WHEN TO SWITCH: RELIABLE ACTION-CHUNK EXTENSION FOR VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models serve as unified policies for robotic manipulation, yet their expensive inference forces robots to pause between policy calls, resulting in stop-and-go execution that interrupts smooth motion and prolongs task completion. Extending the action chunk reduces policy calls and hence these pauses, but predicting farther into the future makes long-chunk execution unreliable. To understand where this unreliability arises, we analyze action errors within long chunks and find that they concentrate around transitions between manipulation subskills, growing sharply with chunk length. This suggests the importance of *transition timing*, *i.e.*, when to switch subskills within a chunk. Motivated by this observation, we introduce RACE (**R**eliable **A**ction-**C**hunk **E**xtension), a framework that predicts the transition timing from an auxiliary one-step denoising pass and conditions action generation on it. By learning and conditioning on transition timing, RACE reduces errors at subskill transitions and enables reliable execution of longer action chunks. Across simulation benchmarks, RACE outperforms fine-tuning at the same chunk length; with $2\times$ longer chunks, it surpasses recent state-of-the-art and efficient VLAs in success rate, and with $4\times$ longer chunks, it remains competitive. On a real robot, RACE uses $4\times$ longer chunks, which reduces the idle time caused by stop-and-go execution by about $5\times$, while achieving a higher success rate than fine-tuning with the same chunk length. Code and a real-robot demo are available on Anonymous Github.

1 INTRODUCTION

Vision-language-action (VLA) models (Brohan et al., 2023; Octo Model Team et al., 2024; Li et al., 2024; Kim et al., 2024; Black et al., 2024), such as $\pi_{0.5}$ (Physical Intelligence et al., 2025) and GR00T (NVIDIA et al., 2025), serve as unified policies for diverse robotic manipulation tasks by generating actions on top of large-scale vision-language models. However, their large scale makes each policy inference computationally expensive. Consequently, the robot often finishes executing its current actions before the next ones are ready, forcing repeated pauses between policy calls that disrupt continuous robot operation, *i.e.*, stop-and-go execution (Black et al., 2025; Tang et al., 2026). Over hundreds of control steps, these pauses accumulate into *idle time*, which prolongs task completion and interrupts smooth robot motion, hindering the real-world deployment of VLAs.

Action chunking (Zhao et al., 2023; Chi et al., 2023; Black et al., 2024) mitigates these pauses by predicting multiple consecutive actions per policy call, keeping the robot in motion longer between policy inferences. Extending this chunk (Wang et al., 2026c) further reduces the number of policy calls and the time lost to these pauses, making robot execution more continuous. Moderately longer chunks even improve task success, which has been attributed to temporal consistency (Liu et al., 2025a) or implicit ensembling (Lazzati et al., 2026). Pushing the chunk further, however, forces the policy to predict farther into the future and to execute longer without intermediate feedback (*i.e.*, open-loop execution). Beyond a certain length, this makes execution unreliable and success rates decline (Wang et al., 2026a; Liang et al., 2026) (Fig. 1).

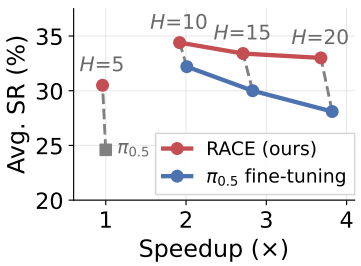
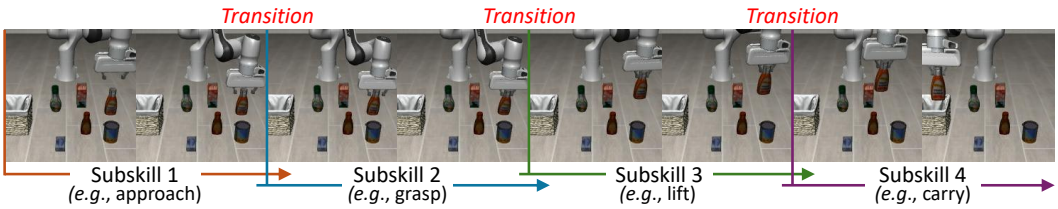
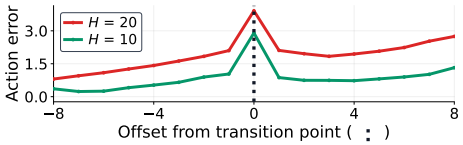


Figure 1: Effect of chunk length H on VLABench, with per-episode inference speedup over $\pi_{0.5}$ (Appendix E.2).

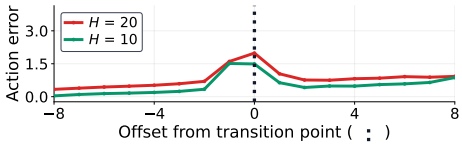
Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping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



(a) Transition between manipulation subskills



(b) Transition-localized errors, $\pi_{0.5}$



(c) Transition-localized errors, RACE (ours)

Figure 2: Action prediction errors around subskill transitions: (a) an example of transitions between manipulation subskills; (b) action errors spike at the transition point within a chunk, and these spikes grow with the chunk length H; and (c) RACE reduces these error spikes across chunk lengths H.

To better understand why long-chunk execution becomes unreliable, we examine how action errors are distributed within a chunk. Besides the gradual error growth expected from open-loop execution, we also observe pronounced error spikes around *transition points*, the steps at which the robot switches from one manipulation subskill (*e.g.*, approaching, grasping, or lifting) to the next (Fig. 2a). Since demonstrations do not annotate subskills, we detect these points as abrupt changes in action dynamics using PELT (Killick et al., 2012), following prior work (Jia et al., 2024; Wang et al., 2025) (Appendix A). The error spikes at these transition points grow substantially with chunk length (Fig. 2b; measured as in Appendix E.4). As the chunk grows longer, the policy must anticipate these subskill switches farther in advance without intermediate feedback, making these switches increasingly difficult to predict. This suggests that *transition timing*, *i.e.*, when to switch subskills within a chunk, is an important challenge in long-chunk execution.

Motivated by this observation, we introduce RACE (**R**eliable **A**ction-**C**hunk **E**xtension), a framework that enables VLAs to reliably execute longer action chunks by learning to predict transition timing and conditioning action generation on it. Learning to predict transition timing makes the action expert aware of upcoming transitions, and conditioning on the predicted timing further informs every denoising step of where transitions are expected within the chunk, together reducing errors around transitions. To achieve this, RACE proceeds in two stages: 1) **Transition-timing Prediction via One-step Denoising**, which estimates a transition-timing prior from an auxiliary one-step denoising pass and the cached VLM features, supervised by transition labels automatically extracted from demonstrations; and 2) **Transition-conditioned Action Generation**, which injects the prior into every denoising step through token-wise modulation of the action expert, with learnable gates controlling its strength per step. Since error spikes appear at transitions at every chunk length (Fig. 2b), RACE reduces these errors across chunk lengths (Fig. 2c), improving task success not only for longer chunks but at each chunk length (Fig. 1).

In our experiments, RACE outperforms fine-tuning at the same chunk length on manipulation benchmarks; with $2\times$ longer chunks, it achieves higher success rates than recent state-of-the-art VLAs and efficient VLA approaches, and with $4\times$ longer chunks, it remains competitive with them. On a real robot, using $4\times$ longer chunks reduces idle time by about $5\times$, and RACE achieves a higher success rate than fine-tuning with the same chunk length.

Our contributions are summarized as follows:

- We identify transitions between manipulation subskills as where action errors concentrate when extending action chunks, suggesting the importance of *transition timing* within a chunk.
- We introduce RACE, a framework for **R**eliable **A**ction-**C**hunk **E**xtension that learns to predict transition timing from an auxiliary one-step denoising pass, making the action expert transition-aware, and conditions generation on it via token-wise modulation with per-step learnable gates.
- Experiments show that RACE achieves higher success rates than recent state-of-the-art and efficient VLA approaches with $2\times$ longer action chunks and remains competitive with $4\times$ longer chunks, and enables longer chunks that reduce idle time on a real robot.

2 Related Work

Vision-Language-Action Models. VLAs adapt pretrained vision-language models to robot control using diverse robot demonstrations (Wang et al., 2026d). Early models such as RT-2 (Brohan et al., 2023) and OpenVLA (Kim et al., 2024) generate discrete action tokens autoregressively, while later models predict continuous action chunks through parallel decoding (Kim et al., 2025) or flow-matching action experts (Physical Intelligence et al., 2025; NVIDIA et al., 2025). RACE reduces the pauses caused by their inference latency by making longer action chunks reliable.

Action Chunking and Asynchronous Execution. Action chunking (Zhao et al., 2023; Chi et al., 2023) predicts multiple future actions per policy call, but later actions in a longer chunk become unreliable without new observations. *Adaptive chunking methods* (Liang et al., 2026; Wang et al., 2026a; Feng et al., 2026a) therefore execute only the reliable prefix of each chunk. PACE (Nie et al., 2026) shortens execution around transitions detected from low-speed valleys, whereas RACE makes those actions reliable so that the whole chunk can be executed. Closest to our work, PolicyTrim (Wang et al., 2026c) extends the chunk through reinforcement learning at a substantial training cost, whereas RACE requires only lightweight post-training of the action expert (Appendix B.7). *Asynchronous execution* (Black et al., 2025; Tang et al., 2026; Wang et al., 2026b) hides inference latency by generating the next chunk during execution, but some idle time remains when inference takes longer than executing a chunk; longer chunks can further reduce it (Sec. 4.5), so RACE is complementary to asynchronous execution.

Efficient VLA Inference. Another line of work reduces the cost of each policy call by pruning visual tokens (Liu et al., 2025b; Ma et al., 2026; Feng et al., 2026b) or reusing computation across consecutive calls (Xu et al., 2025; Liu et al., 2026b). These methods leave the number of policy calls unchanged, whereas RACE reduces it; the two are orthogonal and can be combined.

Subskill Structure in Policy Learning. Demonstrations are often decomposed into subskills, through either change-point detection on action trajectories (Killick et al., 2012; Jia et al., 2024) or manual stage annotation (Wu et al., 2025; Buamanee et al., 2026). Prior work injects this structure into the policy as a prediction target, by predicting the goal state of each subskill (Jia et al., 2024); as an input, by conditioning on the inferred stage or within-subtask progress (Wu et al., 2025; Buamanee et al., 2026); or as an auxiliary output, by generating a progress sequence jointly with actions (Liu et al., 2026a). RACE instead supervises the action expert to predict the transition timing within each chunk, encouraging transition-aware action representations, and further conditions action generation on the predicted timing.

3 RACE: Reliable Action-Chunk Extension

In this section, we present RACE, a framework that enables VLAs to reliably execute longer action chunks (Fig. 3). We first give an overview of how RACE augments flow-matching action generation with a transition-timing prior (Sec. 3.1), and then describe how this prior is predicted from an auxiliary one-step denoising pass (Sec. 3.2) and how it conditions action generation (Sec. 3.3).

3.1 Overview of RACE

Our framework builds upon a flow-matching VLA, such as $\pi_{0.5}$ (Physical Intelligence et al., 2025), which consists of a vision-language model (VLM) and an action expert. At each policy call t, the VLM first encodes the current visual observation $\mathbf{o}_t$ and the language instruction $\boldsymbol{\ell}$ into context cache features $\mathbf{z}_t = \mathrm{VLM}(\mathbf{o}_t, \boldsymbol{\ell})$. Conditioned on $\mathbf{z}_t$, the action expert generates an action chunk $\mathbf{A}_t = (\mathbf{a}_t, \ldots, \mathbf{a}_{t+H-1})$ of length H by iteratively transforming a sequence of Gaussian noise $\mathbf{A}_t^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ over K denoising steps: $\mathbf{A}_t^k = \mathbf{A}_t^{k-1} + \frac{1}{K} v_\theta(\mathbf{A}_t^{k-1}, \mathbf{z}_t)$ for $k = 1, \ldots, K$, where v_θ is the velocity field predicted by the action expert, which also takes the flow time $(k-1)/K$ as input (omitted for brevity), and $\mathbf{A}_t^k$ denotes the intermediate chunk after the k-th denoising step, with the final action chunk $\mathbf{A}_t = \mathbf{A}_t^K$.

RACE augments this process with a transition-timing prior $\hat{\mathbf{p}}_t$, an estimate of where subskill transitions occur within the chunk, and conditions every denoising step on it, as follows:

$$\mathbf{A}_t^k = \mathbf{A}_t^{k-1} + \frac{1}{K} v_\theta\big(\mathbf{A}_t^{k-1}, \mathbf{z}_t, \alpha^k \hat{\mathbf{p}}_t\big), \qquad k = 1, \ldots, K, \tag{1}$$

where $\alpha^k \in (0, 1)$ is a learnable per-step gate with sigmoid activation that controls how strongly the prior is injected at each denoising step. During training, the flow time is sampled continuously as in $\pi_{0.5}$ rather than at the K inference steps, so we use the gate α^k of the step whose interval $[\frac{k-1}{K}, \frac{k}{K})$

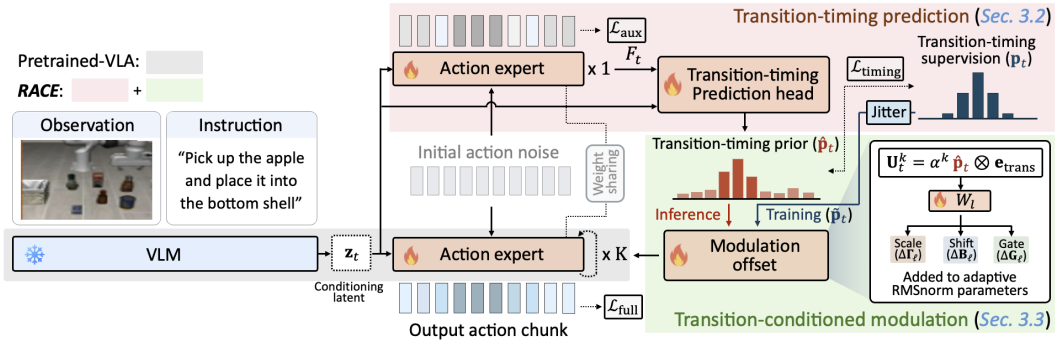


Figure 3: Overview of RACE. Given an observation and instruction, the frozen VLM extracts features cached for both passes. RACE predicts a transition-timing prior via an auxiliary one-step denoising pass (Sec. 3.2), then restarts denoising from the same noise, conditioned on it (Sec. 3.3).

contains it. RACE thus runs three steps: (1) an auxiliary one-step pass from $\mathbf{A}_t^0$ without the prior yields action-expert features; (2) a head estimates $\hat{\mathbf{p}}_t$ from them and $\mathbf{z}_t$ (Sec. 3.2); and (3) the full denoising restarts from the same $\mathbf{A}_t^0$, conditioning every step on $\hat{\mathbf{p}}_t$ (Sec. 3.3). During training, the jittered target $\tilde{\mathbf{p}}_t$ replaces $\hat{\mathbf{p}}_t$ in step (3), while the head is supervised with the target $\mathbf{p}_t$ (Sec. 3.3). Only the action expert and the added modules are trained with the VLM frozen, and inference adds only the auxiliary pass and the head to each policy call, both reusing the cached VLM features.

3.2 Transition-timing Prediction via One-step Denoising

To condition action generation on a transition-timing prior $\hat{\mathbf{p}}_t$, RACE predicts this prior from two sources of information with a learnable transition-timing prediction head: (1) **the cached VLM features** $\mathbf{z}_t$, computed in Sec. 3.1, which encode contextual information about the current observation and instruction. (2) **hidden states from the action expert**, extracted through an auxiliary one-step denoising pass starting from initial noise $\mathbf{A}_t^0$. While the VLM features provide contextual cues, the auxiliary hidden states can offer preliminary motion information aligned with each action position in the chunk. We next describe how the head predicts $\hat{\mathbf{p}}_t$ from these two sources.

