initial position to a target position, requiring the robot to redirect its motion. The object positions, perturbation timing, and the evaluation protocol are fixed across methods. We adopt the latency-

Table 9: Performance statistics for the Pick Beverage and Fold Towel task. The 96% confidence intervals are computed using bootstrap resampling.

Model	Method	Pick Beverage				Fold Towel			
		Score		Duration (s)		Score		Duration (s)	
		Mean	96% CI	Mean	Std	Mean	96% CI	Mean	Std
$\pi_{0.5}$	Sync	0.879	(0.786, 0.950)	13.0	0.9	0.788	(0.600, 0.925)	24.7	0.5
	Naive Async	0.957	(0.886, 1.000)	12.5	1.2	0.825	(0.613, 0.988)	24.0	2.4
	Training-time RTC	0.950	(0.879, 0.993)	11.9	2.4	0.888	(0.700, 1.000)	20.7	0.4
	FASTER	0.957	(0.886, 1.000)	12.0	0.9	0.963	(0.925, 1.000)	20.5	0.4

Table 10: Comparison of ball release-to-motion intervals on RTX 4060.

Method	Mean	Std.
Naive Async	916ms	327ms
Training-time RTC	855ms	323ms
FASTER	**683**ms	**96**ms

Table 11: Comparison of recovery time and successful grasps after object relocation.

Method	Mean	Std.	Success
Sync	3.34s	0.29s	**10/10**
Naive Async	2.83s	0.17s	9/10
Training-time RTC	2.35s	0.18s	**10/10**
FASTER	**2.23**s	**0.06**s	**10/10**

sensitive deployment configurations used for Table Tennis in Table 6 and conduct 10 trials per method. Recovery time is measured from the onset of object relocation to a successful grasp at the new position, and statistics are computed over successful trials only.

Table 11 shows that FASTER achieves the lowest mean recovery time and succeeds in all 10 trials, with no unstable behavior observed. Qualitatively, Sync exhibits pauses at chunk boundaries, while Naive Async introduces inter-chunk discontinuities and fails one grasp. Training-time RTC and FASTER produce smoother transitions. The shorter recovery time of FASTER is consistent with its reduced latency in generating corrective actions.

E.3 Additional Real-world Experiments

We evaluate FASTER on real-world Pick Beverage and Fold Towel tasks using the X-VLA [120] model and present the results in Figure 10. We observe that though X-VLA performs significantly worse than the state-of-the-art VLA $\pi_{0.5}$, our method consistently achieves better or comparable results in terms of both completion scores and task duration. It is worth noting that Naive Async often fails due to unstable actions caused by inter-chunk discontinuities.

We supplement the performance statistics of Figure 5 and Figure 6 in Table 8 and Table 9, respectively.

E.4 Simulation Benchmarks

We conduct experiments on two widely used simulation benchmarks in VLA research: LIBERO [53] and CALVIN [68]. Although these simulation environments are not directly affected by inference latency, they provide a controlled protocol for evaluating whether FASTER preserves the original model performance. As shown in Table 12, HAS maintains competitive performance on both benchmarks, with limited degradation despite its aggressive action sampling strategy. These results suggest that, although FASTER may slightly affect long-horizon accuracy in simulation, it provides a favorable trade-off for latency-sensitive real-world settings.

We further evaluate on the Kinetix benchmark [4] to compare with state-of-the-art real-time methods. We include additional baselines: Bidirectional Decoding (BID) [57], Inference-time RTC [4], VLASH [89], and REMAC [94]. Since flow-matching sampling is applied to the entire model in this benchmark, FASTER reduces the inference latency of the immediate action by $5\times$. We conduct fair comparisons under the *latency-aware setup*: given the maximum supported delay is 4, we compare the baselines at $d = 4$ with FASTER at $d = 1$ under the same execution horizon. As shown in Table 13, FASTER outperforms all baselines by a clear margin, highlighting the importance of reducing reaction latency in asynchronous execution, even in simulated environments.

Table 12: Comparison of performance on simulation benchmarks LIBERO and CALVIN. "$*$" indicates experimental results evaluated on our cluster with official checkpoints.

