Action ControlNet: A Lightweight Delay-Aware Adapter for Smooth Asynchronous Control in Vision-Language-Action Models

Tiecheng Guo and Meng Guo*

***Abstract*— Vision-language-action (VLA) models have shown strong potential for general-purpose robot manipulation, but their inference latency remains a major obstacle to stable high-frequency control. Asynchronous execution mitigates this bottleneck by overlapping policy inference with action execution, yet the next action chunk is still predicted from stale observations while the robot continues to move. Direct chunk stitching therefore introduces handoff discontinuities, action jitter, and failures in contact-rich manipulation. Existing remedies typically require either full-policy retraining or architecture-specific runtime logic. This work proposes Action ControlNet (ACNet), a lightweight delay-aware adapter that uses the executed motion suffix as a residual condition for a mostly frozen action head. ACNet leaves the pretrained backbone unchanged, introduces few trainable parameters, and remains compatible with generative action heads such as diffusion and flow matching. On Kinetix, Meta-World MT50, and a real-world SO-ARM101 platform, ACNet improves robustness under inference delay and yields smoother asynchronous trajectories than direct chunk stitching, while remaining more lightweight than full delay-conditioned retraining.**

I. Introduction

Vision-language-action (VLA) models [1] have recently emerged as a promising framework for general-purpose robot manipulation, benefiting from large-scale multimodal pretraining that supports broad task coverage and instruction following across diverse scenarios. Modern VLA policies often pair large vision-language backbones with generative action heads, such as diffusion [2] or flow matching [3], and generate short action chunks rather than individual actions. In the standard synchronous deployment loop, the robot first waits for policy inference to finish and only then executes the predicted chunk. Because inference latency is non-negligible for large backbones and iterative action heads, this infer-then-execute schedule introduces idle intervals between consecutive chunks. In practice, these idle intervals appear as visible stop-and-go pauses during manipulation, making the robot motion less continuous and increasing the wall-clock time required to complete a task. This synchronous latency bottleneck motivates asynchronous execution, where model inference is overlapped with robot motion.

Asynchronous inference allows the controller to avoid idle waiting and maintain a higher effective action frequency [4]. Yet this strategy primarily addresses scheduling rather than modeling. The next action chunk is still generated from an observation that is stale when the chunk is finally applied, and direct stitching of consecutive chunks can break continuity at the handoff boundary. The resulting discontinuities often appear as abrupt action changes, oscillatory corrections, and trajectory jitter, and in contact-rich or precision-sensitive manipulation they can accumulate into outright task failure. Existing remedies alleviate this issue only partially. Runtime inpainting or interpolation can smooth local discontinuities, but such methods are often heuristic and architecture-dependent. Auxiliary correction heads can compensate for delay-induced errors, but many are tightly coupled to specific action representations or require nontrivial policy modification. Training-time delay simulation and full-policy adaptation offer a more direct route to robustness, but they increase training cost substantially and weaken the practical advantage of reusing large pretrained VLA policies.

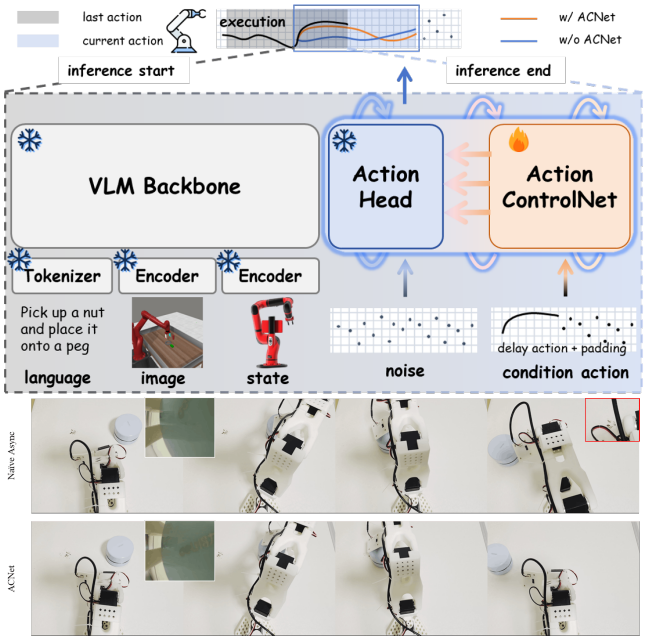

Fig. 1. **Top**: Overview of the asynchronous VLA control setting and the proposed ACNet; **Bottom**: Real-world asynchronous rollouts on the SO-ARM101 platform, for the *clean the table* task. In contact-rich manipulation, inter-chunk jitter in Naïve Async (**upper**) can shift the contact point and lead to task failure, whereas ACNet (**lower**) maintains smoother contact.