Auxiliary One-step Denoising Pass. Before the full denoising process of Eq. (1), RACE performs an auxiliary one-step denoising pass on the initial noise sample $\mathbf{A}_t^0$, computing the velocity $\mathbf{v}_t^{\text{aux}} = v_\theta\left(\mathbf{A}_t^0, \mathbf{z}_t\right)$ without the prior. This pass yields the action expert's final-layer hidden states:

$$\mathbf{F}_t = (\mathbf{f}_t, \ldots, \mathbf{f}_{t+H-1}) \in \mathbb{R}^{H \times d_a}, \tag{2}$$

where d_a denotes the hidden dimension of the action expert, and $\mathbf{f}_{t+j}$ is the final-layer hidden state corresponding to action $\mathbf{a}_{t+j}$ in the chunk. This pass yields the action expert's coarse estimate of the upcoming trajectory given the observation, aligned with each action position in the chunk. We also train this pass with the flow-matching objective, so that it remains a valid denoising step with informative hidden states. The auxiliary pass is used only to obtain $\mathbf{F}_t$ and its velocity for $\mathcal{L}_{\text{aux}}$.

Transition-timing Prediction Head. The prediction head estimates the transition-timing prior $\hat{\mathbf{p}}_t$ from the final-layer action-token features $\mathbf{F}_t$ and the cached final-layer VLM features $\hat{\mathbf{z}}_t$ in $\mathbf{z}_t$. It applies cross-attention between these features, followed by self-attention over the action tokens, a linear projection $\mathbf{w}_p \in \mathbb{R}^{d_z \times 1}$, and a sigmoid function $\sigma(\cdot)$:

$$\begin{aligned} &\hat{\mathbf{p}}_t = \sigma\left(\widetilde{\mathbf{F}}_t \mathbf{w}_p\right) \in [0,1]^H, \quad \widetilde{\mathbf{F}}_t = \bar{\mathbf{F}}_t + \text{SelfAttn}(\bar{\mathbf{F}}_t) \quad \text{where,} \\ &\bar{\mathbf{F}}_t = \hat{\mathbf{F}}_t + \text{CrossAttn}\left(\hat{\mathbf{F}}_t, \hat{\mathbf{z}}_t, \hat{\mathbf{z}}_t\right) \in \mathbb{R}^{H \times d_z}, \quad \hat{\mathbf{F}}_t = \mathbf{F}_t \mathbf{W}_f \in \mathbb{R}^{H \times d_z}. \end{aligned} \tag{3}$$

Here, $\mathbf{W}_f \in \mathbb{R}^{d_a \times d_z}$ is the projection matrix, and d_z is the dimension of $\hat{\mathbf{z}}_t$. The resulting transition-timing prior $\hat{\mathbf{p}}_t = (\hat{p}_t, \ldots, \hat{p}_{t+H-1}) \in [0,1]^H$ assigns a score to each action in the chunk, indicating how close it is to a subskill transition. This prior then conditions action generation (Sec. 3.3).

Transition-timing Supervision. We derive the transition-timing target from demonstration actions as change points of PELT-based unsupervised subskill discovery (Jia et al., 2024; Killick et al., 2012) (Appendix A.3). Since a transition spans several steps and the detected points may not exactly match the true transition steps, we convert each transition point into a soft target $\mathbf{p}_t$ that peaks at the point and decays with distance (Appendix A.4; Fig. 8). The head is then optimized with a binary

cross-entropy loss between the predicted prior $\hat{\mathbf{p}}_t$ and a target $\mathbf{p}_t = (p_t, \ldots, p_{t+H-1}) \in [0,1]^H$:

$$\mathcal{L}_{\text{timing}} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \text{BCE}(\hat{\mathbf{p}}_t,\ \mathbf{p}_t)\,, \tag{4}$$

where $\mathcal{T}$ denotes the set of chunk start (*i.e.*, policy call) steps across the demonstration trajectories. Since the head reads the action-expert features from the auxiliary pass, $\mathcal{L}_{\text{timing}}$ is also back-propagated into the action expert, encouraging its representations to capture upcoming transitions.

3.3 Transition-conditioned Action Generation

We now describe how RACE conditions the action expert on the transition-timing prior $\hat{\mathbf{p}}_t$ during the full denoising process in Eq. (1) of Sec. 3.1. At each denoising step, the prior is injected through the action expert's adaptive RMSNorm layers (Physical Intelligence et al., 2025), which follow the adaptive normalization of DiT (Peebles & Xie, 2023), as described below.

Transition-conditioned Modulation. To inject the timing prior $\hat{\mathbf{p}}_t \in [0,1]^H$ into the action expert, we first construct token-wise transition features $\mathbf{U}_t^k \in \mathbb{R}^{H \times d_a}$ as:

$$\mathbf{U}_t^k = \alpha^k\, \hat{\mathbf{p}}_t \otimes \mathbf{e}_{\text{trans}} \in \mathbb{R}^{H \times d_a}, \tag{5}$$

where $\mathbf{e}_{\text{trans}} \in \mathbb{R}^{d_a}$ is a randomly initialized learnable transition embedding, $\otimes$ denotes the outer product, and α^k is the learnable denoising-step gate from Eq. (1), which controls the strength of transition-prior conditioning at each step k. We then map these features to token-wise offsets $[\Delta\boldsymbol{\Gamma}_\ell, \Delta\mathbf{B}_\ell, \Delta\mathbf{G}_\ell]$ of the scale, shift, and gate parameters of adaptive RMSNorm layer ℓ, and add them to the original modulation parameters, $\boldsymbol{\gamma}_\ell, \boldsymbol{\beta}_\ell, \mathbf{g}_\ell \in \mathbb{R}^{d_a}$, of the pretrained action expert:

$$\begin{aligned} [\Delta\boldsymbol{\Gamma}_\ell, \Delta\mathbf{B}_\ell, \Delta\mathbf{G}_\ell] &= \mathbf{U}_t^k \mathbf{W}_\ell \ \in \ \mathbb{R}^{H \times 3d_a}, \\ [\boldsymbol{\Gamma}_\ell, \mathbf{B}_\ell, \mathbf{G}_\ell] &= \mathbf{1}_H \left[\boldsymbol{\gamma}_\ell^\top, \boldsymbol{\beta}_\ell^\top, \mathbf{g}_\ell^\top\right] + [\Delta\boldsymbol{\Gamma}_\ell, \Delta\mathbf{B}_\ell, \Delta\mathbf{G}_\ell]\,, \end{aligned} \tag{6}$$

where $\mathbf{1}_H \in \mathbb{R}^{H \times 1}$ is the all-ones vector that copies the original parameters to every action token, and $\mathbf{W}_\ell \in \mathbb{R}^{d_a \times 3d_a}$ projects the transition features into token-wise offsets. We initialize $\mathbf{W}_\ell$ to zero, so that training starts from the original modulation, and $\mathbf{W}_\ell$ has no bias, so that a zero prior leaves the action expert unmodified. These token-wise parameters $[\boldsymbol{\Gamma}_\ell, \mathbf{B}_\ell, \mathbf{G}_\ell]$ then modulate both the attention and feed-forward modules at each adaptive RMSNorm layer ℓ of the action expert:

$$\mathbf{X}_{\ell+1} = \mathbf{X}_\ell + \mathbf{G}_\ell \odot \text{Module}_\ell(\tilde{\mathbf{X}}_\ell), \quad \tilde{\mathbf{X}}_\ell = (1 + \boldsymbol{\Gamma}_\ell) \odot \text{RMSNorm}(\mathbf{X}_\ell) + \mathbf{B}_\ell, \tag{7}$$

where Module_ℓ denotes either an attention or feed-forward module at layer ℓ, $\mathbf{X}_\ell \in \mathbb{R}^{H \times d_a}$ denotes the hidden states of the H action tokens at layer ℓ, and $\odot$ denotes the element-wise product. This conditioning mechanism allows the action expert to adapt its denoising behavior based on the predicted transition-timing prior $\hat{\mathbf{p}}_t$.

Training Objective. Our framework is trained with the standard flow-matching objective (Lipman et al., 2023) for both the full and auxiliary denoising passes, together with the transition-timing prediction loss in Eq. (4): $\mathcal{L} = \mathcal{L}_{\text{full}} + \lambda_{\text{aux}}\mathcal{L}_{\text{aux}} + \lambda_{\text{timing}}\mathcal{L}_{\text{timing}}$. Here, both losses regress the velocity field toward the direction from the noise sample to the ground-truth chunk: $\mathcal{L}_{\text{full}}$ at a sampled flow time, and $\mathcal{L}_{\text{aux}}$ at the initial noise without modulation, *i.e.*, on the auxiliary pass. The coefficients λ_{aux} and λ_{timing} balance the two additional terms. During training, the action expert is conditioned on the jittered target $\tilde{\mathbf{p}}_t$ rather than the prediction $\hat{\mathbf{p}}_t$ (*i.e.*, teacher forcing and jittering; Appendix C.4), while at inference it is conditioned on the predicted $\hat{\mathbf{p}}_t$, as in Eq. (1).

4 Experiments

4.1 Experimental Setup

Dataset and Metric. For simulation experiments, we evaluate our framework on three manipulation benchmarks: VLABench (Zhang et al., 2025), RoboCasa-H50 (Nasiriany et al., 2024), and LIBERO (Liu et al., 2023), with the LIBERO results reported in Appendix B.1. We follow the standard evaluation protocols of each benchmark, with the dataset and protocol details provided in Appendix E.5. Throughout, H_{exec} denotes the number of actions executed per policy call. For each H_{exec}, we post-train RACE on top of the $\pi_{0.5}$ fine-tuned for that benchmark, together with a fine-tuned $\pi_{0.5}$ variant trained with the same recipe; both predict and execute a full chunk of H_{exec} actions. The original $\pi_{0.5}$ instead predicts 10 actions and executes only the first 5, following its official protocol. As evaluation metrics, we measure the task success rate and the speedup relative to the baseline $\pi_{0.5}$, measured as the reduction in cumulative policy-inference time per episode, excluding simulator execution time (Appendix E.2). The real-world setup and metrics are in Appendix E.1.

Table 1: Comparison on VLABench. We report the success rate (SR) and the progress score (PS), the fraction of completed sub-tasks. Spd is the per-episode inference speedup relative to the $\pi_{0.5}$ baseline (Appendix E.2). H_{exec} is the number of actions executed per policy call, and † denotes a model fine-tuned to execute that many actions. ‡ indicates the average execution length, since adaptive chunking methods truncate each chunk adaptively. Results for a wider range of chunk lengths are in Appendix B.2.

Method	H_{exec}	Spd↑	In-dist.		Category		Common.		Instruct.		Texture		Avg.	
			SR	PS	SR	PS	SR	PS	SR	PS	SR	PS	SR	PS
$\pi_{0.5}$ Baseline	5	1.00×	40.4	56.7	21.4	35.6	17.0	33.7	18.0	35.9	26.0	42.1	24.6	40.8
RACE (ours)	5	0.96×	51.6	68.2	25.0	38.1	26.0	40.0	20.8	37.6	29.0	45.3	30.5	45.8
Action-Chunk Extension														
$\pi_{0.5}$ Fine-tuning	10	2.01×	53.0	67.8	30.2	41.8	24.0	40.5	22.0	40.4	31.8	49.0	32.2	47.9
	15	2.83×	51.4	65.6	26.6	39.6	23.4	40.3	19.0	37.1	29.4	46.3	30.0	45.8
	20	3.82×	48.8	65.3	22.8	37.2	20.2	38.2	20.8	39.3	27.8	43.9	28.1	44.8
RACE (ours)	10	1.92×	57.6	70.9	30.4	41.8	23.2	39.0	25.0	42.0	36.0	52.8	34.4	49.3
	15	2.71×	54.2	67.1	28.8	39.4	25.6	41.4	25.2	42.6	33.2	50.0	33.4	48.1
	20	3.68×	54.0	68.2	27.2	38.8	28.2	41.4	21.0	39.4	34.8	51.0	33.0	47.8
Recent State-of-the-Art VLAs														
ACoT-VLA (Zhong et al., 2026)	5	0.62×	46.4	68.0	22.2	40.0	24.4	42.0	20.4	39.2	34.4	55.2	29.6	48.9
	20†	2.36×	49.4	70.8	21.4	39.2	24.4	43.0	28.2	46.0	42.0	65.2	33.1	52.8
Adaptive Chunking Methods														
AutoHorizon (Wang et al., 2026a)	9.45‡	1.89×	43.8	59.5	23.8	37.8	16.4	33.3	18.2	36.3	24.4	40.6	25.3	41.5
PACE (Nie et al., 2026)	8.58‡	1.69×	41.2	58.4	22.8	36.4	20.0	36.8	16.0	35.6	26.8	44.4	25.4	42.3
Other Efficient VLA Methods														
Latent Bridge (Liu et al., 2026b)	5	1.63×	39.2	56.8	23.2	37.0	19.4	36.0	14.8	33.2	28.0	44.8	24.9	41.6
GridS (Feng et al., 2026b)	5	1.24×	46.2	63.4	24.0	38.4	19.8	38.4	13.6	32.0	38.4	55.2	28.4	45.5

Implementation Details. We build our framework on the official $\pi_{0.5}$ (Physical Intelligence et al., 2025) checkpoints fine-tuned for each benchmark, with $K = 10$ denoising steps. For the transition-timing prediction (Sec. 3.2), the cached VLM features and the action-expert hidden states have dimensions $d_z = 256$ and $d_a = 1024$. For the transition-conditioned generation (Sec. 3.3), we initialize every gate α^k to 0.5 and optimize it during training. In the training objective, we set $\lambda_{\text{aux}} = 0.1$ and $\lambda_{\text{timing}} = 0.05$. We train for 40K steps on VLABench and 20K steps on the other benchmarks using AdamW with a learning rate that decays from 5×10^{-5} to 5×10^{-6} following a cosine schedule, and a batch size of 64 on four NVIDIA RTX A6000 GPUs.

4.2 Main Results

Comparison on VLABench. Tab. 1 reports the success rate (SR), the progress score (PS), *i.e.*, the fraction of completed sub-tasks, and the per-episode inference speedup across the five VLABench tracks. At every $H_{\text{exec}} \in \{10, 15, 20\}$, RACE outperforms the fine-tuned $\pi_{0.5}$ executing the same number of actions, and at $H_{\text{exec}} = 20$, it still outperforms the $\pi_{0.5}$ baseline while requiring far less inference time per episode. At $H_{\text{exec}} = 5$, RACE also achieves a higher success rate than the official $\pi_{0.5}$ checkpoint. ACoT-VLA (Zhong et al., 2026), a recent state-of-the-art VLA built on $\pi_{0.5}$, reaches a success rate comparable to RACE at $H_{\text{exec}} = 20$ with a higher progress score but a lower speedup, as it runs an additional multi-step denoising process whereas RACE adds a single auxiliary pass (Appendix B.6 and B.7). With $H_{\text{exec}} \geq 10$, RACE also outperforms adaptive chunking methods (Wang et al., 2026a; Nie et al., 2026) and efficient VLA methods (Liu et al., 2026b; Feng et al., 2026b) in both success rate and speedup. Results for longer chunks are in Appendix B.2.

