KPE-AC: KINETIC PATH ENERGY FOR ADAPTIVE CHUNKING IN VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Action chunking enables a vision-language-action (VLA) policy to predict multiple future actions in a single inference, improving efficiency while maintaining temporal coherence. A main challenge is selecting the execution horizon—that is, how many actions from the generated chunk to execute before querying the policy again. Short horizons allow frequent observation updates and more precise control but require more computation, whereas long horizons reduce inference frequency but delay responses to changes in the environment. Recent works have proposed adaptive execution-horizon selection methods, but this problem is still underexplored. In this work, we look inside the generation process of a single action chunk to guide adaptive action chunking. Inspired by Kinetic Path Energy analysis in image generation, we introduce Kinetic Path Energy-guided Adaptive Chunking (KPE-AC). We adapt KPE to action generation by measuring deviations from constant-velocity transport, yielding correction KPE. We analyze how this energy reflects uncertainty in future actions and leverage it to determine how many actions to execute. Beyond looking inside a single action chunk, we assess Cross-Chunk Consistency (CCC) by comparing actions predicted for the same future time steps in the previous and current action chunks. CCC adjusts the execution horizon while leaving the predicted actions unchanged. Experiments in simulation and real-world manipulation demonstrate the effectiveness of KPE-AC across multiple flow-based VLA backbones.

1 INTRODUCTION

Building on advances in vision-language models (VLMs), vision-language-action (VLA) models bring visual-language understanding into robot control and show strong performance across a range of manipulation tasks (Brohan et al., 2023; Kim et al., 2024; Octo Model Team et al., 2024; Black et al., 2024; 2025a). Recent VLAs leverage action chunking: the policy predicts a sequence of future actions and executes multiple actions before replanning. This reduces policy queries and helps produce smooth, temporally consistent motion (Zhao et al., 2023; Chi et al., 2023; Black et al., 2024). Action chunking thus involves both predicting a sequence and deciding how much of it to execute. The execution horizon determines how long the robot follows its current plan before incorporating new observations. However, many VLA policies execute a fixed number of actions before querying the policy again (Octo Model Team et al., 2024; Black et al., 2024; 2025a; NVIDIA et al., 2025; Kim et al., 2025; Shukor et al., 2025).

Choosing a fixed execution horizon creates a clear trade-off. Short horizons provide frequent feedback for precise control but require more policy queries. Long horizons reduce inference frequency but delay responses to environmental changes. The preferred horizon also changes during a task: simple movements permit longer execution, whereas precise interactions require more frequent replanning. Motivated by this trade-off associated with fixed execution horizons, recent methods adapt the horizon at inference time. Adaptive Action Chunking (AAC) estimates action entropy from multiple stochastic action chunks generated under the same observation (Liang et al., 2026), while Auto-Horizon uses action self-attention to estimate how far the policy's predictions remain reliable (Wang et al., 2026a). These methods, however, offer limited performance gains, and AAC incurs additional inference cost from generating multiple action chunks at each query. To address these limitations, we take a different perspective. *We look inside the generation process of a single action chunk* and leverage predicted velocities during denoising to guide execution-horizon selection.

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

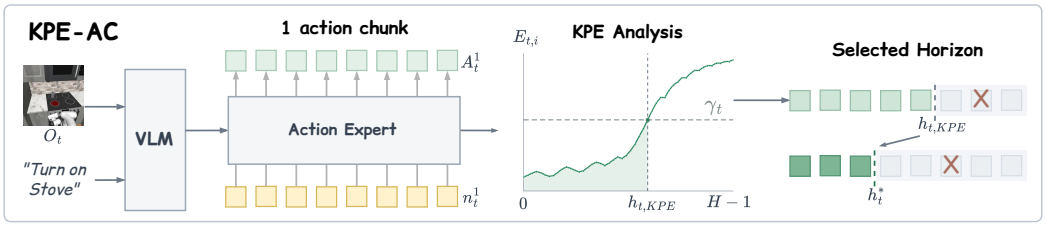

Figure 1: **Overview of our KPE-AC**. We look inside the generation process of a single action chunk and leverage late-stage correction KPE energy to select a candidate execution horizon.

In flow-based VLAs, the action expert gradually transforms noise into an action chunk through a sequence of velocity-guided denoising steps (Lipman et al., 2022; Black et al., 2024; 2025a). The predicted velocities describe the updates to each future action along the way, providing a view of how the final prediction is formed. Inspired by Kinetic Path Energy (KPE) analysis in image generation (Li et al., 2026; 2025), we adopt KPE in the action generation domain, using the energy accumulated along the denoising path to guide execution-horizon selection.