The dominant error in asynchronous chunked control is treated here as a local handoff problem rather than a global loss of task understanding. Under inference delay, the visual and language inputs can often still specify the task and coarse plan correctly, while the unreliable component is the continuation from the motion already being executed. This motivates treating delayed VLA control as a boundary

The authors are with the School of Advanced Manufacturing and Robotics, Peking University, Beijing 100871, China.
*Corresponding author: Meng Guo, meng.guo@pku.edu.cn.

conditioning problem at the chunk handoff. Following this perspective, **Action ControlNet (ACNet)** is introduced as a parameter-efficient module inspired by ControlNet [5]. ACNet encodes the short action suffix executed during inference latency, referred to as the *delay action*, and injects it into a largely frozen action head as a residual correction. This provides the action expert with an explicit boundary condition while preserving the task semantics encoded by the frozen backbone of the perception-language networks.

The contributions are threefold. (I) delay-induced degradation in asynchronous chunked VLA control is formulated as a boundary-conditioning problem at chunk handoff, with the executed motion suffix identified as the key conditioning signal. (II) ACNet is proposed as a lightweight action-head-level residual adapter that conditions a largely frozen action head on this suffix, requires only a few trainable parameters, and can be attached as a plug-and-play module to existing chunked VLA policies with generative action heads such as diffusion and flow matching. (III) simulation results show that ACNet achieves performance comparable to full delay-conditioned retraining on Kinetix and Meta-World MT50, and real-world experiments show visibly reduced transition stuttering on a SO-ARM101 platform.

II. Related Work

A. Vision-Language-Action Models and Generative Policies

Different from earlier explicit skill modeling and demonstration-based manipulation methods [6], [7], large-scale pretraining has accelerated generalist robot policies that combine perception, language, and action. Representative VLA systems include RT-1 [8], RT-2 [1], Octo [9], OpenVLA [10], and π_0 [11], which improve semantic generalization, transferability, and policy capacity across manipulation tasks. In parallel, action models such as ACT [12], Diffusion Policy [2], flow matching [3], and Q-Transformer [13] improve chunked or generative control. These models are expressive, but their inference cost makes high-frequency closed-loop deployment difficult. ACNet complements this line of work by adapting the action head for delayed asynchronous execution.

B. Asynchronous Inference and Delay-Aware Robot Control

Asynchronous inference improves throughput by decoupling computation from execution. In the simplest setting, as used in SmolVLA [14] and denoted as Naïve Async , the robot executes the current chunk while the policy predicts the next one. This removes idle waiting, but the next chunk is still generated from stale observations, creating prediction–execution mismatch at handoff. Prior methods address this mismatch through action inpainting (RTC [4]), lightweight correction heads [15], future-state prediction (VLASH [16]), or training-time delay simulation [17]. These methods show that asynchronous control requires temporal alignment inside the policy. ACNet targets this gap with parameter-efficient residual conditioning in a mostly frozen action head.

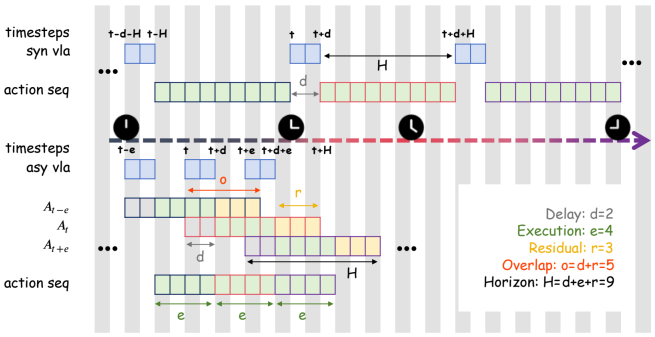

Fig. 2. Comparisons between synchronous (**Top**) and asynchronous (**Bottom**) execution for chunked VLA policies.

III. Problem Formulation

Consider a pretrained chunked vision-language-action (VLA) policy $\mathcal{M}_\theta$ with parameters θ. At control step t, the policy receives the current observation o_t, which may include visual and proprioceptive inputs, together with a language instruction l, and predicts an action chunk of horizon H:

$$\mathbf{a}_t = \mathcal{M}_\theta(o_t, l) = \{a_t^{(0)}, a_t^{(1)}, \cdots, a_t^{(H-1)}\}, \quad \mathbf{a}_t \in \mathbb{R}^{H \times d_a}, \tag{1}$$

where d_a is the action dimensionality. The chunk index t denotes the physical control step at which the corresponding inference is launched.