Method	LIBERO					CALVIN ABC→D					
	Spatial	Object	Goal	10	Avg.	1	2	3	4	5	Avg. Len
$\pi_{0.5}$	98.8	98.2	98.0	92.4	96.9	94.2	88.7	85.7	83.2	79.5	4.313
$\pi_{0.5}$+FASTER	98.6	97.8	97.8	91.6	96.5	95.1	89.1	85.0	81.9	78.1	4.292
X-VLA*	97.8	99.4	97.8	96.8	98.0	95.7	89.8	82.4	77.0	70.2	4.151
X-VLA+FASTER	97.8	99.4	98.2	93.0	97.1	98.1	91.4	82.4	72.9	64.5	4.093

Table 13: Comparison of performance on Kinetix benchmark under the latency-aware setup.

Method	d	s	Solve Rate
Naive Async	4	4	0.492
BID [57]	4	4	0.553
Inference-time RTC [4]	4	4	0.614
Training-time RTC [5]	4	4	0.726
REMAC [94]	4	4	0.779
VLASH [89]	4	4	0.813
FASTER	1	4	**0.869**

We compare FASTER with uniform few-step generation baselines, including DPM-Solver++ 2M [61] and two second-order ODE solvers, Midpoint and Heun in Table 14. Here we consider two evaluation setups. In the *latency-aware setup*, the response delay is determined by the number of function evaluations (NFE) required before the first action becomes available, capturing the joint effect of action quality and responsiveness. In the *delay-controlled setup*, all methods share the same simulated delay of $d = 1$, isolating closed-loop action-generation quality from differences in response delay. The execution horizon is set to $s = 4$.

We use Training-time RTC with a 5-step Euler solver as the baseline. We also evaluate two FASTER configurations, with $u_d = 0.8$ and $u_d = 0.6$, which finalize the first action after one and two sampling steps, respectively, while continuing to refine subsequent actions.

DPM-Solver++ 2M uses a first-order initialization before applying its second-order multistep update. It is therefore equivalent to first-order solver in the one-step setting and provides no improvement over Euler. At two steps, its solve rate remains comparable to Euler. Midpoint improves action quality in some delay-controlled comparisons, but its two NFEs per step increase the effective response delay in the latency-aware setting. Heun performs poorly in this few-step regime.

FASTER obtains the highest solve rates among the few-step methods in both settings and approaches the 5-step baseline in the delay-controlled comparison. The asymmetric allocation of HAS prioritizes the near-term actions to reduce latency while largely preserving action quality by allocating additional refinement to later actions. These results illustrate the trade-off: additional solver evaluations may improve integration accuracy, but their associated response delay can reduce performance in dynamic tasks.

E.5 Ablation Study

We conduct ablation studies to analyze the impact of our proposed methods on task performance. Since real-world experiments are difficult to repeat with a large number of rollouts and are easily affected by environmental factors, we use simulation benchmarks for more controlled evaluation. Moreover, as the LIBERO benchmark is highly saturated, we focus on CALVIN to better highlight long-horizon performance.

We first investigate the factor α in the Horizon-Aware Schedule, which controls the decreasing trend of hit times across the action index. As shown in Figure 11, a smaller α leads to a faster decay of hit times, allocating more denoising steps to future actions. From Table 15, we observe that HAS is robust to different values of α, except when $\alpha = 1.0$, with the largest difference in Avg. Len being only 0.18.

Table 14: Comparison of performance with uniform few-step generation baselines on Kinetix benchmark under the delay-controlled and latency-aware setups.