Directly applying raw KPE to action generation, however, can be misleading. A large displacement from the initial noise to the final action produces a large KPE value even when generation proceeds at a constant velocity. We therefore measure how much each action's predicted velocity deviates from its average over the generation process and refer to it as **Correction KPE**. We investigate this energy under a conditional Gaussian model to show how its expected value relates to action uncertainty. We also consider when these corrections occur. Early in generation, the model forms an initial action sequence from noise, while later updates refine the details of that sequence. We therefore focus on late-stage corrections to quantify the stability of action predictions. Experiments with a pre-trained VLA demonstrate a positive association between late correction energy and variation across independent predictions under the same observation. These findings support using late correction energy to choose the execution horizon.

Beyond looking inside a single action chunk, we assess **Cross-Chunk Consistency (CCC)** between consecutive plans. Even when the current chunk exhibits small late-stage corrections, its predictions may differ substantially from the previous plan. Correction energy does not directly measure this disagreement, since it only examines the current generation. So we compare actions predicted for the same future time steps across the two chunks and use CCC to adjust the execution horizon. Larger differences shorten the horizon and prompt earlier observation updates, while the predicted actions remain unchanged.

We evaluate KPE-AC on RoboCasa (Nasiriany et al., 2024), LIBERO (Liu et al., 2023), and a real-world environment. KPE-AC achieves 49.36% success across 28 RoboCasa tasks with $\pi_{0.5}$, outperforming the evaluated adaptive baselines. It also improves success across π_0, $\pi_{0.5}$, and GR00T N1.5 on LIBERO and over fixed-horizon baselines in real-world Pick-and-Place and Target Placement.

2 Related Work

Vision-Language-Action Models. VLA models connect vision-language understanding to robot control through different action formulations. RT-2 and OpenVLA predict discrete action tokens from pretrained VLMs (Brohan et al., 2023; Kim et al., 2024). Diffusion Policy generates continuous action sequences through denoising (Chi et al., 2023), while π_0, $\pi_{0.5}$, and GR00T N1 use flow-based action experts conditioned on VLM representations (Black et al., 2024; 2025a; NVIDIA et al., 2025). X-VLA further supports policy sharing across robots through embodiment-specific soft prompts (Zheng et al., 2026). At execution time, ACT and RTC maintain continuity between chunks through temporal ensembling and action inpainting, respectively (Zhao et al., 2023; Black et al., 2025b), while REMAC combines masked action training with prefix-preserved sampling (Wang et al., 2026b). Our method focuses on selecting the execution horizon from the action expert's generation dynamics without further training.

Adaptive Action Chunking. Action chunking generates a sequence of future actions, while the execution horizon specifies how many actions to execute before observing and replanning. To bal-

ance continuity with responsiveness to new observations, BID uses backward coherence with the previous plan to help select among candidate trajectories (Liu et al., 2024), while SGAC combines self-guidance from past observations with selective action updates (So et al., 2025). Other methods adapt the execution horizon directly. AAC selects this horizon using action entropy estimated from multiple chunks generated under the same observation, at the cost of additional sampling (Liang et al., 2026). AutoHorizon estimates the execution horizon from action-token self-attention (Wang et al., 2026a). Adaptive Action Acceptance (A^3) selects a representative chunk from multiple samples and verifies its executable prefix through conditional re-decoding (Chen et al., 2026). KPE-AC instead utilizes velocities already computed during a single generation and uses CCC to adjust the execution horizon, without re-decoding or modifying the predicted actions.

Flow Matching and Kinetic Path Energy. Flow matching and rectified flow learn transport paths from noise to data (Lipman et al., 2022; Tong et al., 2024; Liu et al., 2022). Kinetic-energy objectives have been studied for generative transport, neural ODEs, and stochastic policies (Shaul et al., 2023; Finlay et al., 2020; Neklyudov et al., 2023; Lv et al., 2026). In image generation, KPE relates trajectory energy to semantic quality and local data density (Li et al., 2025; 2026), while FlowGuard uses velocity changes to assess generation reliability (Li & Boström, 2025). We adapt this perspective to robot actions by measuring deviations from constant-velocity transport and using late-stage correction energy for execution-horizon selection.