In asynchronous chunked execution, a new prediction is launched every e control steps, while the predicted chunk becomes available only after an inference delay of d steps. During this interval, the robot continues executing the previous chunk. Thus, the chunk launched at step t is generated from (o_t, l) but is applied only at time $t+d$. Its first d actions are therefore stale when the chunk becomes available. For boundary analysis, the chunk horizon is described as:

$$H = d + e + r, \tag{2}$$

where e is the relaunch interval, d is the inference delay, and r is an optional future suffix retained beyond the handoff window. This horizon accounting is not intended as a new execution model; it only makes explicit which part of the previously predicted chunk has been executed during inference latency. At the handoff time $t+d$, the first executable action of the new chunk, $a_t^{(d)}$, should continue smoothly from the last executed action of the previous chunk, $a_{t-e}^{(e+d-1)}$. The relevant motion context is therefore the executed suffix of the previous chunk:

$$\mathbf{a}_t^{\text{delay}} = \{a_{t-e}^{(e)}, \cdots, a_{t-e}^{(e+d-1)}\} \in \mathbb{R}^{d \times d_a}. \tag{3}$$

This short executed suffix is referred to as the delay action. Since the action head expects an H-step tensor, the delay action is padded to horizon H and denoted by $\tilde{\mathbf{a}}_t^{\text{delay}}$. A pretrained chunked VLA policy can be decomposed into a backbone of perception-language network followed by an action head as described below:

$$\mathcal{M}_\theta(o_t, l) = \mathcal{A}_\psi(\mathcal{B}_\omega(o_t, l)), \tag{4}$$

where $\theta = \{\omega, \psi\}$, $\mathcal{B}_\omega$ denotes the perception-language backbone, and $\mathcal{A}_\psi$ denotes the action head. The key observation is that asynchronous delay primarily introduces a local handoff mismatch rather than a global failure of task understanding. The observation and language instruction usually still specify the task semantics and coarse plan, while the unreliable component is the continuation from the motion already executed during the delay. Therefore, instead of relearning the full visuomotor mapping, the goal is to construct a delay-aware policy conditioned on both the task context and the following executed motion context:

$$\widehat{\mathbf{a}}_t = \widehat{\mathcal{M}}_{\theta,\eta}(o_t, l, \widetilde{\mathbf{a}}_t^{\text{delay}}), \tag{5}$$

where η denotes newly introduced trainable parameters. The adapted policy $\widehat{\mathcal{M}}_{\theta,\eta}$ is required to preserve the pretrained task semantics of $\mathcal{M}_\theta$ while reducing the handoff mismatch.

Formally, given a training distribution $\mathcal{D}$ over $(o_t, l, \mathbf{a}_t^*)$ and a deployment delay distribution $p(d)$, the goal is to seek a parameter-efficient adaptation $\eta^\star$ that minimizes the expected delayed-control risk as:

$$\eta^\star = \arg\min_{\eta:\,|\eta| \ll |\theta|} \mathbb{E}\left[\mathcal{L}_{\text{pred}}(\widehat{\mathbf{a}}_t, \mathbf{a}_t^*) + \lambda \mathcal{L}_{\text{bd}}(\widehat{\mathbf{a}}_t, \widetilde{\mathbf{a}}_t^{\text{delay}})\right], \tag{6}$$

where $(o_t, l, \mathbf{a}_t^*) \sim \mathcal{D}$ and $d \sim p(d)$. Here, $\mathcal{L}_{\text{pred}}$ is the base action-generation objective, $\mathcal{L}_{\text{bd}}$ is a boundary-consistency term, and $\lambda \geq 0$ balances task fidelity against handoff smoothness. A natural boundary loss is given by:

$$\mathcal{L}_{\text{bd}} = \left\| \widehat{a}_t^{(d)} - a_{t-e}^{(e+d-1)} \right\|_2^2, \tag{7}$$

where $\widehat{a}_t^{(d)}$ is the first executable action of the predicted chunk and $a_{t-e}^{(e+d-1)}$ is the last executed action of the previous chunk. Thus, the problem is to adapt a pretrained chunked VLA policy to asynchronous deployment by using the executed delay action as a boundary condition, while introducing only a small number of trainable parameters and avoiding full backbone retraining.

IV. Action ControlNet

ACNet implements the boundary-conditioned objective by augmenting a pretrained VLA model only at the action head. This design follows four requirements: preserve the pretrained backbone (**R1**), encode the executed suffix (**R2**), apply the correction locally in the action head (**R3**), and remain efficient across sampled delays (**R4**). Specifically, the delay-aware policy is factorized as:

$$\widehat{\mathcal{M}}_{\theta,\eta}(o_t, l, \widetilde{\mathbf{a}}_t^{\text{delay}}) = \widehat{\mathcal{A}}_{\psi,\eta}\Big(\mathcal{B}_\omega(o_t, l), \widetilde{\mathbf{a}}_t^{\text{delay}}\Big), \tag{8}$$

where $\mathcal{B}_\omega$ is the frozen perception-language backbone, $\widehat{\mathcal{A}}_{\psi,\eta}$ is the delay-aware action head, and η collects the new trainable parameters. The parameter set η is instantiated as **Action ControlNet (ACNet)**, a lightweight residual conditioning branch that preserves the backbone and adapts only the action head. The overall architecture is shown in Fig. 3.