Comparison on RoboCasa-H50. Tab. 2 reports success rates and per-episode inference speedup on the eight RoboCasa-H50 categories, with the average weighted by the number of tasks per category. RACE outperforms the fine-tuned $\pi_{0.5}$ at every H_{exec}, and at $H_{\text{exec}} = 20$, it still outperforms the $\pi_{0.5}$ baseline while requiring far less inference time per episode. At $H_{\text{exec}} = 5$, RACE also achieves a higher success rate than the official $\pi_{0.5}$ checkpoint. The recent state-of-the-art VLA, ACoT-VLA (Zhong et al., 2026), fine-tuned to execute $H_{\text{exec}} = 20$ actions falls below RACE at the same length, with a lower speedup. With $H_{\text{exec}} \geq 10$, RACE also outperforms efficient VLA methods (Liu et al., 2026b; Feng et al., 2026b) in both success rate and speedup, and adaptive chunking methods (Wang et al., 2026a; Nie et al., 2026) at $H_{\text{exec}} \in \{10, 15\}$, while remaining comparable at $H_{\text{exec}} = 20$ with about twice their speedup.

Table 2: Comparison on RoboCasa-H50. We report the success rate on eight task categories, with the average weighted by the number of tasks per category. Spd is the per-episode inference speedup relative to the $\pi_{0.5}$ baseline. ‡ indicates the average execution length, since adaptive chunking methods truncate each chunk adaptively.

Method	H_{exec}	Spd↑	PnP	Doors	Drawer	Sink	Stove	Coffee	Micro.	Nav.	Avg.
$\pi_{0.5}$ Baseline	5	1.00×	51.5	57.0	95.0	74.7	40.0	65.3	79.0	22.0	60.4
RACE (ours)	5	0.97×	57.2	56.0	93.0	77.3	42.0	68.0	80.0	40.0	63.5
Action-Chunk Extension											
$\pi_{0.5}$ Fine-tuning	10	1.96×	62.2	56.0	94.0	75.3	45.0	65.3	89.0	26.0	65.0
	15	2.94×	60.5	58.0	94.0	68.7	35.0	72.0	91.0	26.0	64.2
	20	3.84×	56.8	61.0	89.0	62.7	38.0	62.7	87.0	18.0	60.8
RACE (ours)	10	1.91×	65.0	61.0	95.0	73.3	42.0	70.0	84.0	34.0	66.8
	15	2.91×	61.2	65.5	95.0	79.3	44.0	69.3	92.0	24.0	67.3
	20	3.77×	62.2	63.5	93.0	68.0	34.0	64.7	84.0	28.0	64.0
Recent State-of-the-Art VLAs											
ACoT-VLA (Zhong et al., 2026)	5	0.70×	58.2	61.5	89.0	74.7	40.0	67.3	92.0	28.0	64.3
	20	2.66×	55.0	58.0	79.0	64.0	23.0	57.3	75.0	38.0	57.1
Adaptive Chunking Methods											
AutoHorizon (Wang et al., 2026a)	9.41‡	1.87×	58.5	58.0	90.0	74.7	34.0	64.7	84.0	42.0	63.0
PACE (Nie et al., 2026)	8.11‡	1.63×	61.3	57.5	91.0	74.0	32.0	67.3	85.0	44.0	64.2
Other Efficient VLA Methods											
Latent Bridge (Liu et al., 2026b)	5	1.63×	56.2	59.0	93.0	67.3	33.0	58.7	74.0	44.0	60.3
GridS (Feng et al., 2026b)	5	1.25×	46.5	50.5	90.0	71.3	26.0	57.3	70.0	46.0	55.1

Table 3: Ablation of each component and training objective. The auxiliary one-step pass is always performed when the transition-timing prediction head is present, since the head reads its features; $\mathcal{L}_{\text{aux}}$ only controls whether this pass is additionally supervised as a denoiser.

Transition-timing Prediction Head (Sec. 3.2)	Transition-conditioned Action Generation (Sec. 3.3)	Training Objectives			Avg. SR
		$\mathcal{L}_{\text{full}}$	$\mathcal{L}_{\text{aux}}$	$\mathcal{L}_{\text{timing}}$	
-	-	✓	-	-	28.1
-	-	✓	✓	-	28.4
✓	-	✓	-	✓	30.6
✓	✓	✓	-	✓	32.4
✓	✓	✓	✓	✓	33.0

Comparison on LIBERO. On LIBERO, RACE outperforms the fine-tuned $\pi_{0.5}$ at every H_{exec}, and PolicyTrim (Wang et al., 2026c) in success rate with $H_{\text{exec}} \in \{10, 15\}$ (Appendix B.1).

4.3 Ablation Study

We conduct ablation studies with an execution chunk length of $H_{\text{exec}} = 20$ on VLABench.

Effect of Each Component and Training Objective. In Tab. 3, we add each component and training objective one at a time, starting from the fine-tuned $\pi_{0.5}$ with only $\mathcal{L}_{\text{full}}$. Adding $\mathcal{L}_{\text{aux}}$ alone, without the prediction head, yields only a small improvement, so the extra denoising objective contributes little by itself. Adding the prediction head with $\mathcal{L}_{\text{timing}}$ raises the success rate, even though the prediction is not injected into generation. Since the head reads the action-expert features from the auxiliary pass, $\mathcal{L}_{\text{timing}}$ back-propagates into the action expert and makes its features transition-aware. Conditioning on the predicted prior improves it further under the same training objectives, showing that timing supervision and timing conditioning provide complementary gains. Finally, $\mathcal{L}_{\text{aux}}$ keeps the auxiliary pass a valid denoiser, which may provide better features for the head.

Ablation within Transition-timing Prediction Head. In Tab. 4, we conduct the ablation study on the transition labels and input information sources for the transition-timing prediction head (Sec. 3.2). **(a) Transition labels** (Tab. 4a). Replacing the PELT labels with the random transitions shrinks the learned modulation offsets by more than an order of magnitude; the model learns to ignore the uninformative prior and falls back to naïve fine-tuning. Speed-minimum labels (Nie et al., 2026) also underperform PELT. Appendix A.5 compares these label sets against the transitions of the demonstration generator, and less accurate labels indeed yield smaller gains. Further results with other PELT settings are reported in Appendix C.1. **(b) Information sources** (Tab. 4b). Using

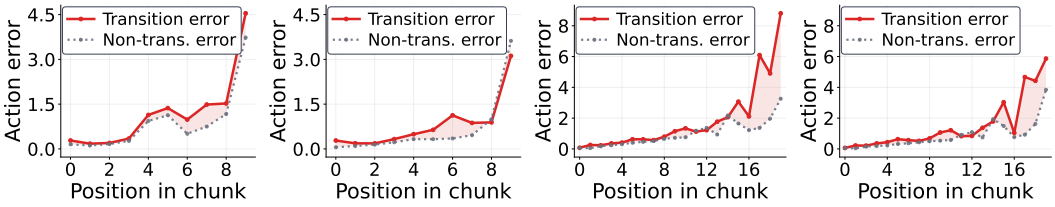
Table 4: Ablation within the transition-timing prediction head (Sec. 3.2). Analyses of transition-label validity and sensitivity to PELT settings are provided in Appendices A.5 and C.1, respectively.

(a) Transition-label sources

Label source	Avg.	Modulation offsets (Sec. 3.3)		
		$\|\Delta\Gamma\|$	$\|\Delta\mathbf{B}\|$	$\|\Delta\mathbf{G}\|$
Random	28.5	0.30	0.35	0.35
Speed minima (Nie et al., 2026)	31.2	5.45	6.37	5.72
PELT (used)	33.0	3.67	4.12	3.81

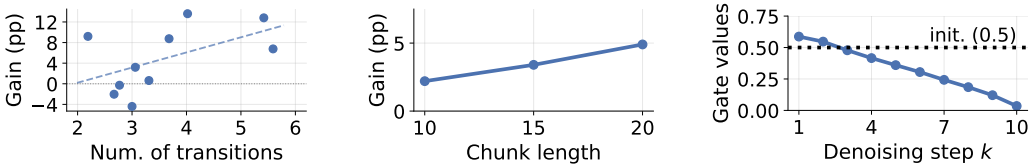
(b) Information sources

Input features for the head (Sec. 3.2)	Avg.
VLM features only	31.6
Action features only	32.5
Both (ours)	33.0



(a) $H_{\text{exec}} = 10$; $\pi_{0.5}$ (b) $H_{\text{exec}} = 10$; ours (c) $H_{\text{exec}} = 20$; $\pi_{0.5}$ (d) $H_{\text{exec}} = 20$; ours

Figure 4: Errors at transition and non-transition actions with $H_{\text{exec}} = 10$ and 20 on VLABench, where $\pi_{0.5}$ is fine-tuned to execute H_{exec} actions per policy call. At each position within a chunk, we compare the error of actions at transition points (transition error) with that of the remaining actions (non-transition error), which we use as a reference for the error growth with prediction horizon.



(a) Gain on # of transitions (b) Gain on chunk lengths (c) Learned gate values α^k

Figure 5: Effect of the transition-timing prior on VLABench. (a) gain of RACE over the fine-tuned $\pi_{0.5}$ as a function of the number of transitions in each task on VLABench, (b) the same gain of RACE at each chunk length, and (c) learned gate values α^k in Eq. (1) across denoising steps.

the VLM features alone performs worst: with the head attached to the VLM, $\mathcal{L}_{\text{timing}}$ never reaches the action expert, so the gain from reshaping its features (Tab. 3) is lost. The action-expert features alone come second, and combining both performs best (further ablations are in Appendix C).

4.4 Understanding the Effectiveness of RACE

Where Errors Arise in Longer Chunks. To examine where errors arise as chunks grow longer, we compare actions predicted from demonstration states with the demonstrated actions on VLABench at $H_{\text{exec}} \in \{10, 20\}$ (Fig. 4; Appendix E.4). We compare errors at transitions with those of non-transition actions at the same chunk position. Both errors grow toward the end of the chunk, as expected from open-loop prediction farther ahead. However, errors at transitions remain generally higher, and this gap widens at $H_{\text{exec}} = 20$. This indicates that transitions pose additional difficulty beyond the growth with prediction horizon. RACE reduces errors at transitions over the fine-tuned $\pi_{0.5}$, particularly near the end of the chunk, while also lowering errors at non-transition actions. For the gripper, whose binary command mainly reflects when it switches, RACE switches within one step of the correct step more often and misses fewer switches than the fine-tuned $\pi_{0.5}$, with a similar false-switch rate (Tab. 27 of Appendix D.4, with transition errors by action component in Tab. 26).

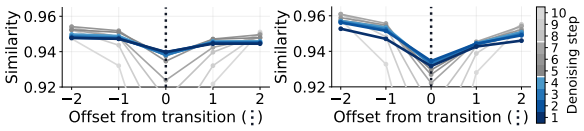
Gains on Transition-rich Tasks. Fig. 5a illustrates the gain of RACE over the fine-tuned $\pi_{0.5}$ at $H_{\text{exec}} = 20$ for ten VLABench tasks. For each task, gains are averaged over five tracks and plotted against the average number of transitions per training demonstration. Although gains vary across tasks, RACE tends to achieve larger improvements on tasks with more subskill transitions. This trend is consistent with our motivation to improve action generation around subskill transitions. The detailed statistics of each task are provided in Appendix A.2.

Gains on Longer Chunks. Fig. 5b shows the gain of RACE over the fine-tuned $\pi_{0.5}$ across chunk lengths on VLABench. The gain is positive at every H_{exec} and grows as the chunk becomes longer.

Since a longer chunk is more likely to span one or more subskill transitions, this pattern indicates that transition timing is an obstacle to chunk extension and that RACE relieves it.

Learned Gates across Denoising Steps. In Fig. 5c, we report the learned per-step gates α^k of Eq. (1), which control how strongly the prior modulates each denoising step. Initialized at the same value, the gates decrease monotonically across denoising steps after training, so the prior modulates the early steps most strongly. This pattern is in line with our analysis in Fig. 6.

Transition Boundaries across Denoising Steps. To examine how RACE changes the action expert's representations, we measure the cosine similarity between the hidden states of adjacent action tokens, aligned at the transition point, at each denoising step in Fig. 6. A dip at the transition indicates that the action expert separates the two neighboring behaviors. In $\pi_{0.5}$, the dip is weak at the early steps and becomes clearer only later. With RACE, the dip is more pronounced from the first step, consistent with the timing supervision and the learned gates (Fig. 5c), which peak early.



(a) $\pi_{0.5}$ (b) RACE (ours)

Figure 6: Adjacent action-token similarity.

Perturbing Transition-timing at Inference. To examine how much the predicted timing contributes at inference, we keep the trained model fixed and replace the prior with zeros or random peaks, or shift it by one step (Tab. 5; VLABench, $H_{\text{exec}} = 20$). A zero prior removes the modulation and lowers the success rate, showing that conditioning contributes at inference. Much of the gain over the fine-tuned $\pi_{0.5}$ remains, consistent with the transition-aware features learned through timing supervision (Tab. 3). A random prior lowers it further. Shifting the predicted prior by a single step also lowers the success rate below the zero-prior setting and moves the generated gripper switches in the same direction (Appendices D.2 and D.3), showing that the model relies on where the prior places transitions, not merely on its presence. Although the predicted prior does not capture every transition (Appendix D.1), it yields the highest success rate among these settings.

Table 5: Replacement and shift of timing prior.

	Timing prior at inference			Prior shift	
	Random	Zero	Predicted (ours)	-1	$+1$
Avg.	23.5	31.8	33.0	30.2	30.7

4.5 Real-World Deployment

Results on a Real Robot. To verify that RACE reliably executes longer action chunks that reduce idle time on a real robot, we fine-tune each policy on 30 demonstrations of a pick-and-place task and evaluate it over 50 trials (details in Appendix E.1; demo videos on Anonymous Github). All methods use asynchronous execution with real-time chunking (Black et al., 2025), generating the next chunk while the current one is executed; each policy thus predicts a chunk twice as long as it executes (*e.g.*, $H = 40$ for $H_{\text{exec}} = 20$). We report the success rate, task-completion time, and idle time averaged over successful trials, *i.e.*, the total time the robot receives no new action command. As shown in Tab. 6, a short chunk still leaves idle time even with asynchronous execution (analyzed in Appendix D.5). Extending the chunk greatly reduces it, since executing more actions per policy call leaves enough time to generate the next chunk; the remaining idle time comes mostly from the first policy call at the start of each episode. Task-completion time decreases only slightly, since asynchronous execution already hides most of the latency and the completion time is dominated by the robot motion itself; the reduced idle time mainly removes pauses that interrupt smooth motion and can shift the robot dynamics away from the demonstrations (Black et al., 2025). At the same chunk length, RACE achieves a higher success rate than fine-tuning.

Table 6: Results on a real robot.

Method	H_{exec}	SR↑	Time↓	Idle↓
$\pi_{0.5}$ Fine-tuning	5	36%	12.6 s	2.11 s
	20	48%	11.6 s	0.40 s
RACE (ours)	20	66%	11.8 s	0.41 s

5 Conclusion

In this work, we study why longer action chunks become unreliable in VLAs, and observe that action errors concentrate at transitions between manipulation subskills and grow with the chunk length. Based on this, we propose RACE, which improves long-chunk execution through transition-aware learning and transition-conditioned generation, resulting in competitive success rates while executing $4\times$ longer action chunks. On a real robot, RACE enables longer chunks that reduce idle time, with a higher success rate than fine-tuning at the same chunk length.

AI USE STATEMENT

In this work, we used generative AI tools to edit the paper for readability, to draft and revise parts of the text, to search for related work, to assist in implementing the method and experiments, and to name the subskills in Fig. 7 for illustration only, which are not used for training or analysis. All code was designed, reviewed, and tested by the authors, and all reported results were verified by the authors. We have not used generative AI tools for generating synthetic datasets, developing theoretical models, formulating or proving mathematical claims, proposing or refining hypotheses, designing the research methodology, translating content, cleaning datasets, qualitative analysis, or interpreting results. We have reviewed all AI-assisted content and take responsibility for the final content of this work, including text, code, and claims produced with the aid of generative AI.

ETHICS STATEMENT

This work uses publicly available simulation benchmarks and demonstrations collected in our laboratory, and does not involve human subjects or personal data. Since longer action chunks reduce how often the robot reacts to new observations, deploying such policies near people requires additional safeguards (Appendix F.2).