3 PRELIMINARIES

3.1 FLOW-BASED VLA

Flow-based VLA models use a vision-language model (VLM) to extract vision-language features $\mathbf{c}_t$ and a flow-matching action head v_θ to generate robot actions (Black et al., 2024; 2025a). At timestep t, a ground-truth action chunk is defined as $\mathbf{A}_t = \{\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+H-1}\} \in \mathbb{R}^{H\times D}$, where H is the action chunk length and D is the action dimension. During training, we sample a flow-matching timestep $\tau \in [0, 1]$ and noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The noised action chunk is computed as $\mathbf{A}_t^\tau = \tau\mathbf{A}_t + (1-\tau)\boldsymbol{\epsilon}$. Conditioned on the vision-language features $\mathbf{c}_t$ and, when available, the robot state $\mathbf{s}_t$, the action head $v_\theta(\mathbf{A}_t^\tau, \tau, \mathbf{c}_t, \mathbf{s}_t)$ learns to predict the target velocity $\mathbf{A}_t - \boldsymbol{\epsilon}$ by minimizing the flow-matching loss (Lipman et al., 2022; Liang et al., 2026):

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}\left[\left\|v_\theta(\mathbf{A}_t^\tau, \tau, \mathbf{c}_t, \mathbf{s}_t) - (\mathbf{A}_t - \boldsymbol{\epsilon})\right\|_F^2\right]. \tag{1}$$

At inference time, the model starts from random noise $\mathbf{A}_t^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and generates an action chunk through K iterative denoising steps. Using forward Euler integration with $\Delta\tau = 1/K$, each step updates the action chunk as

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

The final output $\mathbf{A}_t^1$ is an action chunk of length H. Many existing VLA policies execute the first h actions from each chunk before querying the policy again, where the execution horizon $h \le H$ is fixed in advance (Black et al., 2024; Kim et al., 2025). Longer horizons reduce policy queries but limit responsiveness, while shorter horizons enable more precise control through frequent observations at a higher computational cost.

3.2 KINETIC PATH ENERGY IN IMAGE GENERATION

Recent work on image generation examines flow matching through kinetic energy and optimal transport (Shaul et al., 2023; Tong et al., 2024; Li et al., 2026). In this view, a sample is treated as a particle moving from noise to data under a learned velocity field. Rather than examining only the final image, Kinetic Path Energy (KPE) measures the accumulated kinetic effort along its generation trajectory (Li et al., 2026). For a sampling trajectory $\mathbf{x}^\tau$, KPE is defined as

$$E_{\mathrm{KPE}} = \frac{1}{2}\int_0^1 \|v_\theta(\mathbf{x}^\tau, \tau)\|_2^2 \,\mathrm{d}\tau, \tag{3}$$

where we adopt the convention of unit mass ($m = 1$), standard in the free-particle action formulation (Goldstein et al., 2002). In image generation, (Li et al., 2026) shows that higher KPE is

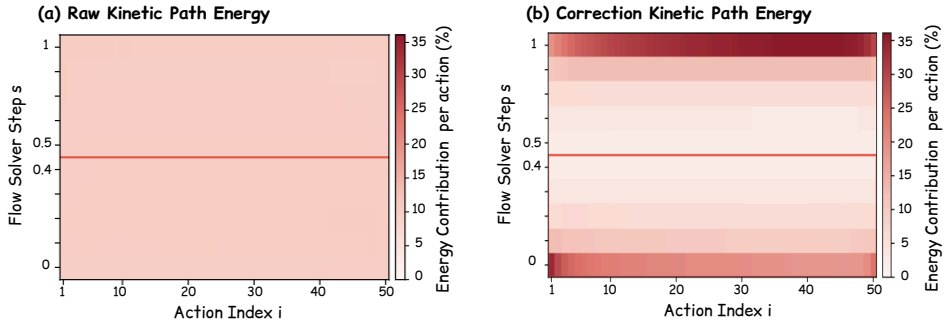

Figure 2: **KPE distributions.** Raw KPE (a) and correction KPE (b) across action positions and denoising steps. Values show the percentage of each action's total energy contributed by each step, averaged over RoboCasa rollouts. The red horizontal line separates the first and final five denoising steps.

associated with clearer class-specific features and lower local support in representation space. Its empirical flow-matching analysis also links excessive late-stage energy to memorization, indicating that higher energy does not always imply better generation. We bring this kinetic perspective from image generation to action generation, examining how each future action changes during denoising.