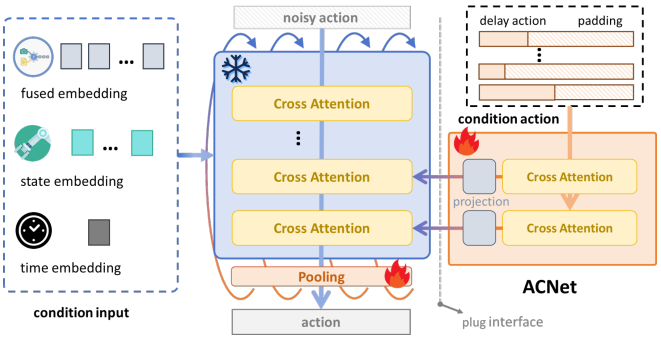

Fig. 3. Architecture of ACNet. The pretrained perception-language backbone and main action expert are kept mostly frozen, while the delay action is encoded by a lightweight side branch and injected into the action head through projection layers as residual conditioning.

A. Delay-Action Encoding

At time $t + d$, the latent $\mathcal{B}_\omega(o_t, l)$ is available, but the new chunk must continue the motion already executed during the delay. The observed delay segment has length d, while the policy expects an H-step tensor. The executed suffix is therefore retained explicitly, and the unobserved future positions are padded with learnable tokens, i.e.,

$$\widetilde{\mathbf{a}}_t^{\text{delay}} = \left[a_{t-e}^{(e)}, \cdots, a_{t-e}^{(e+d-1)}, \mathbf{p}_d, \cdots, \mathbf{p}_{H-1}\right], \tag{9}$$

where $\mathbf{p}_j \in \mathbb{R}^{d_a}$ denotes the padding token at future position j. Learnable padding explicitly marks unavailable slots, avoiding the ambiguity of zero actions and the artificial variation of noise padding.

ACNet then applies a lightweight transformer encoder $\mathcal{E}_\phi$ to $\widetilde{\mathbf{a}}_t^{\text{delay}}$, followed by the same terminal temporal pooling operator used in the action expert, yielding

$$\mathbf{c}_t = \text{Pool}\Big(\mathcal{E}_\phi(\widetilde{\mathbf{a}}_t^{\text{delay}})\Big) \in \mathbb{R}^{d_c}, \tag{10}$$

where $\mathbf{c}_t$ represents the short-horizon executed suffix. Reusing the expert's terminal pooling aligns this context with the action head's temporal abstraction instead of introducing a separate control stream.

B. Residual Injection as Local Boundary Correction

Because the pretrained backbone already encodes the task context, ACNet injects the delay context only into the action head. The injection is residual so that the original predictor remains the default mode. Let $\mathbf{h}_l$ denote the hidden state entering block l of an L-layer action expert, and let $\mathcal{S} \subseteq \{1, \cdots, L\}$ be the set of injected blocks. For each $l \in \mathcal{S}$, the proposed ACNet applies the following update:

$$\mathbf{h}_l' = \mathbf{h}_l + \mathcal{Z}_{\phi,l}(\mathbf{c}_t), \tag{11}$$

where $\mathcal{Z}_{\phi,l}$ projects $\mathbf{c}_t$ to the hidden width of the action expert. This compact side branch can collapse toward zero when the delay cue is uninformative and acts near the stage where the output trajectory is formed.

Moreover, let g_l denote the downstream mapping from the hidden state in block l to the first executable action $\widehat{a}_t^{(d)}$. If