Method	Steps to 1st Action	NFE to 1st Action	Delay-controlled d	Delay-controlled Solve Rate	Latency-aware d	Latency-aware Solve Rate
Euler (Baseline)	5	5	1	0.875	4	0.726
Euler	1	1	1	0.713	1	0.713
DPM-Solver++ 2M [61]	1	1	1	0.713	1	0.713
Midpoint	1	2	1	0.862	2	0.816
Heun	1	2	1	0.373	2	0.320
FASTER	1	1	1	**0.869**	1	**0.869**
Euler	2	2	1	0.790	2	0.757
DPM-Solver++ 2M [61]	2	2	1	0.787	2	0.756
Midpoint	2	4	1	0.853	4	0.709
Heun	2	4	1	0.675	4	0.531
FASTER	2	2	1	**0.874**	2	**0.845**

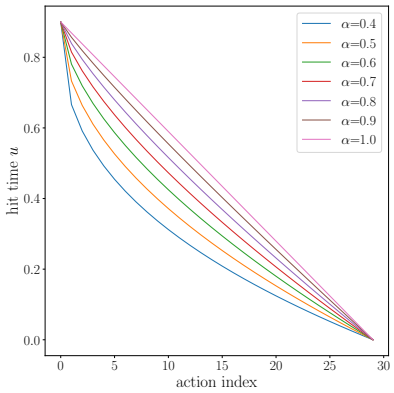


Figure 11: Hit times used in ablation study, with factor α from 0.4 to 1.0.

Table 15: Ablation study of factor α in the Horizon-Aware Schedule, using X-VLA on the CALVIN benchmark. Mixing probability p is set to 0.5.

α	CALVIN ABC→D 1	2	3	4	5	Avg. Len
0.4	96.7	91.3	82.5	73.1	63.5	4.071
0.5	95.1	88.4	79.3	69.6	58.7	3.911
0.6	97.5	89.5	80.2	71.2	60.7	3.991
0.7	98.1	91.4	82.4	72.9	64.5	**4.093**
0.8	95.9	88.7	79.7	71.2	61.5	3.970
0.9	99.0	88.1	75.2	70.3	59.4	3.921
1.0	94.7	84.0	71.7	61.3	51.8	3.635

We further analyze the influence of the mixed schedule in FASTER training. In addition to varying the mixing probability p between HAS and the conventional constant schedule ($p = 0$), we also include the independent time schedule proposed in Diffusion Forcing [10], where the timestep is sampled independently across the action index during training while HAS is used for sampling.

As shown in Table 16, a small value of p leads to degraded performance, since the inference-time schedule is observed less frequently during training. As discussed in the main paper, fine-tuning a pretrained VLA primarily with HAS, *i.e.*, using a large value of p, can also introduce severe adverse effects, particularly on long-horizon performance. Although training with the independent schedule achieves reasonable accuracy, it is not directly comparable to our training strategy: independently sampling timesteps for each action in training enlarges the search space and can induce inconsistencies between training and inference [87]. These results highlight the importance of the mixed schedule in FASTER. We therefore use the default value $p = 0.5$ for all reported results and do not further tune p.

We also evaluate different values of α in real-world experiments with two models and observe no clear performance differences. This suggests that task-specific hyperparameter tuning is not necessary for HAS, and that our default configuration generalizes well across different settings.

E.6 Error Analysis

We evaluate the action prediction accuracy of FASTER through open-loop comparisons against ground-truth demonstrations using fine-tuned $\pi_{0.5}$ models on real-world tasks. Predictions and ground truth are converted to the same absolute joint space before computing errors, ensuring consistent coordinates and units across action indices. We compare the standard 10-step constant schedule, a uniform 1-step constant schedule, and HAS to characterize the accuracy-latency trade-off.

Table 16: Ablation study of mixing probability p in the mixed schedule, using X-VLA on the CALVIN benchmark. Factor α is set to 0.7.

p	CALVIN ABC→D					
	1	2	3	4	5	Avg. Len
Baseline	95.7	89.8	82.4	77.0	70.2	4.151
0.3	93.7	85.2	76.0	65.5	55.2	3.756
0.5	98.1	91.4	82.4	72.9	64.5	**4.093**
0.7	89.6	76.7	63.2	50.8	40.3	3.206
1.0 (w/o mixed)	89.0	74.7	60.4	49.0	38.1	3.112
Independent	91.4	82.6	74.0	64.6	54.5	3.671