4 Adaptive Chunking with Single Action Generation

4.1 Kinetic Path Energy in Vision-Language-Action Models

We extend the kinetic perspective from image generation to action generation in flow-based VLA models. Using KPE for horizon selection raises two questions: *(1) Can KPE be used directly to guide action execution?* and *(2) Which denoising steps should we examine?*

From raw KPE to correction energy. KPE measures the total kinetic effort along a generation trajectory (Li et al., 2026). However, even a straight, constant-velocity path can have high KPE when the initial noise and final action are far apart. To separate this overall displacement from deviations along the path, we adopt the constant-velocity reference used in rectified flow (Liu et al., 2022). For each future action i, this reference is its mean velocity over the full trajectory, $\bar{\mathbf{v}}_{t,i} = \int_0^1 \mathbf{v}_{t,i}^\tau \, \mathrm{d}\tau$. It represents constant-velocity motion between the same endpoints. We subtract this reference and define the correction energy as

$$E_{t,i}^{\mathrm{corr}} = \frac{1}{2} \int_0^1 \left\| \mathbf{v}_{t,i}^\tau - \bar{\mathbf{v}}_{t,i} \right\|_2^2 \, \mathrm{d}\tau, \tag{4}$$

where $\mathbf{v}_{t,i}^\tau$ is the predicted velocity of action i at denoising time τ. Figure 2 shows that raw energy is distributed almost uniformly across denoising steps, while correction energy reveals larger deviations at the beginning and end of generation. Additional examples are provided in Appendix A.

Relation to action uncertainty. Even under the same observation, a flow-based VLA produces different action predictions from different initial noise samples. The variation of these predictions describes the action uncertainty we seek to capture. Correction energy represents this variation across predictions relative to the changes made within a single generation. To examine this relationship, consider a scalar action $a \mid o \sim \mathcal{N}(\mu, \sigma^2)$, where σ measures the spread of possible actions. Under the independent noise–action interpolation in Section 3.1 and the optimal flow-matching velocity field, subtracting the trajectory's mean velocity removes the contribution of μ. The remaining correction energy, integrated over the full trajectory, satisfies

$$\mathbb{E}_\xi \left[E^{\mathrm{corr}} \mid o \right] = \left(1 - \frac{\pi}{4} \right) \sigma, \qquad \xi \sim \mathcal{N}(0, 1), \tag{5}$$

Thus, in this Gaussian setting, greater action uncertainty leads to higher correction energy in expectation. The full derivation is provided in Theorem B.1 of Appendix B. We test this connection in a trained VLA by measuring action uncertainty across 32 independent predictions under the

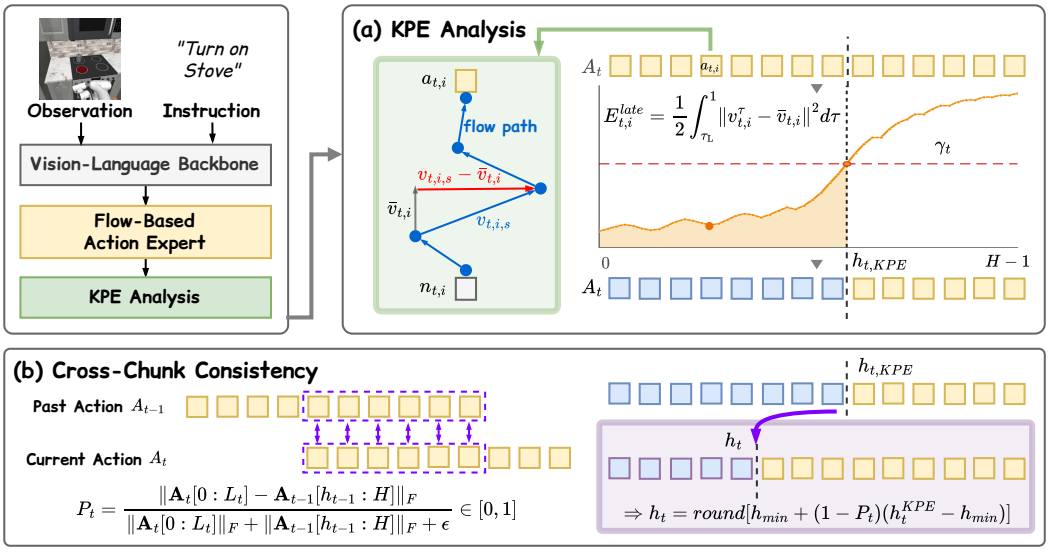

Figure 3: **KPE-guided adaptive action chunking.** KPE measures action-wise correction relative to direct transport in one flow generation and selects a candidate horizon h_t^{KPE}. CCC compares the time-aligned overlap of consecutive chunks to adjust this horizon to h_t, leaving the predicted actions unchanged.

same observation. Correction energy from a separate, single generation identifies the top 20% most variable action positions with an AUROC of 0.601, above the random-ranking reference of 0.5 (Appendix B.1, Figure 10). This supports using within-chunk energy differences as a relative uncertainty signal for execution.