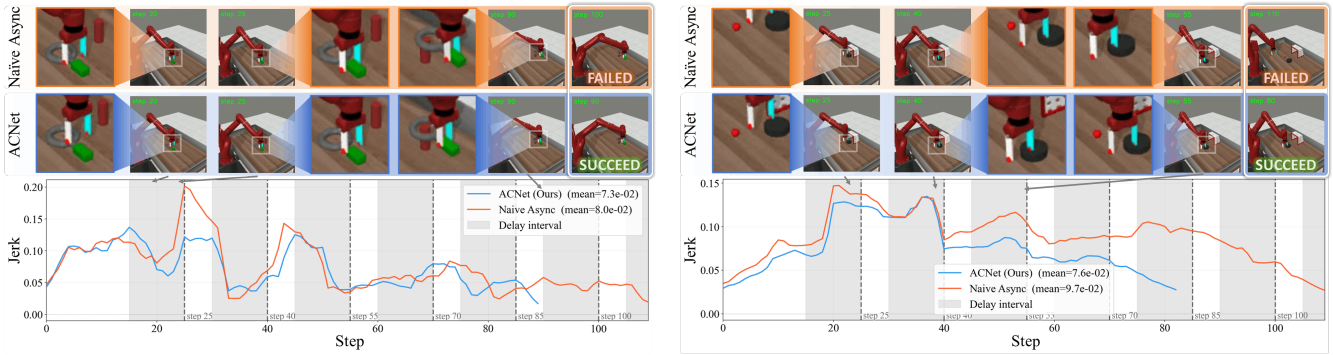

Fig. 4. **Left**: nut-assembly-v3 task; **Right**: plate-slide-back-v3 task. Action jerk over the first 50 executed steps for representative asynchronous rollouts with $H = 50$ and $d = 10$. Gray regions denote delayed intervals, and dashed lines denote chunk replacement events. ACNet yields a flatter jerk profile than Naïve Async around the handoff boundary, indicating smoother cross-chunk transitions.

g_l is differentiable at $\mathbf{h}_l$, then for a small residual $\mathbf{u}_l = \mathcal{Z}_{\phi,l}(\mathbf{c}_t)$, it holds that:

$$\widehat{a}_t^{(d)} = g_l(\mathbf{h}_l + \mathbf{u}_l) \approx g_l(\mathbf{h}_l) + J_l \mathbf{u}_l, \tag{12}$$

where J_l is the Jacobian of g_l at $\mathbf{h}_l$. Equation (12) provides a local intuition: to first order, ACNet only needs to produce a residual direction whose downstream effect matches the desired boundary correction, rather than reconstructing the entire chunk from scratch. This supports treating asynchronous delay as local boundary conditioning rather than full visuomotor remapping. In the 8-layer DiT-style action expert of Evo-1 [18], this design is instantiated by injecting only the final block, i.e., $\mathcal{S} = \{L\}$. Later layers are closest to the action output and therefore most directly influence the handoff boundary; Sec. V-F validates this choice empirically.

C. Training Objective

Because ACNet is attached only to the action head, delay robustness can be learned without repeatedly forwarding the full backbone for every sampled delay. For a fixed observation-instruction pair, the visual-language latent produced by $\mathcal{B}_\omega(o_t, l)$ is invariant to the sampled delay. This latent is therefore cached once and reused across multiple delay conditions. For a sampled delay d, the corresponding condition vector is given by:

$$\mathbf{c}_{t,d} = \mathrm{Pool}\Big(\mathcal{E}_\phi(\widetilde{\mathbf{a}}_{t,d}^{\mathrm{delay}})\Big), \tag{13}$$

where $\widetilde{\mathbf{a}}_{t,d}^{\mathrm{delay}}$ is the padded suffix. This strategy increases delay coverage while keeping the dominant visual-language computation amortized. Furthermore, for the flow-matching action expert in Evo-1, the residual pathway makes the velocity predictor delay conditioned:

$$\widehat{\mathbf{v}}_{t,\tau} = v_{\psi,\eta}^{\mathrm{ACNet}}(\mathbf{x}_\tau, \tau, \mathbf{c}_{t,d}), \tag{14}$$

where $\mathbf{x}_\tau$ is the interpolated action state at flow time $\tau > 0$. Relative to the original flow-matching head, the structural change is the additional residual conditioning pathway. It is worth noting that the pretrained backbone and the main action generator remain unchanged.

Moreover, for the flow-matching training, samples are drawn as $\tau \sim \mathrm{Beta}(2,2)$ and $\mathbf{z} \sim \mathcal{U}([-1,1]^{H \times d_a})$, define

$$\mathbf{x}_\tau = (1-\tau)\mathbf{z} + \tau \mathbf{x}_0, \tag{15}$$

where $\mathbf{x}_0 = \mathbf{a}_t^*$ is the target chunk. The training objective is

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{d,\tau,\mathbf{x}_0,\mathbf{z}}\left[\left\|(\mathbf{x}_0 - \mathbf{z}) - v_{\psi,\eta}^{\mathrm{ACNet}}(\mathbf{x}_\tau, \tau, \mathbf{c}_{t,d})\right\|_2^2\right], \tag{16}$$

where the expectation is taken over sampled delays, flow times, target chunks, and initial action samples. This objective instantiates the delayed-control risk in (6) with a flow-matching prediction loss, while ACNet enforces boundary-aware conditioning through the residual pathway in (11).