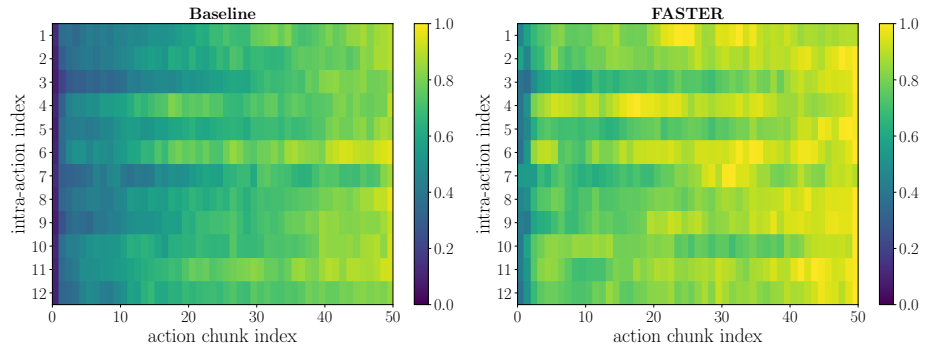


Figure 12: Open-loop action prediction error. We compare the full-step baseline with FASTER. The heatmaps report the mean absolute error for each action dimension and action chunk index, normalized by the maximum error of each action dimension computed over 200 random samples and shared across both models.

Figure 12 visualizes the per-dimension normalized mean absolute error (MAE) of the 10-step baseline and FASTER across action chunk indices. Both methods exhibit an overall increase in error toward later action indices, supporting our main insight that near-term actions are generally easier to predict. FASTER incurs higher errors on some action dimensions, indicating a fidelity cost associated with accelerated sampling. Nevertheless, these errors are distributed more uniformly across chunk indices and remain relatively low for near-term actions, which are directly executed for immediate reaction. This suggests that HAS preserves sufficient precision for latency-critical actions while enabling substantially faster response in real-time control.

We complement this visualization with a quantitative comparison. Table 17 reports MAE averaged over evaluation samples and action dimensions. For the first action, FASTER yields an MAE of 2.07, compared with 0.50 for the 10-step baseline and 2.09 for uniform 1-step sampling. The approximately $4.1\times$ relative increase over the full-step baseline quantifies the immediate-action fidelity cost. Nevertheless, this error remains lower than the baseline's error at the 15th action (2.62). Thus, although 1-step sampling is less accurate than full-step baseline for the immediate action, its absolute error remains within the error range observed in the baseline.

FASTER does not apply the one-step approximation uniformly to the entire chunk. As subsequent actions receive progressively more refinement, their accuracy recovers: relative to uniform one-step sampling, FASTER reduces the MAE consistently. At the 50th action, FASTER approaches the full-step baseline (4.01 *vs.* 3.65), while uniform 1-step sampling remains substantially less accurate (5.74).

Together with the latency measurements and closed-loop evaluations, these results characterize the intended trade-off: FASTER accepts a measurable loss in immediate-action fidelity to reduce TTFA, while additional refinement helps preserve the quality of later actions.

Table 17: Open-loop action prediction error. Values are mean absolute error in absolute joint space, averaged over 200 random samples and action dimensions and multiplied by 10^3. Column headings indicate positions within the predicted action chunk.

Method	1st	2nd	15th	50th
10-step Baseline	0.50	1.19	2.62	3.65
1-step Constant	2.09	2.53	4.15	5.74
FASTER	2.07	2.27	3.81	4.01

F Limitations and Future Work

Despite the improved reaction capability demonstrated by FASTER, several limitations remain. First, our method is primarily applicable to flow-based or diffusion-based VLA models whose action generation process involves iterative sampling. The actual latency reduction also depends on the implementation and hardware. For example, we observe that in JAX implementations, the forward-pass runtime is not always proportional to the number of iterations, which may limit the practical speedup obtained.

Second, FASTER is a deployment-oriented inference scheduling and streaming framework rather than a fundamental modeling advance. Thus, it does not by itself resolve limitations inherited from the underlying VLA model, such as perception failures, language grounding errors, or inaccurate action generation. The robot may still fail if the base policy produces incorrect actions for the task.