Focusing on late denoising. To use correction energy for horizon selection, we also consider when the corrections occur. Early in denoising, the model is still forming an action sequence from noise, so substantial corrections can arise while constructing the initial prediction. Later corrections occur as the sequence approaches the final actions that will be executed. This motivates focusing on the adjustments made during the final stage of generation. Figure 2(b) shows that correction energy rises again near the end of denoising, revealing these late adjustments. We accumulate their energy as

$$E^{\text{late}}_{t,i} = \frac{1}{2}\int_{\tau_{\text{L}}}^{1} \left\|\mathbf{v}^{\tau}_{t,i} - \bar{\mathbf{v}}_{t,i}\right\|_2^2 \, \mathrm{d}\tau, \tag{6}$$

where τ_{L} marks the start of the late-denoising interval, and the reference velocity remains the mean over the full trajectory. This yields an action-wise profile of final-stage corrections for horizon selection. Appendix G.2 details the comparison with the full denoising trajectory.

4.2 Adaptive Action Chunking with Kinetic Path Energy

We use the late-correction energy $E^{\text{late}}_{t,i}$ as a signal of action uncertainty to determine how many actions to execute. We compare energies within a single action chunk using an adaptive threshold γ_t, and execute the chunk up to, but not including, the first action whose energy exceeds this threshold. To define this threshold, we compute the maximum KPE energy $E^{\max}_t$ and the mean KPE energy $\bar{E}_t$ over a single action chunk $\mathbf{A}^1_t$. We define their difference as $\Delta_t = E^{\max}_t - \bar{E}_t$ and use it to adjust the threshold relative to the mean energy:

$$\gamma_t = \begin{cases} \bar{E}_t \left(1 + \dfrac{\Delta_t}{E^{\max}_t}\right), & E^{\max}_t > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

This threshold accounts for the mean correction KPE energy of the chunk while remaining between its mean and maximum energies. We then select the candidate execution horizon as

$$h^{\text{KPE}}_t = \begin{cases} \min\{i \in \mathcal{I} : E^{\text{late}}_{t,i} > \gamma_t\}, & \text{if such an index exists,} \\ H, & \text{otherwise,} \end{cases} \tag{8}$$

where $\mathcal{I} = \{h_{\min}, \ldots, H-1\}$ is the candidate range and $h_{\min}$ is the minimum execution horizon. If no action exceeds the threshold, we retain the full chunk as the candidate execution horizon.

4.3 Cross-Chunk Consistency

Correction KPE energy provides a signal of action uncertainty within a single generation process. However, this internal view does not directly capture how predictions change across observations. Prior work uses comparisons between old and new predictions to guide plan updates (So et al., 2025) and emphasizes continuity across chunks and alignment with current observations (Black et al., 2025b; Wang et al., 2026b). We therefore introduce Cross-Chunk Consistency (CCC) to complement correction KPE energy with a signal of temporal uncertainty across generations.

At query t, we compare actions predicted for the same future time steps in the previous and current chunks. After executing h_{t-1} actions from the previous chunk, the overlapping interval has length $L_t = H - h_{t-1}$. For $L_t > 0$, we define the aligned blocks as $\mathbf{A}_{t-1}[h_{t-1} : H]$ and $\mathbf{A}_t[0 : L_t]$. We measure their difference across the entire overlapping interval using

$$P_t = \frac{\|\mathbf{A}_t[0 : L_t] - \mathbf{A}_{t-1}[h_{t-1} : H]\|_F}{\|\mathbf{A}_t[0 : L_t]\|_F + \|\mathbf{A}_{t-1}[h_{t-1} : H]\|_F + \epsilon}. \tag{9}$$

where $\epsilon > 0$ ensures numerical stability. This normalization gives $P_t \in [0, 1]$, with larger values indicating less consistent predictions across the two chunks. If no previous chunk or no overlapping interval is available, we set $P_t = 0$.