V. Experiments

A. Evaluation Protocol

ACNet is evaluated on two asynchronous manipulation benchmarks: the Kinetix setup from RTC [4] in the Kinetix simulator [19], and the full Meta-World MT50 suite [20]. All experiments compare four deployment schemes under the same asynchronous execution protocol: **Naïve Async** , which directly stitches consecutive chunks; **RTC** [4], which freezes committed actions and inpaints the remaining suffix; **Training-RTC** [17], which performs delay-conditioned training; and **ACNet**, the proposed delay-aware adapter.

On Kinetix, the RTC evaluation protocol is followed [4]. The RTC and Training-RTC baselines are based on a π_0 policy [11]: to keep training exposure comparable, Naïve Async and RTC share the same backbone trained for 32 epochs, while Training-RTC follows the schedule in [17], with 24 epochs of standard training and 8 additional epochs of delay-conditioned adaptation. For ACNet, the Evo-1 backbone is used [18]: stage 1 is trained for 1 epoch and stage 2 for 24 epochs, after which the backbone is frozen and ACNet together with the terminal pooling layer is trained for 8 epochs. Delays $d \in \{0, 1, 2, 3, 4\}$ are evaluated.

On Meta-World, all 50 MT50 tasks are evaluated. The RTC and Training-RTC baselines use their π_0-based implementations, whereas ACNet uses Evo-1 [18] as the backbone. For the benchmark-wide results in Table II, the prediction horizon is $H = 50$, the execution interval is $e = 25$, and the

TABLE I

KINETIX RESULTS UNDER ASYNCHRONOUS INFERENCE DELAY.

Method	Success rate under delay d					Avg. ($d > 0$)	Params. (%)
	0	1	2	3	4		
Naïve Async	0.89	0.74	0.69	0.55	0.46	0.61	**0**
RTC	**0.91**	0.75	0.80	0.72	0.61	0.72	**0**
Training-RTC	0.89	**0.88**	0.83	**0.79**	**0.70**	**0.80**	100
ACNet	0.90	0.87	**0.84**	0.76	0.68	0.79	∼20

TABLE II

META-WORLD MT50 RESULTS UNDER ASYNCHRONOUS DELAY.

Method	Success rate under delay d				Avg.	Lat.	Freq.
	0	5	10	15			
Naïve Async	0.80	0.71	0.70	0.70	0.70	**73 ms**	**13.6 Hz**
RTC	0.79	0.72	0.72	0.71	0.71	159 ms	6.28 Hz
Training-RTC	0.80	**0.77**	**0.74**	**0.73**	**0.74**	134 ms	7.46 Hz
ACNet	**0.81**	0.76	**0.74**	0.73	0.74	91 ms	11.0 Hz

TABLE III

PERFORMANCE ON SO-ARM101 OVER 10 TRIALS PER TASK.

Method	Put cube into box	Clean table	Overall
Naïve Async	9/10 (90%)	8/10 (80%)	17/20 (85%)
ACNet	10/10 (100%)	10/10 (100%)	20/20 (100%)

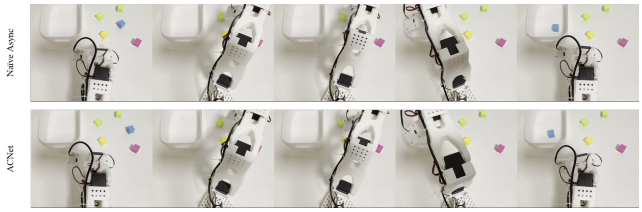

Fig. 5. Representative real-world asynchronous rollouts on the SO-ARM101 platform, for the *put the blue cube into the box*. Snapshots are ordered from left to right. The upper sequence shows Naïve Async and the lower sequence shows ACNet.

evaluated delays are $d \in \{0, 5, 10, 15\}$. The representative jerk plots in Fig. 4 visualize rollouts on *nut-assembly-v3* and *plate-slide-back-v3* with $H = 50$ and $d = 10$. All Meta-World experiments are run on a single RTX 4080 SUPER.

B. Metrics

Task success rate is the primary metric on both benchmarks. To expose the efficiency–robustness trade-off more clearly, trainable parameters are also reported on Kinetix, measured as the percentage of total model parameters updated during adaptation. Frozen parameters are not counted as trainable. End-to-end inference latency and achieved control frequency are also reported on Meta-World. For representative Meta-World rollouts, translational jerk is also reported. For $\mathbf{a}_t = (a_t^x, a_t^y, a_t^z)$, the step-wise jerk of the resulting trajectory is defined as follows:

$$J_t = \frac{1}{3} \sum_{q \in \{x,y,z\}} \left| a_{t+2}^q - 2a_{t+1}^q + a_t^q \right|, \tag{17}$$

with lower values indicating smoother action evolution.