REPRODUCIBILITY STATEMENT

To support reproducibility, we release the source code anonymously on Anonymous Github. It includes the training code for RACE and the fine-tuned $\pi_{0.5}$ baseline on VLABench, the transition labels together with the script that computes them using PELT, and instructions for running inference with a trained model. The implementation details are described in Sec. 4.1 and the appendix, including the transition discovery (Appendix A), the dataset and evaluation protocols (Appendix E.5), and the pseudocode for training and inference (Appendix E.6). If the paper is accepted, we will publicly release the full code, including the other benchmarks, together with the trained model weights.

REFERENCES

Anthropic. Claude code. https://claude.com/claude-code, 2025. Accessed: 2026-09.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023.

Thanpimon Buamanee, Masato Kobayashi, and Yuki Uranishi. Bi-HIL: Bilateral control-based multimodal hierarchical imitation learning via subtask-level progress rate and keyframe memory for long-horizon contact-rich robotic manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026. Accepted.

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

Xiangdong Feng, Yuxuan Cheng, Chen Shi, Boyao Han, Yuxuan Yan, Yitong Hong, Zhuotao Tian, and Li Jiang. Denoising tells when to replan: Denoising-variance adaptive chunking for flow-based robot policies. *arXiv preprint arXiv:2606.03847*, 2026a.

Yixu Feng, Zinan Zhao, Yanxiang Ma, Chenghao Xia, Chengbin Du, Yunke Wang, and Chang Xu. See what matters: Differentiable grid sample pruning for generalizable vision-language-action model. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026b.

Zhiwei Jia, Vineet Thumuluri, Fangchen Liu, Linghao Chen, Zhiao Huang, and Hao Su. Chain-of-thought predictive control. In *International Conference on Machine Learning (ICML)*, 2024.

Rebecca Killick, Paul Fearnhead, and Idris A. Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025.

Filippo Lazzati, Kyle Stachowicz, William Chen, Alberto Maria Metelli, Andrew Wagenmaker, and Sergey Levine. Why does action chunking improve behavioral cloning performance in robotic control? In *Conference on Robot Learning (CoRL)*, 2026.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, Hang Li, and Tao Kong. Vision-language foundation models as effective robot imitators. In *International Conference on Learning Representations (ICLR)*, 2024.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, Fangqiang Ding, and Ismini Lourentzou. PALM: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026a.

Yudong Liu, Yuan Li, Zijia Tang, Yuxi Zheng, Yueqian Lin, Qinsi Wang, Yi Li, Shuangjun Liu, Shuai Zhang, Taotao Jing, Dashan Gao, Ning Bi, Jingwei Sun, Yiran Chen, and Hai Li. Latent bridge: Feature delta prediction for efficient dual-system vision-language-action model inference. *arXiv preprint arXiv:2605.02739*, 2026b.

Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via guided test-time sampling. In *International Conference on Learning Representations (ICLR)*, 2025a.

Zhe Liu, Jinghua Hou, Yuxiang Lu, Zhenya Yang, Xianzhe Fan, Junwei Luo, Junyi Li, Ruihua Han, Zhi Hou, and Hengshuang Zhao. StreamPI: Streaming multimodal temporal modeling for vision-language-action models. *arXiv preprint arXiv:2608.26067*, 2026c.

Ziyan Liu, Yeqiu Chen, Hongyi Cai, Tao Lin, Shuo Yang, Zheng Liu, and Bo Zhao. Bridging the semantic-action gap in visual token pruning for efficient VLA inference. *arXiv preprint arXiv:2511.16449*, 2025b.

Shilin Ma, Chubin Zhang, Changyuan Wang, Yuji Wang, Yue Wu, Zixuan Wang, Jingqi Tian, Zheng Zhu, and Yansong Tang. SAFE-Pruner: Semantic attention-guided future-aware token pruning for efficient vision-language-action manipulation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2026.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of everyday tasks for generalist robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

Junnan Nie, Jiayi Li, Jiachen Zhang, Junyi Lao, Chenghao Liu, Tianle Zhang, Liang Lin, and Songfang Huang. PACE: Phase-aware chunk execution for robot policies with action chunking. *arXiv preprint arXiv:2606.00537*, 2026.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

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. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. VLA knows its limits: Adaptive execution horizons for robot policies. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2026a.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Rao Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. In *International Conference on Learning Representations (ICLR)*, 2026b.

Ruiqi Wang, Dezhong Zhao, Ziqin Yuan, Tianyu Shao, Guohua Chen, Dominic Kao, Sungeun Hong, and Byung-Cheol Min. PRIMT: Preference-based reinforcement learning with multimodal feedback and trajectory synthesis from foundation models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Xianghui Wang, Feng Chen, Wenbo Zhang, Hua Yan, Zixuan Wang, Changsheng Li, and Yinjie Lei. PolicyTrim: Boosting intrinsic policy efficiency of vision-language-action models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2026c.

Yuqi Wang, Xinghang Li, Wenxuan Wang, Junbo Zhang, Yingyan Li, Yuntao Chen, Xinlong Wang, and Zhaoxiang Zhang. Unified vision-language-action model. In *International Conference on Learning Representations (ICLR)*, 2026d.

BinXu Wu, TengFei Zhang, Chen Yang, JiaHao Wen, HaoCheng Li, JingTian Ma, Zhen Chen, and JingYuan Wang. SAGE: State-aware guided end-to-end policy for multi-stage sequential tasks via hidden Markov decision process. *arXiv preprint arXiv:2509.19853*, 2025.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient vision-language-action manipulation via adaptive token caching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. VLABench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

Linqing Zhong, Yi Liu, Yifei Wei, Ziyu Xiong, Si Liu, and Guanghui Ren. ACoT-VLA: Action chain-of-thought for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

When to Switch: Reliable Action-Chunk Extension for Vision-Language-Action Models

- Appendix -

Overview of Appendix

We provide the table of contents for the Appendix below:

A. Subskill Transition Discovery
- A.1. Visualization of Subskill Transitions
- A.2. Statistics of Subskill Transitions
- A.3. Change-point Detection
- A.4. Soft Labels for Subskill Transitions
- A.5. Validity of Transition Labels with Demonstration Generator Labels

B. Additional Experiments
- B.1. Comparison on LIBERO
- B.2. Scaling to Longer Chunks
- B.3. Results with VLM Joint Training
- B.4. Decoupling Prediction and Execution Chunk Lengths
- B.5. Generalization to Other VLA Baselines
- B.6. Latency Overhead Comparison
- B.7. Training Efficiency Comparison
- B.8. Statistical Significance

C. Additional Ablation Study
- C.1. Sensitivity to Change-point Detection Settings
- C.2. Effect of Per-step Learnable Gate
- C.3. Effect of Soft Transition Label
- C.4. Effect of Teacher Forcing and Jittering

D. Additional Analyses
- D.1. Accuracy of Transition-timing Prediction
- D.2. Sensitivity to Transition-timing Shift at Inference
- D.3. Effect of the Prior on Generated Switching Timing
- D.4. Decomposing Transition Errors by Action Component
- D.5. Decomposing Idle Time in Real-robot Experiments

E. Additional Details
- E.1. Details on Real-robot Experiments
- E.2. Details on Simulation Speedup Measurement
- E.3. Details on Other VLA Baselines
- E.4. Details on Action Error Measurement
- E.5. Details on Dataset and Evaluation Protocols
- E.6. Pseudocode for Training and Inference

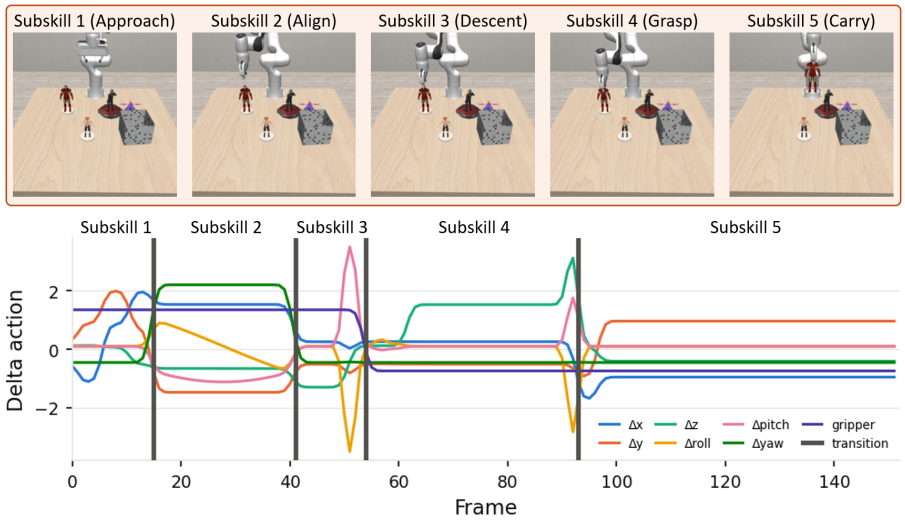
F. Further Discussion
- F.1. Limitations
- F.2. Broader Impact

A SUBSKILL TRANSITION DISCOVERY

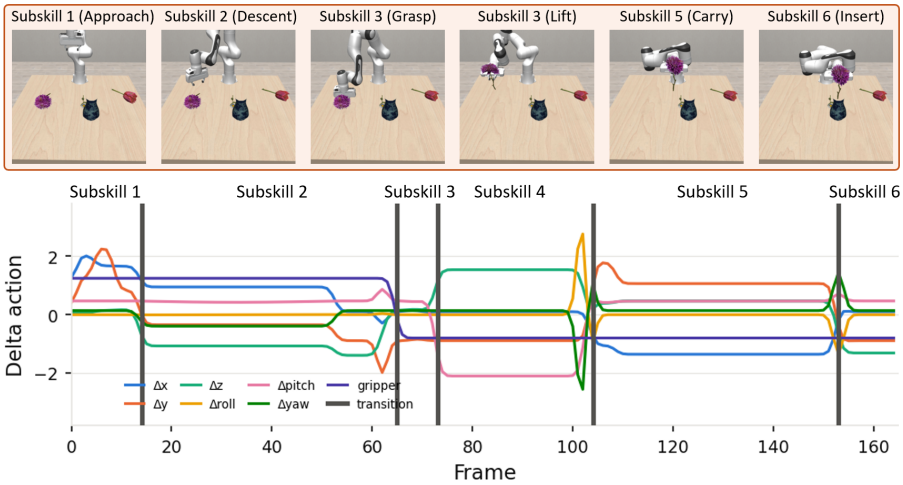
To obtain subskill transition labels, we apply unsupervised subskill discovery (Jia et al., 2024) to the action sequence of each training demonstration, which segments it into subskills without any manual annotation using change-point detection, *i.e.*, PELT algorithm (Killick et al., 2012). Below, we first visualize examples of the detected transitions, report statistics of transitions, describe how they are obtained, and then validate the transition labels.

A.1 VISUALIZATION OF SUBSKILL TRANSITIONS

Fig. 7 shows the detected transitions on two VLABench demonstrations (Zhang et al., 2025), overlaying the PELT change points on the action signal alongside representative frames of each subskill. The subskill names shown in the figure are generated with Claude Code (Anthropic, 2025) for illustration only; RACE uses the transition points as labels for policy learning, not the subskill categories. The change points fall at visible subskill boundaries, such as the switch from approach to grasp.



(a) Language instruction: *"Put the ironman into the giftbox."*



(b) Language instruction: *"Insert the chrysanthemum into the vase."*

Figure 7: Visualization of subskill transitions. Change points detected by PELT (Killick et al., 2012) are overlaid on the action sequences of two VLABench demonstrations, alongside a representative frame for each subskill. Subskill names are generated with Claude Code (Anthropic, 2025) for illustration only; only the transition points are used as labels for policy learning, not the semantics.

A.2 Statistics of Subskill Transitions

Tab. 7 summarizes the detected transitions on the VLABench training demonstrations. Across the ten tasks, an episode contains 3.57 change points on average, ranging from 2.19 for *"select_fruit"* to 5.59 for *"add_condiment"*. The number of transitions is not simply proportional to episode length: *"select_fruit"* has among the longest episodes (151 steps) but the fewest transitions, whereas *"select_chemistry_tube"* has short episodes (78 steps) yet more transitions than several longer tasks.

Table 7: Statistics of detected subskill transitions on VLABench training demonstrations. For each task, we report the number of steps and the number of subskill transitions (*i.e.*, change points) per episode as mean $\pm$ standard deviation.

Task	# Episodes	Total steps per episode	Transitions per episode
add_condiment	500	167.6 ± 14.1	5.59 ± 0.80
insert_flower	500	167.3 ± 15.6	5.42 ± 0.87
select_toy	500	160.2 ± 13.7	4.02 ± 1.09
select_chemistry_tube	500	78.4 ± 8.6	3.68 ± 0.69
select_drink	500	97.4 ± 8.0	3.31 ± 0.60
select_poker	500	75.4 ± 7.0	3.06 ± 0.47
select_painting	500	68.7 ± 8.9	3.00 ± 0.56
select_book	500	72.7 ± 8.6	2.77 ± 0.66
select_mahjong	500	111.4 ± 15.4	2.67 ± 0.88
select_fruit	500	151.3 ± 15.0	2.19 ± 1.12
All	5,000	115.0 ± 41.8	3.57 ± 1.35

A.3 Change-point Detection

Following CoTPC (Jia et al., 2024) and PRIMT (Wang et al., 2025), we segment each demonstration into subskills with the PELT algorithm (Killick et al., 2012). Given a demonstration of T actions $(\mathbf{a}_1, \ldots, \mathbf{a}_T)$, where each $\mathbf{a}_t \in \mathbb{R}^D$ consists of a pose delta and a gripper command, we form the input signal from the first difference of the pose delta (with angles wrapped to $[-\pi, \pi)$) and the gripper command, and standardize each dimension within the episode, $\tilde{\mathbf{a}}_t = (\mathbf{s}_t - \boldsymbol{\mu})/\boldsymbol{\sigma}$, where $\mathbf{s}_t$ is the signal at step t, $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^D$ are its per-dimension mean and standard deviation over the episode, and the division is element-wise. This normalization prevents action dimensions with different scales (*e.g.*, translation, rotation, and gripper control) from disproportionately influencing change-point detection. PELT then identifies a set of change points $\mathcal{C} = \{c_1, \ldots, c_m\}$ by minimizing

$$\sum_{i=0}^{m} \mathrm{cost}\big(\tilde{\mathbf{a}}_{c_i:c_{i+1}}\big) + \beta m, \qquad c_0 = 1, \quad c_{m+1} = T + 1, \tag{8}$$

where $\tilde{\mathbf{a}}_{c_i:c_{i+1}}$ denotes the action segment between consecutive change points and β controls the penalty for introducing additional change points. We use a kernel-based segment cost with a Gaussian kernel $k(\mathbf{u}, \mathbf{v}) = \exp(-\gamma\|\mathbf{u}-\mathbf{v}\|^2)$, which captures changes in the distribution of actions across subskills, such as a transition from steady translational motion to small corrective movements. For a segment spanning steps a through $b-1$, the cost is

$$\mathrm{cost}(\tilde{\mathbf{a}}_{a:b}) = \sum_{t=a}^{b-1} k(\tilde{\mathbf{a}}_t, \tilde{\mathbf{a}}_t) - \frac{1}{b-a} \sum_{s,t=a}^{b-1} k(\tilde{\mathbf{a}}_s, \tilde{\mathbf{a}}_t). \tag{9}$$

Each change point c is the first step of a new segment; we discard the terminal boundary. We set $\beta = 4$ for all benchmarks and set the bandwidth γ per demonstration by the median heuristic, *i.e.*, the inverse of the median squared distance over all pairs of standardized action steps in that demonstration ($\gamma \approx 0.1$ on our signals).