We use P_t to adjust the execution horizon rather than modify the generated actions. Specifically, the final horizon is $h_t = \text{round}\big[h_{\min} + (1 - P_t)(h_t^{\text{KPE}} - h_{\min})\big]$. Consistent predictions preserve the KPE candidate horizon, whereas larger differences shorten it toward $h_{\min}$, allowing the policy to observe and replan sooner. In this way, correction KPE energy and CCC guide execution through within-generation corrections and cross-chunk prediction changes, respectively. Algorithm 1 in Appendix C summarizes the online procedure.

5 Experiments

Our experiments examine four questions: *(1) Can KPE-AC improve task success using a single generated action chunk per query? (2) Do these benefits transfer across flow-based VLA backbones? (3) Does adaptive execution improve real-world manipulation? (4) What do KPE and CCC each contribute to horizon selection?* We evaluate success and inference demand on RoboCasa, transfer across three VLA backbones on LIBERO, and physical manipulation on an SO-101 robot, followed by component and energy-signal ablations.

5.1 Simulation Experiments

Benchmark. We evaluate $\pi_{0.5}$ on 28 RoboCasa tasks (Nasiriany et al., 2024; Black et al., 2025a), comparing KPE-AC with execution horizons $h \in \{5, 10, 30, 50\}$, AAC with 20 candidates per query (Liang et al., 2026), and AutoHorizon (Wang et al., 2026a). All methods share the same checkpoint and are evaluated for 50 episodes per task (1,400 total). The four task groups contain seven tasks each, giving them equal weight in the overall score. Evaluation and compute details are in Appendices D.1 and D.2.

Results. Table 1 reveals different execution needs across task groups, with no single fixed horizon performing best in every group. Neither the shortest nor the longest horizon is best in any group, indicating that more frequent observations alone do not ensure better execution. KPE-AC achieves 49.36% overall success, the highest among the evaluated adaptive-horizon methods. These results with a shared checkpoint show that signals from a single generation support adaptive execution without a separate horizon choice for each task group.

Horizon Selection and Inference Demand. The inference comparison clarifies this result (Figure 11 in Appendix D.3). Generated chunks include all candidates: each AAC query contributes 20 generations, whereas KPE-AC and AutoHorizon contribute one. Costs are pooled over successful and failed episodes. KPE-AC improves success over AAC with fewer generated chunks

Method	Success Rate (%) ↑				
	Relocation	Container	Rotation	Button	Overall
Fixed horizon					
$\pi_{0.5}$ ($h = 5$)	54.57	47.14	38.00	22.29	40.50
$\pi_{0.5}$ ($h = 10$)	<u>61.14</u>	51.43	**43.71**	28.00	46.07
$\pi_{0.5}$ ($h = 30$)	56.86	<u>58.00</u>	<u>42.29</u>	**35.43**	<u>48.14</u>
$\pi_{0.5}$ ($h = 50$)	54.00	56.86	40.29	24.57	43.93
Adaptive horizon					
AAC ($N = 20$)	50.57	57.14	35.43	26.29	42.36
AutoHorizon	56.00	56.57	36.29	28.57	44.36
KPE-AC (ours)	**64.86**	**60.57**	<u>42.29</u>	<u>29.71</u>	**49.36**

Table 1: **Success rates (%) on 28 RoboCasa tasks.** h denotes the execution horizon. **Bold** and <u>underlined</u> values mark the best and second-best results per column.

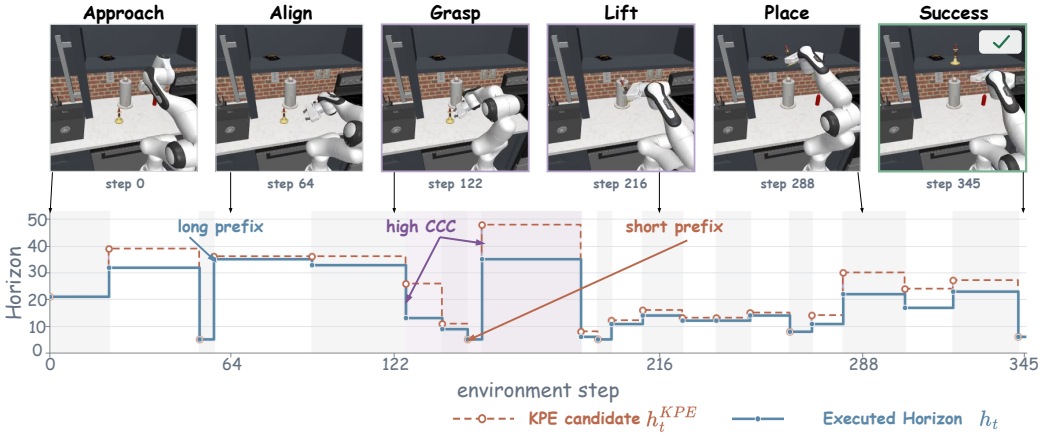

Figure 4: Horizon adaptation during a `PickPlaceCounterToCabinet` rollout. Top: observations. Bottom: KPE candidate (orange) and executed (blue) horizons.