C. Quantitative Results

Table I reports Kinetix results. Averaged over delayed settings, ACNet reaches 0.79 success, compared with 0.72 for RTC and 0.61 for Naïve Async , while remaining close to Training-RTC at 0.80. This robustness is obtained while training only around 20% of all model parameters, whereas Training-RTC updates 100%. Thus, although the average success gap between ACNet and Training-RTC is small, ACNet achieves comparable robustness with an around 80% lower trainable-parameter fraction. In Table I, Params. (%) denotes the percentage of total model parameters updated during adaptation. Table II shows the same trend on Meta-World MT50. ACNet achieves 0.74 average success, exceeding Naïve Async and RTC and matching Training-RTC, while reducing latency to 91 ms versus 159 ms for RTC and 134 ms for Training-RTC. Its achieved control frequency is 11.0 Hz, compared with 6.28 Hz and 7.46 Hz for those two baselines. This latency advantage is mainly due to the underlying model choice rather than the ACNet side branch being intrinsically faster than delay-conditioned training: RTC and Training-RTC are based on π_0, whereas ACNet is built on the lightweight Evo-1 backbone. On the same backbone, the proposed ACNet would add a small encoder and projection overhead to a standard forward pass. As shown in the above results, this overhead is outweighed by the lower end-to-end runtime of Evo-1.

D. Real-World Experiments

ACNet is further evaluated on a real tabletop manipulation platform built on the SO-ARM101 robot arm with a target box, colored cubes, and a tabletop vacuum. The real-world dataset contains 50 training rollouts. The model is optimized for 10 epochs and evaluated on *put the blue cube into the box* and *clean the table* over 10 trials per task under the same asynchronous protocol as simulation. The quantitative results are summarized in Table III. ACNet achieves 20/20 total successes across the two tasks, whereas Naïve Async achieves 17/20. The qualitative rollouts in Fig. 5 show the same pattern: Naïve Async has larger oscillations near chunk transitions, while ACNet maintains smoother transfer and more stable contact during cleaning.

E. Trajectory Smoothness Analysis

To examine motion quality beyond success rate, representative rollouts from *nut-assembly-v3* and *plate-slide-back-v3* with $H = 50$ and $d = 10$ are analyzed. The jerk traces in Fig. 4 show that ACNet maintains a flatter profile around delayed intervals and chunk replacement events. In both tasks, Naïve Async exhibits stronger transition-induced jitter, whereas ACNet produces smoother cross-chunk continuation, consistent with the success gains in Table II.

F. Ablation Study

The ablation study examines which design choices are responsible for the performance gains of ACNet. All ablations are conducted on Meta-World MT50 using the Evo-1

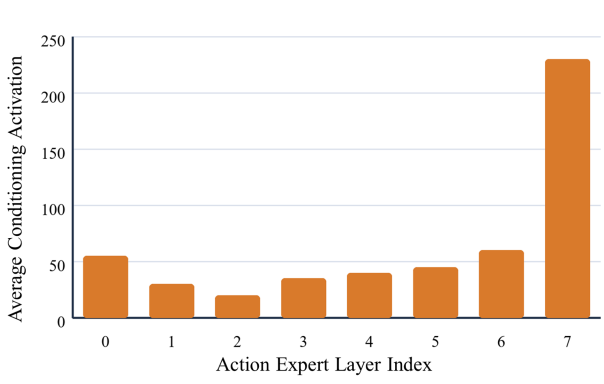

Fig. 6. Average layer-update magnitude across the 8-layer Evo-1 action expert over $K = 1000$ flow-matching integration steps.

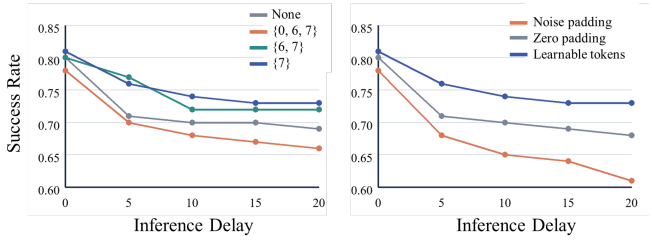

Fig. 7. Ablations on ACNet design choices in Meta-World MT50 with the Evo-1 backbone. **Left**: Conditioning only the final action-expert block yields the best robustness across delays. **Right**: Learnable tokens consistently outperform noise padding and zero padding.

backbone. Two questions are considered: *where* the delay-conditioned residual should be injected, and *how* the unobserved future portion of the delay action should be represented. To determine where delay-aware conditioning is most effective, the mean token-update magnitude of each transformer block is recorded over $K = 1000$ flow-matching steps. Fig. 6 shows that the final block has the largest activity, indicating that late layers contribute most to the final refinement of the predicted chunk. The injection ablation in Fig. 7(a) confirms this choice: final-block conditioning is best across delays, while early-layer injection, especially at layer 0, degrades robustness. This supports using the delay action as a local boundary cue rather than an early perturbation to the task representation. Fig. 7(b) shows that learnable padding tokens outperform zero and random-noise padding at every tested delay. The result supports the minimal-intervention design of ACNet: represent the delay action unambiguously and inject it as a late residual correction close to the action output.