A.4 Soft Labels for Subskill Transitions

PELT yields hard labels that mark a single step per transition. Since a subskill transition may span several action steps, we convert these into soft labels. For a chunk starting at time t, let d_{t+h} be the number of steps from action step $t+h$ to its nearest change point in $\mathcal{C}$, and let $\mathbf{d}_t = (d_t, \ldots, d_{t+H-1})$. The target $\mathbf{p}_t = (p_t, \ldots, p_{t+H-1})$ is

$$\mathbf{p}_t = \exp\left(-\frac{\mathbf{d}_t^2}{2\sigma_p^2}\right), \tag{10}$$

where the square and the exponential are applied element-wise, with $\sigma_p = 1$. Scores below $\exp(-R^2/2)$ for $R = 3$ are set to zero, so each transition affects at most seven steps. Steps beyond the end of an episode are clamped to its last action and receive a score of zero. The target is computed once before training and stored alongside the demonstrations.

Visualization of Soft Labels and Predicted Prior. Fig. 8 illustrates how this target is constructed and what the head predicts, using a chunk of one of the held-out demonstrations (Appendix A.5). PELT detects change points from the standardized action signal (top), here where a rotation ends and where the gripper closes. Within the chunk, each change point is converted into a soft label that peaks at its position and spreads over neighboring steps (bottom, bars). The head is trained to reproduce this soft label from the auxiliary pass, so its prediction $\hat{\mathbf{p}}_t$ is a per-position score of how likely each action in the chunk is a transition, rather than a single transition index. In this example, the predicted prior peaks at both change points, although it is lower and slightly wider than the label. We chose this example to illustrate the target and prediction clearly; the head does not capture every change point in general, and its average accuracy is reported in Appendix D.1.

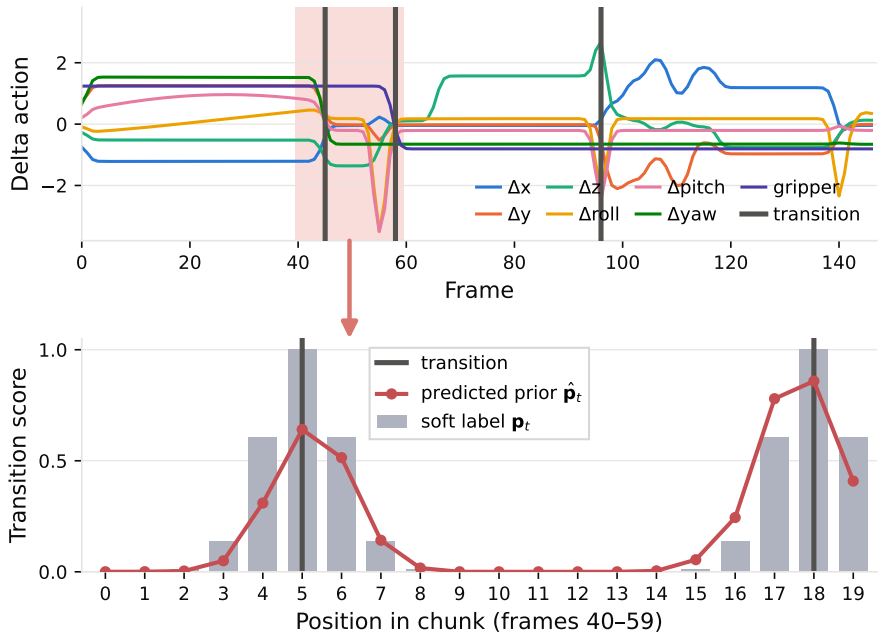


Figure 8: An example of transition-timing target and prediction on a held-out VLABench demonstration ($H_{\text{exec}} = 20$). Top: standardized action signal and PELT change points; the pink-shaded chunk is enlarged in the bottom plot. Bottom: soft label $\mathbf{p}_t$ and predicted prior $\hat{\mathbf{p}}_t$ in this chunk.

A.5 Validity of Transition Labels with Demonstration Generator Labels

To assess whether the PELT change points capture meaningful subskill transitions, we compare them with reference boundaries obtained from the VLABench demonstration generator. We conduct this analysis on 300 newly generated demonstrations spanning all ten VLABench tasks, which are not used for training and serve as the held-out demonstrations throughout the appendix. The reference boundaries are used only for evaluation. We first describe how the generator provides the reference boundaries and then evaluate how closely the PELT points align with them.

Reference Subskill Boundaries from the Demonstration Generator. The VLABench generator constructs each demonstration by executing a sequence of scripted skills. These skills move the end effector through scene-dependent targets, such as pre-grasp, grasp, and lift poses, interleaved with gripper operations. We instrument the generator to record the steps at which it switches between these targets or primitives and use them as reference subskill boundaries. Intermediate motion-planning waypoints are excluded because they lie within a motion segment and do not indicate a change in subskill. This procedure yields an average of 4.5 reference boundaries per episode. Fig. 9 visualizes the resulting reference boundaries alongside the PELT change points. We conduct

this analysis only on VLABench, whose scripted generator provides reference subskill boundaries unavailable in the other benchmarks, *e.g.*, human teleoperation used in LIBERO (Liu et al., 2023).

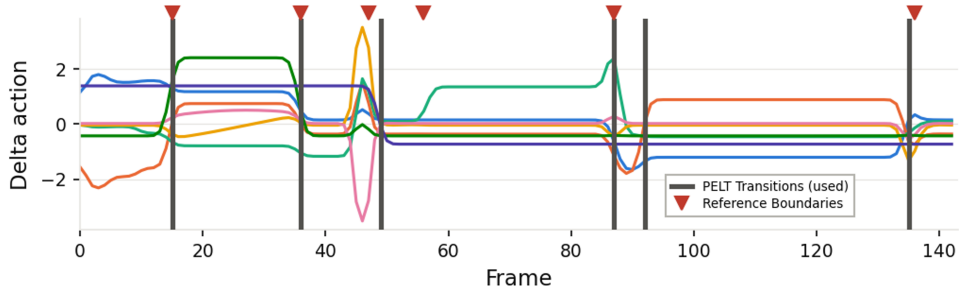


Figure 9: Qualitative comparison between PELT change points and reference subskill boundaries recorded by the VLABench demonstration generator.

Comparison with Reference Subskill Boundaries. Tab. 8 evaluates the three transition-label sources considered in the ablation study of Tab. 4a: random labels, speed-minimum labels (Nie et al., 2026), and PELT change points. We apply each source to 300 newly generated VLABench demonstrations and compare its labels with the reference subskill boundaries recorded by the demonstration generator (4.5 per episode on average). We use a tolerance of ± 3 steps, the range over which each label is spread by the soft labels used for training (Appendix A.4). Precision is the fraction of labels that have a reference boundary within ± 3 steps, and recall is the fraction of reference boundaries that have a label within ± 3 steps. Both are counted per transition rather than per action step, without one-to-one matching, and computed by pooling all episodes rather than averaging per episode. Random labels draw the number of labels per episode uniformly from $\{1, \ldots, 9\}$ and place them at uniformly sampled steps. PELT achieves substantially higher precision and recall than the alternative sources, indicating closer alignment with the reference subskill boundaries. This result is consistent with the transition-label ablation (Tab. 4a), where PELT outperforms speed-minimum labels.

Table 8: Agreement between transition labels and the reference subskill boundaries recorded by the VLABench demonstration generator, on 300 newly generated demonstrations with a matching tolerance of ± 3 steps. Precision and recall are counted per transition and pooled over all episodes.

Transition-label sources	Transitions per episode	Precision	Recall
Random	5.0	0.30	0.28
Speed minima (Nie et al., 2026)	2.4	0.40	0.21
PELT (used)	3.8	0.82	0.70

B Additional Experiments

B.1 Comparison on LIBERO

In Tab. 9, we compare RACE on LIBERO with the $\pi_{0.5}$ baseline, its fine-tuned variants, and other chunk-extension, adaptive chunking, and efficient VLA methods, all built on the same $\pi_{0.5}$ baseline. The success rates of the $\pi_{0.5}$ baseline and Latent Bridge are taken from Liu et al. (2026b), and those of StreamPI (Liu et al., 2026c), ACoT-VLA (Zhong et al., 2026), PolicyTrim (Wang et al., 2026c), SAFE-Pruner (Ma et al., 2026), and GridS (Feng et al., 2026b) from their original papers, while Spd is measured for all methods in our setup. At every H_{exec}, RACE outperforms the fine-tuned $\pi_{0.5}$ executing the same number of actions. With $H_{\text{exec}} \in \{10, 15\}$, RACE performs comparably to recent state-of-the-art VLAs (Liu et al., 2026c; Zhong et al., 2026), which require more inference time than the $\pi_{0.5}$ baseline, and outperforms adaptive chunking methods (Wang et al., 2026a; Nie et al., 2026) and efficient VLA methods (Liu et al., 2026b; Ma et al., 2026; Feng et al., 2026b) in both success rate and per-episode inference speedup. Since these efficient VLA methods reduce the per-call cost, they are complementary to RACE and could be combined with it. Even at $H_{\text{exec}} = 20$, RACE remains competitive with the $\pi_{0.5}$ baseline while requiring far less inference time per episode. PolicyTrim (Wang et al., 2026c) achieves the largest speedup by optimizing task completion through reinforcement learning, but its success rate is lower than that of RACE with $H_{\text{exec}} \in \{10, 15\}$. Moreover, its reinforcement learning incurs a substantial training cost, as it collects fresh policy

rollouts at every iteration and trains a separate model for each suite: it takes 457 hours of wall-clock training per suite (about 1,828 hours for all four suites), whereas RACE takes 17 hours for all four suites, both on four NVIDIA RTX A6000 GPUs (Appendix B.7).

Table 9: Comparison on LIBERO. $*$ PolicyTrim trains a separate model for each suite, and its H_{exec} is averaged over the four suites. Its training cost is also substantially higher; see Appendix B.7. Spd is the per-episode inference speedup relative to the $\pi_{0.5}$ baseline. ‡ indicates the average execution length, since adaptive chunking methods truncate each chunk adaptively.

Method	H_{exec}	Spd↑	Spatial	Object	Goal	Long	Avg.
$\pi_{0.5}$ Baseline	5	1.00×	98.83	98.17	97.00	93.83	96.96
Action-Chunk Extension							
$\pi_{0.5}$ Fine-tuning	10	1.98×	97.60	98.80	97.60	94.40	97.10
	15	2.96×	97.40	99.00	97.80	93.60	96.95
	20	3.91×	94.80	97.60	94.00	91.00	94.35
PolicyTrim* (Wang et al., 2026c)	13.75	4.85×	97.80	98.50	98.80	93.30	97.10
RACE (ours)	10	1.95×	99.40	99.60	98.60	97.00	98.65
	15	2.86×	99.00	99.80	98.60	96.00	98.35
	20	3.78×	97.60	99.20	95.80	94.20	96.70
Recent State-of-the-Art VLAs							
StreamPI (Liu et al., 2026c)	5	0.91×	98.80	99.80	99.60	95.00	98.30
ACoT-VLA (Zhong et al., 2026)	5	0.68×	99.40	99.60	98.80	96.00	98.45
Adaptive Chunking Methods							
AutoHorizon (Wang et al., 2026a)	7.10‡	1.43×	97.20	99.40	98.20	94.20	97.25
PACE (Nie et al., 2026)	8.96‡	1.81×	98.00	98.80	97.80	94.80	97.35
Other Efficient VLA Methods							
SAFE-Pruner (Ma et al., 2026)	5	1.45×	97.00	98.80	96.00	90.20	95.50
Latent Bridge (Liu et al., 2026b)	5	1.64×	99.00	97.67	97.33	93.67	96.92
GridS (Feng et al., 2026b)	5	1.24×	98.60	98.80	98.40	95.20	97.75

B.2 Scaling to Longer Chunks

In Tab. 10, we evaluate RACE with execution chunk lengths ranging from $H_{\text{exec}} = 5$ to 40 to examine the limits of chunk extension. RACE maintains a higher success rate than the $\pi_{0.5}$ baseline up to $H_{\text{exec}} = 30$, but falls slightly below it at $H_{\text{exec}} = 40$. These results show that RACE supports substantial chunk extension. While longer chunks further reduce per-episode policy-inference time, success rates decline at larger execution lengths. This decline is likely because open-loop drift errors, which grow with the chunk length, cannot be fully compensated by the transition-timing prior.

Table 10: Results across a wide range of execution chunk lengths on VLABench.

Method	H_{exec}	Spd↑	In-dist.		Category		Common.		Instruct.		Texture		Avg.	
			SR	PS	SR	PS	SR	PS	SR	PS	SR	PS	SR	PS
$\pi_{0.5}$ Baseline	5	1.00×	40.4	56.7	21.4	35.6	17.0	33.7	18.0	35.9	26.0	42.1	24.6	40.8
RACE (ours)	5	0.96×	51.6	68.2	25.0	38.1	26.0	40.0	20.8	37.6	29.0	45.3	30.5	45.8
	10	1.92×	57.6	70.9	30.4	41.8	23.2	39.0	25.0	42.0	36.0	52.8	34.4	49.3
	20	3.68×	54.0	68.2	27.2	38.8	28.2	41.4	21.0	39.4	34.8	51.0	33.0	47.8
	30	5.11×	50.2	65.9	24.4	35.9	20.0	36.1	16.4	36.4	31.4	47.0	28.5	44.3
	40	6.82×	46.0	63.1	21.8	33.3	15.4	34.0	13.0	33.0	23.8	41.0	24.0	40.9

B.3 Results with VLM Joint Training

To examine whether the VLM frozen setting limits our framework, we additionally update the VLM during post-training and evaluate on VLABench with $H_{\text{exec}} = 20$ in Tab. 11. VLM joint training improves the average success rate and progress score, with the largest gains on the instruction and texture tracks that rely on the VLM. Thus, RACE is not tied to the VLM frozen setting and further benefits from adapting the VLM. We nevertheless keep the VLM frozen in the main experiments, since joint training requires substantially more memory and compute (Tab. 12).

Table 11: Effect of training the VLM jointly on VLABench with $H_{\text{exec}} = 20$. VLM frozen indicates our post-training setting, where the VLM is kept frozen and only the action expert is post-trained. VLM joint training means that the VLM is also updated.

Setting	In-dist.		Category		Common.		Instruct.		Texture		Avg.	
	SR	PS	SR	PS	SR	PS	SR	PS	SR	PS	SR	PS
VLM frozen (used)	54.0	68.2	27.2	38.8	28.2	41.4	21.0	39.4	34.8	51.0	33.0	47.8
VLM joint training	57.8	72.1	30.6	43.2	28.6	43.9	31.0	50.4	46.6	62.6	38.9	54.4

Table 12: Training cost of the VLM-frozen and VLM joint training settings on VLABench with $H_{\text{exec}} = 20$. Both models are trained for 40K steps on four RTX A6000 GPUs. Peak memory is the maximum allocated memory per GPU.

Setting	Training step	Learnable params (M)	Time per update (s)	Wall-clock time (h)	Peak memory (GB/GPU)
VLM frozen (used)	40K	811	3.2	36	18
VLM joint training	40K	3,734	8.6	96	40

B.4 Decoupling Prediction and Execution Chunk Lengths

On the real robot, each policy predicts twice as many actions as it executes ($H = 2H_{\text{exec}}$), since real-time chunking (Black et al., 2025) generates the next chunk while the remaining actions are executed (Sec. 4.5). In the simulation experiments, in contrast, RACE and the fine-tuned $\pi_{0.5}$ predict and execute the same number of actions ($H = H_{\text{exec}}$) to isolate the effect of extending the chunk, since the policy then predicts exactly as far ahead as it executes. We examine the decoupled setting of the real robot in simulation, training both models to predict $H = 40$ actions while executing only the first $H_{\text{exec}} = 20$ on VLABench. As shown in Tab. 13, predicting a longer chunk improves both models. This is likely because errors grow toward the end of a chunk (as analyzed in Fig. 4 of the main manuscript), so executing only its first half avoids the least accurate actions.