Third, our reaction time analysis relies on a simplified timing model. In particular, we assume that sensing, network communication, pre- and post-processing, model inference, and action dispatch can be approximated by a single effective latency term. We also assume periodic inference and uniformly distributed event phases. These assumptions lead to a uniform reaction-time distribution that provides a useful first-order characterization of responsiveness, but may not fully capture all deployment conditions. In practice, system latency may vary due to CPU/GPU scheduling, network jitter, memory contention, or OS overhead.

Finally, the Horizon-Aware Schedule introduces a trade-off between responsiveness and immediate-action fidelity. Although our experiments show that FASTER largely preserves task performance and often improves real-world execution by reducing delay, aggressive early sampling may slightly perturb the original generation trajectory, especially during precision-sensitive phases such as grasping and contact. Improved solvers and adaptive schedules that depend on uncertainty, task phase, or online feedback may help balance accuracy and responsiveness.

G Broader Impacts

FASTER aims to improve the real-time responsiveness of VLA models for physical robot control. Its positive impacts include enabling more reliable robot behavior in dynamic environments, reducing the hardware requirements for deploying generalist policies, and making real-time embodied AI systems more accessible on consumer-grade GPUs.

At the same time, it introduces potential risks. More reactive robots can execute actions faster in the physical world, which may amplify the severity of failures when VLAs make incorrect predictions, misinterpret instructions, or encounter out-of-distribution observations. The method could also contribute to broader automation capabilities. While this can improve productivity, it also raises concerns about labor displacement or misuse.

Our experiments are conducted in controlled research settings. We do not recommend deploying such robot systems in safety-critical or unsupervised human-facing environments without additional risk assessment, safety monitors, and appropriate physical safeguards.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the main contributions: analyzing reaction time in action chunking policies, introducing TTFA, proposing FASTER with a Horizon-Aware Schedule and streaming client-server interface, and validating the method on real-world robotic tasks and simulation benchmarks. The claims are supported by the systematic analysis, methodology, and experiments reported in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations in Appendix F, including the focus on flow-based policies, dependence on base model, reaction time assumption, and the trade-off between responsiveness and action accuracy.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper provides a systematic analysis of reaction time in Section 2, including the definitions and assumptions used to model reaction time as a uniform distribution. The derivations are used to motivate TTFA and the design of FASTER rather than to establish a standalone theorem.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper reports all information including hardware platforms, deployment settings, real-world tasks, simulation benchmarks, implementation details, and hyperparameter settings in Section 4 and Appendix D. We also provide the detailed training and inference algorithms in Algorithm 1 and Algorithm 2 that can be used to reproduce the proposed method. The code is publicly available at `https://github.com/hku-sail/FASTER`.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code and deployment instructions are publicly available at `https://github.com/hku-sail/FASTER`. Since the real-robot platform and experimental environment cannot be fully replicated, we do not plan to release the collected real-world task data. For the simulation experiments, we provide detailed information about the datasets in Appendix D; these datasets are publicly available.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all details in Section 4 and Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: The paper reports latency statistics with mean and standard deviation and includes uncertainty visualization in the pilot study. Details are provided in Table 4 due to limited space. For real-world robot evaluations, we report average completion scores and durations across rollouts using predefined test cases, with 96% CI and standard deviation reported in Table 8 and Table 9; however, large-scale statistical testing is limited by the cost and variability of real-robot experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The detailed information is provided in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work is conducted for robotics research in controlled environments. The paper discusses potential risks and emphasizes the safe deployment of robot policies.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix G discusses both positive and negative impacts of this work.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release models or datasets. We nevertheless discuss physical deployment safeguards in Appendix G.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the existing models, benchmarks, datasets, and codebases used in the experiments. The corresponding sources are listed in the references and Appendix D.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Our released code is accompanied by clear documentation and license.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. All real-world demonstration data is collected by the authors.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method does not use LLMs as an original or non-standard component. The use of language-capable VLA backbones follows existing pretrained model setups, and the LLM backbones are standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.