VI. Conclusion

This paper presented Action ControlNet (ACNet), a lightweight delay-aware adapter for asynchronous deployment of chunked vision-language-action policies. By conditioning a largely frozen action head on the executed motion suffix, ACNet reduces handoff mismatch under inference delay while preserving the pretrained perception-language backbone. Experiments on Kinetix, Meta-World MT50, and a real SO-ARM101 platform show that ACNet improves delayed-control robustness and trajectory smoothness with substantially lower adaptation cost than full delay-conditioned retraining. Future work will extend this boundary-conditioning framework to richer delay signals, larger and time-varying delays, and broader classes of real-time control policies.

References

[1] B. Zitkovich *et al.*, "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control," in *Proc. Conf. Robot Learning (CoRL)*, 2023, pp. 2165–2183.

[2] C. Chi *et al.*, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," in *Proc. Robotics: Science and Systems (RSS)*, 2023.

[3] Y. Lipman *et al.*, "Flow Matching for Generative Modeling," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.

[4] K. Black, M. Y. Galliker, and S. Levine, "Real-Time Execution of Action Chunking Flow Policies," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2025.

[5] L. Zhang, A. Rao, and M. Agrawala, "Adding Conditional Control to Text-to-Image Diffusion Models," in *Proc. IEEE/CVF Int. Conf. Computer Vision (ICCV)*, 2023, pp. 3836–3847.

[6] M. Guo and M. Bürger, "Geometric Task Networks: Learning Efficient and Explainable Skill Coordination for Object Manipulation," *IEEE Transactions on Robotics*, vol. 38, no. 3, pp. 1723–1734, 2022.

[7] A. T. Le, M. Guo, N. van Duijkeren, L. Rozo, R. Krug, A. G. Kupcsik, and M. Bürger, "Learning Forceful Manipulation Skills from Multi-Modal Human Demonstrations," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2021, pp. 7770–7777.

[8] A. Brohan *et al.*, "RT-1: Robotics Transformer for Real-World Control at Scale," in *Proc. Robotics: Science and Systems (RSS)*, 2023.

[9] Octo Model Team *et al.*, "Octo: An Open-Source Generalist Robot Policy," in *Proc. Robotics: Science and Systems (RSS)*, 2024.

[10] M. Kim *et al.*, "OpenVLA: An Open-Source Vision-Language-Action Model," in *Proc. Conf. Robot Learning (CoRL)*, 2025, pp. 2679–2713.

[11] K. Black *et al.*, "π_0: A Vision-Language-Action Flow Model for General Robot Control," in *Proc. Robotics: Science and Systems (RSS)*, 2025.

[12] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware," in *Proc. Robotics: Science and Systems (RSS)*, 2023.

[13] Y. Chebotar *et al.*, "Q-Transformer: Scalable Offline Reinforcement Learning via Autoregressive Q-Functions," in *Proc. Conf. Robot Learning (CoRL)*, 2023, pp. 3909–3928.

[14] M. Shukor *et al.*, "SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics," 2025, arXiv preprint arXiv:2506.01844.

[15] K. Sendai, M. Alvarez, T. Matsushima, Y. Matsuo, and Y. Iwasawa, "Leave No Observation Behind: Real-Time Correction for VLA Action Chunks," 2025, arXiv preprint arXiv:2509.23224.

[16] J. Tang, Y. Sun, Y. Zhao, S. Yang, Y. Lin, Z. Zhang, J. Hou, Y. Lu, Z. Liu, and S. Han, "VLASH: Real-Time VLAs via Future-State-Aware Asynchronous Inference," 2025, arXiv preprint arXiv:2512.01031.

[17] K. Black, A. Z. Ren, M. Equi, and S. Levine, "Training-Time Action Conditioning for Efficient Real-Time Chunking," 2025, arXiv preprint arXiv:2512.05964.

[18] T. Lin, Y. Zhong, Y. Du, J. Zhang, J. Liu, Y. Chen, E. Gu, Z. Liu, H. Cai, Y. Zou, L. Zou, Z. Zhou, G. Li, and B. Zhao, "Evo-1: Lightweight Vision-Language-Action Model with Preserved Semantic Alignment," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2026, pp. 13 397–13 406.

[19] M. Matthews, M. Beukman, C. Lu, and J. N. Foerster, "Kinetix: Investigating the Training of General Agents through Open-Ended Physics-Based Control Tasks," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2025.

[20] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning," in *Proc. Conf. Robot Learning (CoRL)*, 2020, pp. 1094–1100.