Table 13: Results with different prediction chunk lengths H at the same execution chunk length $H_{\text{exec}} = 20$ on VLABench, with the $\pi_{0.5}$ baseline as reference. Only the first H_{exec} of the H predicted actions are executed per policy call.

Method	H	H_{exec}	In-dist.		Category		Common.		Instruct.		Texture		Avg.	
			SR	PS	SR	PS	SR	PS	SR	PS	SR	PS	SR	PS
$\pi_{0.5}$ Baseline	10	5	40.4	56.7	21.4	35.6	17.0	33.7	18.0	35.9	26.0	42.1	24.6	40.8
$\pi_{0.5}$ Fine-tuning	20	20	48.8	65.3	22.8	37.2	20.2	38.2	20.8	39.3	27.8	43.9	28.1	44.8
	40	20	52.6	68.2	24.0	36.5	22.4	40.5	20.8	40.9	36.0	53.7	31.2	48.0
RACE (ours)	20	20	54.0	68.2	27.2	38.8	28.2	41.4	21.0	39.4	34.8	51.0	33.0	47.8
	40	20	56.8	70.2	28.0	38.6	27.4	43.3	28.2	44.9	35.8	51.9	35.2	49.8

B.5 Generalization to Other VLA Baselines

To examine the generality of our framework, we apply it to two other VLA baselines, GR00T-N1.6 (NVIDIA et al., 2025) and OpenVLA-OFT (Kim et al., 2025), and evaluate them on LIBERO in Tab. 14. GR00T-N1.6 generates actions by flow matching, like $\pi_{0.5}$, whereas OpenVLA-OFT produces actions through parallel decoding and regression. The two baselines thus assess whether our transition-timing conditioning generalizes to another flow-matching model and beyond flow-based generation. Implementation details for these VLA models are provided in Appendix E.3. On GR00T-N1.6, RACE outperforms the fine-tuned variant at every chunk length, and at $H_{\text{exec}} = 16$ it even surpasses the baseline that executes half as many actions, with the largest gains on the long-horizon suite at $H_{\text{exec}} \in \{16, 24\}$, where fine-tuning degrades most. On OpenVLA-OFT, RACE again outperforms the fine-tuned variant at every chunk length, but neither recovers the baseline's success rate and the gain over fine-tuning is smaller. One possible reason is the absence of a denoising process: RACE is designed to condition the iterative refinement of the action expert (Fig. 5c), whereas OpenVLA-OFT regresses the chunk in a single pass.

B.6 Latency Overhead Comparison

Tab. 15 compares the per-call latency overhead of RACE and ACoT-VLA (Zhong et al., 2026) over the $\pi_{0.5}$ baseline. All models are measured on a single NVIDIA RTX A6000 GPU, using the official

Table 14: Generalization to other VLA baselines on LIBERO.

(a) GR00T-N1.6 (NVIDIA et al., 2025)

Method	H_{exec}	Spatial	Object	Goal	Long	Avg.
Baseline	8	100.0	100.0	100.0	92.5	98.1
Fine-tuning	16	100.0	100.0	98.6	94.9	98.4
	24	100.0	100.0	98.4	89.6	97.0
	32	98.6	98.8	95.8	90.2	95.8
RACE (ours)	16	100.0	100.0	100.0	97.0	99.3
	24	100.0	100.0	100.0	95.2	98.8
	32	99.0	100.0	99.0	90.0	97.0

(b) OpenVLA-OFT (Kim et al., 2025)

Method	H_{exec}	Spatial	Object	Goal	Long	Avg.
Baseline	8	97.6	98.4	97.9	94.5	97.1
Fine-tuning	16	96.8	99.0	96.2	92.2	96.1
	24	93.0	98.4	92.0	90.2	93.4
	32	92.4	96.4	93.6	88.6	92.8
RACE (ours)	16	97.4	98.5	95.8	94.4	96.5
	24	93.8	97.8	93.0	92.6	94.3
	32	93.2	96.8	92.2	91.4	93.4

implementation codes for ACoT-VLA. Since both methods share the same VLM and action expert, the difference comes only from the modules each adds. RACE adds the auxiliary one-step denoising pass and the transition-timing prediction head (Sec. 3.2) and the transition-conditioned modulation (Sec. 3.3), which together cost about 4% of a policy call. ACoT-VLA instead runs a second flow-matching denoising in its explicit action reasoner to produce a coarse reference trajectory, and an additional cross-attention in its implicit action reasoner, where the former alone costs about as much as the action expert itself. This latency gap explains why ACoT-VLA achieves a lower speedup than RACE at the same H_{exec} in Tab. 1. All latencies here are measured in the same benchmark setting for every model; our current real-robot implementation of RACE runs without CUDA graphs and thus has a higher latency (Appendix D.5).

Table 15: Per-call inference latency, measured on a single NVIDIA RTX A6000 GPU. † denotes ACoT-VLA components measured in our setup with its official implementation, where the explicit action reasoner uses an additional 10-step denoising process to generate coarse action trajectories.

Component	$\pi_{0.5}$	ACoT-VLA	RACE (ours)
VLM	53.5 ms	53.5 ms	53.5 ms
Action expert (*i.e.*, 10 denoising steps)	30.8 ms	30.8 ms	30.8 ms
Explicit action reasoner†	–	32.5 ms	–
Implicit action reasoner†	–	3.8 ms	–
Aux. one-step denoising pass (Sec. 3.2)	–	–	3.0 ms
Transition-timing head (Sec. 3.2)	–	–	∼0.3 ms
Transition-conditioned modulation (Sec. 3.3)	–	–	∼0.0 ms
Total latency	84.3 ms	120.6 ms	87.6 ms
Latency ratio to $\pi_{0.5}$	1.00×	1.43×	1.04×

B.7 Training Efficiency Comparison

To highlight the training efficiency of our framework, we compare the training cost of RACE with the fine-tuned $\pi_{0.5}$ and other methods, such as ACoT-VLA (Zhong et al., 2026), a recent state-of-the-art VLA, and PolicyTrim (Wang et al., 2026c), a chunk-extension method, on VLABench in Tab. 16 and LIBERO in Tab. 17. All methods are measured on four NVIDIA RTX A6000 GPUs, and the compared methods are run from their official implementations. RACE adds only the transition-timing prediction head and the modulation parameters to the action expert, so its training cost is only moderately higher than that of the fine-tuned $\pi_{0.5}$. In contrast, ACoT-VLA also trains the visual encoder and its components with a larger batch, taking more than an order of magnitude longer to train than RACE, and PolicyTrim requires two-stage reinforcement learning with a much larger batch, which incurs substantially higher training cost since every iteration collects fresh policy rollouts in the simulator. Moreover, while the other methods train a single model on all four LIBERO suites, PolicyTrim trains a separate model for each suite, so covering all four suites takes an estimated 1,828 hours, over 100 times longer than RACE (Tab. 17). RACE thus extends the action chunk with far less training cost than these methods.

B.8 Statistical Significance

To examine how much the results vary across training runs, we train RACE with $H_{\text{exec}} = 20$ on VLABench with three different random seeds in Tab. 18. The average success rate is 33.3 ± 0.5

Table 16: Training efficiency comparison on VLABench. All methods are measured on four NVIDIA RTX A6000 GPUs, with the compared methods run from their official implementations. †: following its official setting, ACoT-VLA freezes the LLM backbone and trains the visual encoder, the action expert, and its components.

Method	Training step	Global batch	Learnable params (M)	Time per update (s)	Wall-clock time (h)	Peak memory (GB/GPU)
$\pi_{0.5}$ Fine-tuning	40K	64	693	2.5	28	17
ACoT-VLA† (Zhong et al., 2026)	60K	128	1,309	50.6	843	33
RACE (ours)	40K	64	811	3.2	36	18

Table 17: Training efficiency comparison on LIBERO. All methods are measured on four NVIDIA RTX A6000 GPUs, with PolicyTrim and ACoT-VLA run from their official implementations. †: PolicyTrim trains a separate model for each of the four LIBERO suites with two-stage reinforcement learning (500 + 500 RL iterations); its values are for a single suite, and the parentheses give the estimated total for all four suites. Its time per update is per RL iteration, including rollout collection. ‡: following its official setting, ACoT-VLA freezes the LLM backbone and trains the visual encoder, the action expert, and its components.

Method	Training step	Global batch	Learnable params (M)	Time per update (s)	Wall-clock time (h)	Peak memory (GB/GPU)
$\pi_{0.5}$ Fine-tuning	20K	64	693	2.5	14	17
PolicyTrim† (Wang et al., 2026c)	500 + 500	2,048	693	1,644	457 (×4 = 1,828)	36
ACoT-VLA‡ (Zhong et al., 2026)	20K	128	1,309	49.8	277	33
RACE (ours)	20K	64	811	3.0	17	18

(mean ± standard deviation), and every run outperforms the fine-tuned $\pi_{0.5}$ in Tab. 1, which is trained once, by a margin much larger than this variation. Individual tracks vary more across runs, so small differences on a single track should be interpreted with care. The model in Tab. 1 is the first run, whose results are close to the mean.

Table 18: Statistical significance of RACE on VLABench with $H_{\text{exec}} = 20$. In addition to the model reported in Tab. 1 (Run 1), we train RACE twice more with different random seeds (Runs 2 and 3). The last row reports the mean and standard deviation over the three runs.

RACE	H_{exec}	In-dist.		Category		Common.		Instruct.		Texture		Avg.	
		SR	PS	SR	PS	SR	PS	SR	PS	SR	PS	SR	PS
Run 1 (reported)	20	54.0	68.2	27.2	38.8	28.2	41.4	21.0	39.4	34.8	51.0	33.0	47.8
Run 2	20	53.6	68.9	28.8	40.7	29.0	45.0	22.8	40.4	35.0	52.3	33.8	49.5
Run 3	20	52.4	66.8	32.2	42.8	25.8	40.9	23.8	43.1	30.4	49.3	32.9	48.6
Mean ± std	20	53.3±0.8	68.0±1.1	29.4±2.6	40.8±2.0	27.7±1.7	42.4±2.2	22.5±1.4	41.0±1.9	33.4±2.6	50.9±1.5	33.3±0.5	48.6±0.9

C Additional Ablation Study

C.1 Sensitivity to Change-point Detection Settings

To examine the sensitivity of RACE to the PELT settings, we vary the penalty β in Eq. (8) of Appendix A.3, which controls the number of detected transitions, and retrain our framework with the resulting labels on VLABench with $H_{\text{exec}} = 20$ in Tab. 19. Performance stays stable from $\beta = 2$ to 8 despite a threefold change in label density, so RACE is not sensitive to the PELT settings.

Table 19: Sensitivity to the PELT penalty on VLABench with $H_{\text{exec}} = 20$. We vary the penalty β in Eq. (8) of Appendix A.3.

Penalty β	# of transitions per episode	Avg. SR
2	5.10	33.1
8	1.82	32.8
4 (used)	3.57	33.0

C.2 Effect of Per-step Learnable Gate

Tab. 20 evaluates the learnable per-step gate α^k in Eq. (1), which controls the strength of prior conditioning at each denoising step. Replacing these gates with a fixed value of $\alpha^k = 1$ applies the prior equally at every step and performs slightly worse. This suggests that letting the model adjust the conditioning strength per step is beneficial, in line with the learned gates in Fig. 5c, which take on a step-dependent pattern after training.

Table 20: Effect of the per-step learnable gate on VLABench with $H_{\text{exec}} = 20$.

Gate α^k of Eq. (1) in Sec. 3.1	Avg. SR
None ($\alpha^k = 1$)	30.7
Learnable per step (ours)	33.0

C.3 Effect of Soft Transition Label

As described in Sec. 3.2 and Appendix A.4, we convert the hard transition labels from PELT into soft labels. Tab. 21 compares these two labels (hard labels and soft labels) on VLABench with $H_{\text{exec}} = 20$. We observe that training with hard labels (*i.e.*, one-hot labels at every change point) lowers the success rate, suggesting that spreading each transition over its neighboring steps gives a more useful target than a single step, as a subskill transition spans several actions and the detected points may not exactly match the true transition steps.

Table 21: Effect of soft transition labels on VLABench with $H_{\text{exec}} = 20$. Hard labels mark only the detected transition step; soft labels spread each transition over neighboring steps with a Gaussian profile (Appendix A.4).

Transition label	Avg. SR
Hard labels	32.3
Soft labels (used)	33.0

C.4 Effect of Teacher Forcing and Jittering

During training, we condition the action expert on the transition target $\mathbf{p}_t$ instead of the predicted prior $\hat{\mathbf{p}}_t$, *i.e.*, teacher forcing, avoiding reliance on inaccurate timing predictions early in training. To improve robustness to imperfect priors, we randomly shift the target by one action step with probability 0.5 when using it for conditioning, *i.e.*, jittering, while keeping the target for $\mathcal{L}_{\text{timing}}$ unchanged. At inference, we condition on the predicted prior $\hat{\mathbf{p}}_t$. Tab. 22 evaluates both choices: each has only a marginal effect on VLABench, indicating that RACE is not sensitive to them. However, jittering mitigates the performance drop under inference-time shifts of several steps (Tab. 24 in Appendix D.2).

Table 22: Effect of teacher forcing and jittering on VLABench with $H_{\text{exec}} = 20$. The robustness provided by jittering against inference-time transition shifts is evaluated separately in Tab. 24 of Appendix D.2.

Training strategy	Avg. SR
w/o teacher forcing	32.9
w/o jittering	32.8
Both (ours)	33.0

D Additional Analyses

D.1 Accuracy of Transition-timing Prediction

To check how accurately the head predicts transition timing, we compare its predictions with the PELT labels in Tab. 23 on 5,000 VLABench training demonstrations and the 300 held-out demonstrations (Appendix A.5). All thresholds are selected on the training demonstrations and applied unchanged to the held-out demonstrations. These results measure agreement with the PELT labels on demonstration states, not on the states visited during policy rollouts.

Evaluation Metrics. We evaluate three aspects: **Transition-chunk detection** checks whether a chunk contains a transition, scoring each chunk by the maximum of the predicted prior $\hat{\mathbf{p}}_t$ with a threshold of 0.14. **Transition-point localization** checks where the transition occurs: in each chunk containing a transition, we check whether the position with the highest prior lies within ± 1 or ± 3 steps of the nearest PELT change point. **Multi-transition localization** checks how many transitions in a chunk are captured, since a chunk may contain several transitions while localization credits only the highest peak. We take every peak above a threshold (0.35, 0.35, and 0.40 for $H_{\text{exec}} = 10$, 15, and 20) as a predicted transition and match it to the change points within ± 1 or ± 3 steps. Precision is the fraction of predicted transitions that are matched, and recall is the fraction of change points that are matched. This threshold is higher than that for detection because spurious peaks count as false positives here. Since localization always uses a single peak per chunk, its values are not directly comparable to recall. For transition labels and predictions, we use tolerances of ± 1 and ± 3 steps, matching the resolution at which RACE is trained: the soft labels have a width of $\sigma_p = 1$ and are truncated at $R = 3$ steps (Appendix A.4), and the conditioning target is jittered by one step during training (Appendix C.4).

Table 23: Accuracy (%) of the transition-timing prediction head on VLABench across execution chunk lengths H_{exec}. *Transition-chunk Detection* evaluates, for each chunk, whether it contains a transition. *Transition-point Localization* evaluates, for each chunk containing a transition, where the transition occurs. *Multi-transition Localization* evaluates how many of the transitions within a chunk are captured, counting every predicted peak rather than only the highest one. Results are reported on both training and held-out demonstrations.