and lower estimated compute per 100 executed steps. AutoHorizon uses fewer queries than KPE-AC but has lower success, indicating that query count alone does not explain the outcome. KPE-AC's horizons span a broad range, and Figure 4 shows short and long prefixes within the same `PickPlaceCounterToCabinet` rollout. Where the two curves coincide, CCC retains the KPE candidate; where they separate, disagreement between consecutive plans brings the next observation forward. The figure thus distinguishes the prefix suggested by the current generation from the adjustment based on consistency across observations, showing how both signals determine the final execution length.

5.2 Transfer Across Flow-Based VLAs

Backbones and Evaluation. We evaluate π_0, $\pi_{0.5}$, and GR00T N1.5 (Black et al., 2024; 2025a; NVIDIA et al., 2025) on LIBERO's Spatial, Object, Goal, and Long suites (Liu et al., 2023). Transfer here means reusing the execution rule on existing policies, without training a horizon predictor for each backbone. To isolate the effect of execution strategy, we reproduce all baselines, including AAC and AutoHorizon, using the same checkpoint and evaluation protocol as KPE-AC within each backbone. Each suite contains ten tasks, evaluated with three seeds and 50 episodes per task and seed, totaling 6,000 episodes per method and backbone. Checkpoint details and a separate comparison with published results under their original settings are provided in Appendix E.

Transfer Across Backbones. Table 2 shows that the effect of horizon adaptation depends on both the backbone and the suite: AAC and AutoHorizon improve some settings but fall below their baselines in others. KPE-AC improves all four suites for each backbone, including the Long suite, where every baseline has its lowest success. For π_0 and $\pi_{0.5}$, gains are larger on Goal and Long than on Spatial and Object. The benefit therefore extends to settings where the base policy leaves more room for improvement. The native chunk lengths are 50, 10, and 16 for π_0, $\pi_{0.5}$, and GR00T, respectively;

Table 2: **Backbone transfer on LIBERO.** Success rates (%) with percentage-point changes from each baseline in parentheses. **Bold** and underlined values mark the best and second-best results per suite within each backbone.

	Suite			
Method	Spatial	Object	Goal	Long
π_0 baseline	95.9	97.1	92.8	80.3
π_0 + AAC ($N = 20$)	90.9 (-5.0)	96.7 (-0.4)	91.3 (-1.5)	81.7 (+1.4)
π_0 + AutoHorizon	81.6 (-14.3)	89.3 (-7.8)	87.5 (-5.3)	74.2 (-6.1)
π_0 + Ours	**96.9** (+1.0)	**97.7** (+0.6)	**94.9** (+2.1)	**82.1** (+1.8)
$\pi_{0.5}$ baseline	98.7	98.1	96.5	92.9
$\pi_{0.5}$ + AAC ($N = 20$)	98.1 (-0.6)	98.9 (+0.8)	97.7 (+1.2)	94.5 (+1.6)
$\pi_{0.5}$ + AutoHorizon	98.8 (+0.1)	99.2 (+1.1)	97.2 (+0.7)	91.6 (-1.3)
$\pi_{0.5}$ + Ours	**99.3** (+0.6)	**99.3** (+1.2)	**98.0** (+1.5)	**95.1** (+2.2)
GR00T N1.5 baseline	92.4	98.0	95.9	91.9
GR00T N1.5 + AAC	93.5 (+1.1)	97.5 (-0.5)	94.0 (-1.9)	92.0 (+0.1)
GR00T N1.5 + AutoHorizon	92.5 (+0.1)	98.4 (+0.4)	96.5 (+0.6)	89.5 (-2.4)
GR00T N1.5 + Ours	**93.6** (+1.2)	**98.7** (+0.7)	**96.8** (+0.9)	**93.1** (+1.2)

Table 3: **Real-world success rates on SO-101.** ID/OOD denote training/held-out initial object positions. Averages pool all trials within each task. **Bold** marks the best result per column.