Evaluation type	Metric	Training demos.			Held-out demos.		
		H_{exec}			H_{exec}		
		10	15	20	10	15	20
Transition-chunk Detection	AUROC	96.0	95.4	94.4	89.2	89.7	86.5
	Acc.	87.1	87.0	86.0	79.5	79.6	77.0
Transition-point Localization	Acc. within ± 1 step	80.3	75.4	72.1	68.2	58.5	50.6
	Acc. within ± 3 steps	90.6	85.8	83.3	81.0	69.8	65.1
Multi-transition Localization	Precision within ± 1 step	86.0	86.0	87.5	84.2	81.2	80.9
	Precision within ± 3 steps	90.4	90.9	92.5	93.1	91.2	94.1
	Recall within ± 1 step	67.7	65.6	62.7	54.8	52.2	50.0
	Recall within ± 3 steps	71.2	69.3	66.3	60.5	58.6	58.1

Accuracy Results of Transition-timing Head. On training demonstrations, detection and localization are well above random guessing (50% for detection; 28%, 19%, and 15% within ± 1 step for localization at $H_{\text{exec}} = 10$, 15, and 20), and most predicted peaks fall within the range covered by the soft labels (Appendix A.4). On held-out demonstrations, accuracy drops, most for localization with longer chunks, but remains well above random guessing. We compare detection between the two sets by AUROC, since accuracy depends on the fraction of transition chunks, which is higher on held-out demonstrations. For multi-transition localization, precision is high, so most predicted transitions are real, but recall is lower: of 3.6 and 3.8 transitions per episode on training and held-out demonstrations, about one and up to 1.6 are missed, respectively. The slightly higher held-out precision within ± 3 steps comes from fewer predicted peaks, not better prediction. Overall, the head predicts transitions with high precision but misses some of them. Although this evaluation is limited to demonstration states, the prior also affects the generated actions and rollouts: RACE switches the gripper within one step of the correct step more often (Tab. 27), and shifting the prior by one step lowers the success rate (Appendix D.2).

D.2 Sensitivity to Transition-timing Shift at Inference

To examine the sensitivity to transition-timing shift, we shift the predicted transition timing by δ steps at inference and report success rates on VLABench with $H_{\text{exec}} = 20$ in Tab. 24; peaks shifted beyond the chunk are dropped, so large shifts also remove part of the prior. Any shift lowers performance, even by a single step, showing that the model uses the predicted prior rather than ignoring it. Without jittering (Appendix C.4), the model degrades more for shifts of several steps (*e.g.*, ± 3 steps), while the two models perform similarly for single-step shifts. Jittering thus mitigates large

drops under timing errors of several steps. The partial recovery at the largest shifts reflects the removed prior rather than tolerance to the shift, reaching a level similar to the zero-prior setting of Tab. 5 in the main manuscript.

Table 24: Sensitivity to shifts of the predicted transition timing at inference on VLABench with $H_{\text{exec}} = 20$. The peaks of the predicted prior are shifted by δ steps within the chunk; peaks shifted beyond the chunk are dropped.

Model	Shift δ of the transition-timing prior										
	-15	-10	-5	-3	-1	0	$+1$	$+3$	$+5$	$+10$	$+15$
w/o jittering	28.6 -4.2	29.8 -3.0	24.6 -8.2	24.8 -8.0	30.1 -2.7	32.8	30.8 -2.0	26.4 -6.4	30.4 -2.4	28.7 -4.1	31.1 -1.7
RACE (ours)	32.1 -0.9	31.0 -2.0	29.4 -3.6	30.3 -2.7	30.2 -2.8	33.0	30.7 -2.3	31.2 -1.8	30.9 -2.1	30.9 -2.1	32.1 -0.9

D.3 Effect of the Prior on Generated Switching Timing

In Tab. 25, to see how the prior affects the generated actions, we shift the predicted prior by k steps or replace it with zeros at inference, while keeping the observation, noise, and weights fixed, and measure when the generated gripper command switches on VLABench with $H_{\text{exec}} = 20$. We use the gripper since its binary command switches at a well-defined step, unlike translation and rotation. Shifting the prior moves the generated switch in the same direction, at 0.23 and 0.28 steps per step of shift on training and held-out demonstrations (95% bootstrap confidence intervals exclude zero for every k), showing that the prior directly controls when the generated actions switch. Since the switch moves by only about a quarter of the shift, the model also relies on features learned from the observation. Replacing the prior with zeros leaves the switch offset nearly unchanged but increases missed switches, consistent with the lower success rate of the zero-prior setting in Tab. 5.

Table 25: Effect of the transition-timing prior on the generated gripper switches on VLABench with $H_{\text{exec}} = 20$. The predicted prior is shifted by k steps or replaced with zeros at inference, over chunks where the demonstrated gripper command switches exactly once. Switch offset is the predicted minus the demonstrated switch step. Values under the predicted prior differ slightly from Tab. 27 due to different noise draws.

Prior at inference	Training				Held-out			
	Switch offset	Exact↑	±1 step↑	Missed↓	Switch offset	Exact↑	±1 step↑	Missed↓
Shift $k = -3$	-0.64	47.3	85.6	2.5	-0.47	55.0	82.2	7.5
Shift $k = -2$	-0.48	57.9	91.3	1.1	-0.26	60.1	89.1	5.1
Shift $k = -1$	-0.25	71.9	94.5	1.3	$+0.06$	58.9	90.0	5.6
Predicted (ours)	-0.05	75.0	97.3	1.0	$+0.29$	53.8	90.8	4.1
Shift $k = +1$	$+0.15$	72.8	95.3	2.1	$+0.50$	45.0	85.2	7.1
Shift $k = +2$	$+0.49$	52.2	88.8	4.6	$+0.88$	26.3	75.9	8.8
Shift $k = +3$	$+0.70$	41.0	80.3	6.4	$+1.16$	16.5	62.0	10.9
Zero	-0.02	62.8	85.4	8.8	$+0.36$	43.6	77.1	15.6

D.4 Decomposing Transition Errors by Action Component

The action error in Fig. 4 averages all seven action dimensions, which can hide differences between action components. We therefore split the errors at transitions into translation (x, y, z), rotation (roll, pitch, yaw), and gripper components, using 500 VLABench training demonstrations at $H_{\text{exec}} = 20$. Errors are measured as in Appendix E.4 and averaged over four noise seeds, except that transitions here include the chunk positions within ±1 step of a PELT change point.

As shown in Tab. 26, translation has the largest errors and thus dominates the overall error. Separating the components shows that RACE reduces errors in all three components over the fine-tuned $\pi_{0.5}$, with larger reductions in rotation and gripper. Since we do not subtract open-loop drift here and RACE also reduces errors away from transitions, these results show that errors at transitions decrease, but not that they decrease more than elsewhere.

Table 26: Action errors (MSE) at transitions by action component on VLABench with $H_{\text{exec}} = 20$.

Error component	MSE		Error reduction (%) over
	$\pi_{0.5}$ Fine-tuning	RACE (ours)	$\pi_{0.5}$ Fine-tuning
Translation	3.448	2.995	13.1
Rotation	0.277	0.190	31.5
Gripper	0.124	0.094	24.6
Overall (7-dim)	1.614	1.378	14.6

Gripper Switching Timing. Since the gripper command takes only two values, its error mainly reflects when the gripper opens or closes. We measure this timing directly in Tab. 27 on chunks where the demonstrated gripper command switches exactly once, taking the predicted switch as the step where the predicted gripper value changes sign. Compared with the fine-tuned $\pi_{0.5}$, RACE switches within ± 1 and ± 3 steps of the correct step more often and misses fewer switches on both training and held-out demonstrations, with a similar false-switch rate, suggesting that the gain is not due to predicting more switches. RACE also switches at the exact step more often on training demonstrations, while this gain is not significant on held-out demonstrations. Mean error is also lower on training demonstrations, and lower but not significantly so on held-out demonstrations.

Table 27: Gripper switching timing on VLABench with $H_{\text{exec}} = 20$, over chunks where the demonstrated gripper command switches exactly once, evaluated on 500 randomly sampled training demonstrations and the 300 held-out demonstrations of Appendix A.5. All values are in % except *Mean error*, which is in action steps (0.1 s each). *Mean error* is the mean absolute offset between the predicted and the demonstrated switch, over chunks in which the model predicts a switch; missed switches are counted in the *Missed* column. *False switch* is the fraction of chunks without a demonstrated switch in which the prediction switches. *Difference* is RACE minus fine-tuning on the same chunks, with 95% percentile confidence intervals of this paired difference in brackets, obtained from 1,000 bootstrap resamples of episodes.

Demos.	Method	Exact↑	±1 step↑	±3 steps↑	Missed↓	Mean error (steps)↓	False switch↓
Training	$\pi_{0.5}$ Fine-tuning	70.1	94.1	97.4	2.5	0.32	0.5
	RACE (ours)	75.2	97.2	99.0	0.9	0.26	0.6
	Difference	+5.1 [3.0, 6.9]	+3.1 [2.2, 4.0]	+1.6 [1.0, 2.2]	−1.6 [−2.2, −1.0]	−0.06 [−0.08, −0.04]	+0.0 [−0.1, 0.2]
Held-out	$\pi_{0.5}$ Fine-tuning	50.6	86.9	93.7	6.3	0.54	0.6
	RACE (ours)	53.8	91.5	95.9	3.9	0.50	0.7
	Difference	+3.2 [−1.2, 7.6]	+4.6 [1.9, 7.1]	+2.2 [0.5, 4.1]	−2.4 [−4.3, −0.9]	−0.04 [−0.09, 0.01]	+0.1 [−0.1, 0.3]

D.5 Decomposing Idle Time in Real-robot Experiments

Tab. 28 decomposes the idle time in Tab. 6 for real-robot experiments, which measures the total time the robot receives no new action command during an episode.

When the robot pauses. With asynchronous execution, each policy call proceeds as follows: the client captures new frames from both cameras, sends the observation to the server, and receives the next chunk, while the robot keeps executing the remaining actions of the current chunk. Since the robot keeps moving during inference, real-time chunking skips the first actions of the new chunk, which correspond to the time elapsed during inference, and executes the rest. We call the time to execute these remaining actions the *execution buffer*: the number of actions left in a chunk when it arrives, divided by the control frequency of 30 Hz. The robot pauses only when the latency of a policy call exceeds this buffer.

Breakdown of the Idle Time. With $H_{\text{exec}} = 5$, about three of the ten actions in a chunk are skipped on arrival, leaving an execution buffer of about seven actions, which is slightly shorter than the latency per call on average. The robot therefore executes the whole chunk before the next one arrives, about seven actions per call rather than five, and pauses at most policy calls; these pauses account for most of the idle time. The measured pause per call is slightly larger than this average margin, because calls that finish within the buffer contribute zero rather than a negative

Table 28: Breakdown of the idle time in Tab. 6 over successful trials. The robot keeps executing the current chunk while the next one is prepared, and pauses only when the latency exceeds this execution buffer, *i.e.*, the execution time of the actions remaining in a chunk when it arrives, after real-time chunking skips the actions elapsed during inference. Latencies are averaged over calls and, unlike Tab. 15 for latency overhead comparisons, include camera capture, network transfer, and real-time chunking (Black et al., 2025). Pause per call exceeds Latency − buffer because calls within the buffer count as zero pause.

	$\pi_{0.5}$		RACE (ours)
	$H_{\text{exec}} = 5$	$H_{\text{exec}} = 20$	$H_{\text{exec}} = 20$
# of policy calls per episode	45	17	17
Actions executed per call	7.0	19.9	20.1
Latency per call	**271 ms**	**285 ms**	**372 ms**
Camera capture	122 ms	112 ms	109 ms
Network	49 ms	64 ms	75 ms
Server	100 ms	109 ms	188 ms
Execution buffer	**234 ms**	**1160 ms**	**1072 ms**
Actions skipped on arrival	3.0	5.3	7.8
Latency − buffer	+37 ms	−875 ms	−700 ms
Pause per call	39 ms	0 ms	0 ms
Idle time	**2.11 s**	**0.40 s**	**0.41 s**
Startup (first camera capture and call)	0.27 s	0.28 s	0.28 s
Pauses after the first call	1.71 s	0.00 s	0.00 s
Control loop	0.13 s	0.12 s	0.13 s

pause. With $H_{\text{exec}} = 20$, the buffer is several times longer than the latency, so no pause occurs after the first policy call. Since the pauses depend on the per-call latency, a large part of which is camera capture in our setup, the chunk length needed to remove them depends on the deployment setup. The remaining idle time comes from two fixed costs that do not depend on the chunk length: the startup, where the robot waits for the first camera capture and the first policy call, and the control-loop overhead, such as timer jitter. RACE has a longer server latency than the fine-tuned $\pi_{0.5}$ because our real-time chunking implementation runs it without CUDA graphs, as its modulation changes at every denoising step. It thus skips more actions on arrival, which shortens its execution buffer. This stems from our implementation rather than from RACE itself, and a CUDA-graph-compatible implementation could reduce this extra latency. Even so, the longer execution buffer hides this latency, yielding the same idle time.

E Additional Details

E.1 Details on Real-robot Experiments

In this section, we describe the task and training demonstrations, robot setup, evaluation metrics, and implementation details for the real-robot results in Tab. 6.

Task Description. We evaluate a pick-and-place task with the instruction *"Pick up the gray bowl next to the plate and place it on the plate."* A trial is successful if the robot grasps the bowl and places it stably on the plate. Across trials, we vary the initial positions of the bowl and the plate and use the same set of initial placements for all methods. For training, we use 30 demonstrations of this task collected on the robotic platform, each recording images from both cameras, the robot state, and the action at 30 Hz, where the action consists of the absolute positions of the six joints and the gripper. Fig. 10 visualizes the task with a successful rollout of RACE at $H_{\text{exec}} = 20$.

Robot Setup. We use an AgileX PiPER robotic arm, a 6-DoF lightweight arm with a 626-mm reach and a 1.5-kg payload, equipped with a gripper. A wrist camera and a front camera provide the visual observations.

Evaluation Metric. We run 50 trials per method and report the success rate, task-completion time, and idle time. Task-completion time is measured from the start of a trial until the task is completed, and idle time is the total time the robot receives no new action command during an



(a) Initial state (b) Approach (c) Grasp (d) Lift (e) Place

Figure 10: Task Visualization. Keyframes of a successful rollout of RACE from the front view.

episode (decomposed in Appendix D.5). Both times are averaged over successful trials. All methods use asynchronous execution with real-time chunking (Black et al., 2025) and predict a chunk twice as long as they execute ($H = 2H_{\text{exec}}$).

Implementation Details. All policies are fine-tuned from the pre-trained $\pi_{0.5}$ base model on the 30 demonstrations for 20K steps. We train the fine-tuned $\pi_{0.5}$ with chunk lengths $H = 10$ and $H = 40$, and RACE with $H = 40$. We use AdamW with gradient clipping at 1.0, a batch size of 32, and a cosine learning-rate schedule with 1K warmup steps, decaying from 2.5×10^{-5} to 2.5×10^{-6}, without EMA. Actions are predicted as joint displacements from the first state of each chunk (the gripper command is absolute) and converted back to absolute joint targets before execution. For RACE, we use the same transition-label and loss settings as in simulation. At deployment, policies run asynchronously and request the next chunk after executing $H_{\text{exec}} = H/2$ actions, *e.g.*, a policy with $H = 40$ requests the next chunk after executing 20 actions, while the remaining 20 actions cover the inference time. On the real robot, H_{exec} thus denotes the configured replanning interval. For $H = 10$, the latency exceeds the execution time of the remaining actions, so the robot executes the whole chunk before the next one arrives, and each call executes about seven actions on average rather than five (Appendix D.5).

E.2 Details on Simulation Speedup Measurement

Simulation Experiments. We report the per-episode inference speedup (Spd↑) in the main comparisons (Tab. 1, Tab. 2, and Tab. 9). For each model, we measure the latency of a single policy call and the average number of policy calls per episode over the episodes that both the model and the $\pi_{0.5}$ baseline complete successfully; the speedup is the ratio of this product for the $\pi_{0.5}$ baseline to that for the model. Formally, for the model θ,

$$\text{Spd}_\theta = \frac{\ell_{\pi_{0.5}} \bar{N}_{\pi_{0.5}}}{\ell_\theta \bar{N}_\theta}, \tag{11}$$

where ℓ_θ is the latency of a single policy call and $\bar{N}_\theta$ is the average number of policy calls per episode over these common successful episodes, where an episode of T steps requires $\lceil T/H_{\text{exec}} \rceil$ policy calls. Similar to PolicyTrim (Wang et al., 2026c), this measure counts only the time spent waiting for the policy and excludes simulator execution time, which depends on the simulator rather than on the policy and does not exist on a real robot. Unlike PolicyTrim, which compares only the number of policy calls, we also account for the per-call latency, since RACE adds a small inference overhead. Spd is thus largely determined by H_{exec} and the per-call latency. With synchronous execution, this reduction directly shortens the pauses between policy calls; with asynchronous execution, longer chunks instead allow the remaining latency to be hidden (Appendix D.5). Beyond deployment, fewer policy calls also reduce the inference cost of large-scale simulation rollouts, such as policy evaluation or data collection for reinforcement learning, although the overall gain there depends on the simulator step time excluded from Spd.

Breakdown and Robustness. Since Spd compares the policy-inference time per episode, a higher value could also come from shorter episodes rather than fewer policy calls. To separate these factors, Tab. 29 breaks down the speedup at $H_{\text{exec}} = 20$ on VLABench over the 127 episodes that all three models complete successfully, which cover 32 of the 50 task–track pairs (59, 22, 19, 8, and 19 episodes from the five tracks). The three models take a similar number of environment steps per episode, so the speedup comes from fewer policy calls rather than shorter episodes. RACE makes slightly more calls than the fine-tuned $\pi_{0.5}$ and adds a small per-call overhead (Appendix B.6), so its speedup is slightly lower. In addition, since Spd is computed only over successful episodes, it could change with which episodes are included. Tab. 30 shows that the speedup remains similar

across different sets of episodes, including all episodes regardless of success. Spd should thus be read together with the success rate, where RACE outperforms the fine-tuned $\pi_{0.5}$ (Tab. 1).

Table 29: Breakdown of the inference speedup at $H_{\text{exec}} = 20$ on VLABench, over the 127 episodes that all three models complete successfully. Spd is computed over all episodes pooled and with each task weighted equally; the last column shows its range across tasks.

Model	Env. steps per episode	Policy calls per episode	Spd↑		
			Pooled	Task-balanced	Range across tasks
$\pi_{0.5}$ Baseline	117.9	23.93	1.00×	1.00×	–
$\pi_{0.5}$ Fine-tuning	112.3	6.10	3.92×	3.89×	3.15–4.70×
RACE (ours)	113.6	6.16	3.74×	3.64×	2.54–4.53×

Table 30: Inference speedup at $H_{\text{exec}} = 20$ on VLABench under different sets of episodes. The first row is the setting used in the main comparisons.

Episode set	# Episodes	$\pi_{0.5}$ Fine-tuning	RACE (ours)
Common successes with the $\pi_{0.5}$ baseline (used)	228 / 256	3.82×	3.68×
Common successes of all three models	127	3.92×	3.74×
with each task weighted equally	127	3.89×	3.64×
All episodes, regardless of success	2,500	4.00×	3.94×

Real-robot Experiments. On real robots, we instead report task-completion time and idle time (Sec. 4.5). Task-completion time is measured from the start of a trial until the task is completed, and idle time is the total time the robot receives no new action command during an episode (decomposed in Appendix D.5).

E.3 Details on Other VLA Baselines

We describe how RACE is applied to the two backbones in Tab. 14. **1) GR00T-N1.6:** like $\pi_{0.5}$, it generates actions with a flow-matching action head conditioned on VLM features, so RACE transfers with a single change in how the prior is injected. We use an auxiliary one-step denoising pass to obtain action features and predict the transition-timing prior together with the cached VLM features. Unlike the action expert of $\pi_{0.5}$, which uses adaptive RMSNorm, the action head of GR00T-N1.6 is a diffusion transformer with adaptive LayerNorm (AdaLN) (Peebles & Xie, 2023) conditioned on the denoising timestep. We therefore inject the prior into every subsequent denoising step through these AdaLN layers, modulating their scale and shift token-wise as in Eq. 7. **2) OpenVLA-OFT:** it regresses the chunk with a single language model and has no denoising process. We first forward only the image and instruction tokens and cache their hidden states, and then replace the auxiliary denoising pass by an unmodulated forward pass of the action placeholder tokens to obtain the action features for the prediction head, and finally run them a second time with the prior injected at every RMSNorm layer of the action positions. Since there is no denoising loop, there is a single modulated pass and no per-step gate.

E.4 Details on Action Error Measurement

We measure action errors on VLABench demonstrations, since ground-truth actions are available only for demonstration states and not for the states visited during rollouts. At each policy call, the policy predicts a chunk from the current observation, and we compare the final output of the denoising process with the ground-truth actions of the demonstration over the same steps. The action error of each action is the mean squared error (MSE) between the predicted and ground-truth actions in the normalized action space, averaged over all seven action dimensions. Since transition steps also involve larger changes in action, comparing errors at transition and non-transition actions indicates where errors concentrate rather than separating the causes of execution failures.

Offset from transition point. For Fig. 2, we align the errors at transitions. For each PELT change point at position c within a chunk, we take the error of the action at position $c + \delta$ for offsets $\delta \in \{-8, \ldots, 8\}$, keeping only offsets that lie within the chunk, and average it over all change points. A chunk containing multiple change points contributes each of them separately. An offset of $\delta = 0$ corresponds to the transition point.

Position within chunk. For Fig. 4, we group the actions by their position within the chunk. At each position, we average the error of actions that coincide with a change point (*i.e.*, transition error) and that of the remaining actions (*i.e.*, non-transition error).

Matching local action variation. Because PELT detects changes in action dynamics, larger errors at transitions could reflect larger local action variations. To examine this, we compare the error at each transition with those at non-transition positions, at least three steps from any change point, matched by position within the chunk and by decile of the second-order action difference $c_h = \|\mathbf{a}_h - 2\mathbf{a}_{h-1} + \mathbf{a}_{h-2}\|_2$ of the demonstrated actions, on 500 VLABench training demonstrations with $H_{\text{exec}} = 20$ for $\pi_{0.5}$ and RACE as in Fig. 2 (Tab. 31). Transition errors remain about 1.3 times those at the matched positions for both policies, so the error gap persists after matching on chunk position and this measure of local action variation.

Table 31: Ratio of the action error at transitions to that at non-transition positions matched by position within the chunk and by decile of the second-order action difference c_h, on VLABench with $H_{\text{exec}} = 20$. A ratio of 1 means no extra error at transitions. Brackets denote 95% episode-level bootstrap confidence intervals.

	$\pi_{0.5}$	RACE (ours)
Error ratio (transition / matched)	1.34 [1.15, 1.56]	1.30 [1.05, 1.62]

E.5 Details on Dataset and Evaluation Protocols

VLABench. VLABench (Zhang et al., 2025) is a MuJoCo-based benchmark for long-horizon, language-conditioned manipulation. It contains ten tasks with 500 scripted demonstrations per task, for a total of 5,000 training demonstrations. Each task is evaluated under five tracks: *in-distribution*, which follows the training distribution; *unseen categories*, which changes the object categories; *commonsense*, which requires reasoning beyond explicit instructions; *semantic instructions*, which varies the linguistic specification; and *unseen textures*, which changes object appearance. For each track, we run 50 episodes per task, resulting in 500 episodes per track and 2,500 evaluation episodes in total. We report success rate (SR) for full task completion and progress score (PS) for partial completion. We first average results across the ten tasks within each track and then report the mean across the five tracks.

RoboCasa-H50. RoboCasa (Nasiriany et al., 2024) provides visually diverse kitchen environments with randomized layouts, fixtures, objects, and object placements. We use the RoboCasa-H50 setting, which contains 50 human-teleoperated demonstrations for each of 25 kitchen-manipulation tasks, totaling 1,250 training demonstrations. The tasks cover eight categories: pick-and-place, doors, drawers, sinks, stoves, coffee machines, microwaves, and navigation. We evaluate each task over 50 episodes, yielding 1,250 evaluation episodes per model. Success is determined by the task-specific completion predicate. We average success rates within each category and compute the overall score weighted by the number of tasks per category.

LIBERO. We evaluate on the four standard LIBERO suites (Liu et al., 2023): Spatial, Object, Goal, and Long. Each suite contains ten tasks with 50 human-teleoperated demonstrations per task. Thus, we train on 2,000 demonstrations across the 40 tasks. For evaluation, we run 50 episodes per task using the benchmark-provided initial states, resulting in 500 episodes per suite and 2,000 episodes per model. A rollout is successful when the task-specific goal is achieved within the episode horizon. We report the mean success rate for each suite and the average across all four suites.

E.6 Pseudocode for Training and Inference

In this section, to provide the exact training and inference procedures of RACE, we present their pseudocode in Alg. 1 and Alg. 2. Unlike Sec. 3.1, which omits it for brevity, we write the flow time $s \in [0, 1]$ explicitly; it runs from the noise ($s = 0$) to the action chunk ($s = 1$).

Training Procedure. Each training step first runs the auxiliary pass on the noise sample without modulation, from which the prediction head estimates the prior and $\mathcal{L}_{\text{timing}}$ and $\mathcal{L}_{\text{aux}}$ are computed. The full pass then samples the flow time s continuously as in $\pi_{0.5}$ (Physical Intelligence et al., 2025) and conditions the action expert on the jittered target with the gate α^k of the denoising step whose interval $[\frac{k-1}{K}, \frac{k}{K})$ contains s.

Algorithm 1 Training step of RACE

Input: observation $\mathbf{o}_t$, instruction ℓ, ground-truth chunk $\mathbf{A}_t$, soft label $\mathbf{p}_t$
Output: training loss $\mathcal{L}$

$\mathbf{z}_t \leftarrow \mathrm{VLM}(\mathbf{o}_t, \ell)$ ▷ frozen; cached for both passes
$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Auxiliary pass: unmodulated, at $s = 0$
$\mathbf{F}_t \leftarrow$ hidden states of the action expert on $(\boldsymbol{\epsilon}, \mathbf{z}_t, s = 0)$
$\mathbf{v}_t^{\mathrm{aux}} \leftarrow$ velocity predicted from $\mathbf{F}_t$
$\hat{\mathbf{p}}_t \leftarrow \mathrm{Head}(\mathbf{F}_t, \hat{\mathbf{z}}_t)$ ▷ $\hat{\mathbf{z}}_t$: final-layer features in $\mathbf{z}_t$
$\mathcal{L}_{\mathrm{timing}} \leftarrow \mathrm{BCE}(\hat{\mathbf{p}}_t, \mathbf{p}_t)$
$\mathcal{L}_{\mathrm{aux}} \leftarrow \|\mathbf{v}_t^{\mathrm{aux}} - (\mathbf{A}_t - \boldsymbol{\epsilon})\|^2$
Full pass: modulated, at a sampled flow time
$s \sim p(s)$ ▷ same distribution as $\pi_{0.5}$
$k \leftarrow \lfloor Ks \rfloor + 1$ ▷ denoising step whose interval contains s
$\mathbf{A}_t^s \leftarrow (1 - s)\,\boldsymbol{\epsilon} + s\,\mathbf{A}_t$
$\tilde{\mathbf{p}}_t \leftarrow \mathrm{Jitter}(\mathbf{p}_t)$ ▷ teacher forcing; ± 1 step w.p. 0.5
$\mathcal{L}_{\mathrm{full}} \leftarrow \|v_\theta(\mathbf{A}_t^s, \mathbf{z}_t, s, \alpha^k \tilde{\mathbf{p}}_t) - (\mathbf{A}_t - \boldsymbol{\epsilon})\|^2$
return $\mathcal{L}_{\mathrm{full}} + \lambda_{\mathrm{aux}}\mathcal{L}_{\mathrm{aux}} + \lambda_{\mathrm{timing}}\mathcal{L}_{\mathrm{timing}}$

Inference Procedure. At each policy call, the auxiliary pass predicts the prior once, and the k-th of the K denoising steps starts at flow time $s = (k-1)/K$ and uses the gate α^k.

Algorithm 2 Inference of RACE at policy call t

Input: observation $\mathbf{o}_t$, instruction ℓ, number of denoising steps K
Output: action chunk $\mathbf{A}_t$

$\mathbf{z}_t \leftarrow \mathrm{VLM}(\mathbf{o}_t, \ell)$ ▷ cached for both passes
$\mathbf{A}_t^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Auxiliary pass: unmodulated, at $s = 0$
$\mathbf{F}_t \leftarrow$ hidden states of the action expert on $(\mathbf{A}_t^0, \mathbf{z}_t, s = 0)$
$\hat{\mathbf{p}}_t \leftarrow \mathrm{Head}(\mathbf{F}_t, \hat{\mathbf{z}}_t)$ ▷ $\hat{\mathbf{z}}_t$: final-layer features in $\mathbf{z}_t$
Full denoising: modulated by the prior
for $k = 1, \ldots, K$ **do** ▷ step k starts at $s = (k-1)/K$
  $\mathbf{A}_t^k \leftarrow \mathbf{A}_t^{k-1} + \frac{1}{K}\, v_\theta\big(\mathbf{A}_t^{k-1}, \mathbf{z}_t, \frac{k-1}{K}, \alpha^k \hat{\mathbf{p}}_t\big)$
return $\mathbf{A}_t^K$

F Further Discussion

F.1 Limitations

RACE makes longer action chunks more reliable by learning transition timing and conditioning action generation on it, but several limitations remain. **(1) Imperfect transition timing:** The prediction head misses some transitions, especially on held-out demonstrations and for longer chunks (Appendix D.1), and its labels are pseudo-labels from change-point detection rather than human annotations. Nevertheless, even with this imperfect prior, RACE outperforms naïve fine-tuning without transition timing at longer chunks and reduces errors at transitions (Tab. 1; Fig. 4). **(2) Degradation of very long chunks:** The success rate of RACE declines more steeply beyond $H_{\mathrm{exec}} = 20$ and falls below the baseline at $H_{\mathrm{exec}} = 40$ (Appendix B.2), since it reduces errors around transitions but not the error growth of open-loop execution itself. Still, up to $H_{\mathrm{exec}} = 20$, *i.e.*, four times the chunk length of the $\pi_{0.5}$ baseline, RACE remains comparable to or better than the baseline on all three benchmarks. **(3) Reliance on iterative denoising:** RACE is designed to condition iterative denoising, and its gain is smaller on a VLA without a denoising process (Appendix B.5). Even so, it outperforms the fine-tuned variant at every chunk length on both GR00T-N1.6 (NVIDIA et al., 2025) and OpenVLA-OFT (Kim et al., 2025).

F.2 Broader Impact

This work aims to make robot manipulation with VLA models more continuous by enabling longer action chunks, which reduces the number of policy calls per task and the idle time between them. At the same time, longer chunks mean that the robot executes more actions without new observations, which gives it fewer chances to react to unexpected changes in its environment, such as a person entering the workspace. In addition, RACE is trained on demonstrations, so it may inherit biases or unsafe behaviors present in the training data. Our experiments are limited to simulation benchmarks and a controlled laboratory setting. Deploying such policies in real-world environments, especially near people, therefore requires additional safeguards, such as safety monitoring, collision checking, and human supervision.