	Success Rate (%) ↑						
	Pick-and-Place				Target Placement		
Method	Banana	Box	Beans	Average	ID	OOD	Average
$\pi_{0.5}$ ($h = 10$)	35.00	25.00	35.00	31.67	33.33	25.00	30.56
$\pi_{0.5}$ ($h = 50$)	40.00	55.00	35.00	43.33	62.50	25.00	50.00
KPE-AC (ours)	**55.00**	**70.00**	**40.00**	**55.00**	**70.83**	**50.00**	**63.89**

their baselines execute 5, 5, and 16 actions per query. The gains thus extend from long action chunks to much shorter predictions. These results support using the same generation-based execution rule across policies with different architectures and chunk lengths, without additional policy training.

5.3 Real-World Experiments

Tasks and Fine-Tuning. We evaluate Pick-and-Place (T1) and Target Placement (T2) on an SO-101 robot. T1 requires placing the instructed object into a receptacle; T2 requires placement within a marked region at a specified orientation. Success requires the target condition to remain satisfied after release. Each task uses a separate $\pi_{0.5}$ policy fine-tuned on 48 demonstrations, with 12 additional demonstrations held out for validation. KPE-AC and the $h = 10$ and $h = 50$ baselines share this checkpoint and evaluation conditions. Each method receives 20 trials per T1 object (banana, box, and beans), and 24 in-distribution (ID) and 12 out-of-distribution (OOD) trials for T2, with OOD locations excluded from training. Task averages pool all trials: T1 weights the three objects equally, while T2 weights the ID rate twice as heavily as the OOD rate. Training, evaluation details, and example rollouts are in Appendix F.

Adaptive Execution on the Robot. Table 3 shows that increasing the fixed horizon from 10 to 50 improves average success on both tasks, so more frequent observation alone does not ensure better manipulation. In T1, the longer horizon improves banana and box placement but leaves beans unchanged. KPE-AC improves all three objects over both baselines, with larger gains on banana and box than on beans. Its aggregate improvement therefore spans different objects, although the benefit of execution adaptation varies among them. T2 further distinguishes performance at training and held-out locations: extending the fixed horizon improves ID success but leaves OOD success unchanged. KPE-AC improves both, with a larger gain over $h = 50$ on OOD trials, narrowing the gap between the two location sets. Since all methods use the same task-specific policy, these results suggest that adapting execution helps the learned policy handle the evaluated spatial variations without additional training on the held-out locations.

Table 4: **Ablation of KPE and CCC on RoboCasa.** ✓/× denote enabled/disabled components. Δ is the overall gain in percentage points over $\pi_{0.5}$ with $h = 5$. **Bold** marks the best result per column.

Components		Success Rate (%)					
KPE	CCC	Relocation	Container	Rotation	Button	Overall	Gain
×	×	54.57	47.14	38.00	22.29	40.50	–
✓	×	55.14	59.71	**43.14**	27.71	46.43	+5.93
×	✓	55.71	59.43	41.71	26.86	45.93	+5.43
✓	✓	**64.86**	**60.57**	42.29	**29.71**	**49.36**	**+8.86**

5.4 Ablation Studies and Analysis

We use the same $\pi_{0.5}$ checkpoint and 1,400-episode RoboCasa protocol as in Section 5.1. KPE only executes the corrected late-KPE candidate without CCC adjustment. CCC only applies the CCC rule with $h_t^{\mathrm{KPE}} = H = 50$ and $h_{\min} = 5$, executing five actions whenever no overlap is available (Appendix G). The full method combines both signals.

Contributions of KPE and Cross-Chunk Consistency. Both single-signal variants improve all four task groups over the $h = 5$ baseline and achieve similar overall success (Table 4). Their combination, however, has different effects across groups. Either signal alone accounts for most of the Container gain, while the large Relocation gain appears mainly when they are combined. Thus, the similar aggregate scores of KPE only and CCC only do not make the signals interchangeable. CCC does not measure uncertainty within a generation, while KPE does not compare plans across observations; combining them provides both sources of information. The higher overall success of the full method supports using both views: KPE selects a prefix from the current generation, and CCC determines how much of that prefix to retain based on agreement across observations.

Energy Definition and Denoising Window. The additional ablations examine which part of the generation path is useful for execution (Appendix G). The raw-KPE comparison holds the denoising window and CCC adjustment fixed, changing only whether the full-path mean velocity is subtracted. Corrected KPE improves overall success over raw KPE through gains on Relocation and Container, while Late-5 improves over Full-10 in three of the four groups. Late-5 and Full-10 both subtract the full-path mean velocity and use the same execution bounds and CCC adjustment. Both windows retain all ten denoising updates, so the difference concerns the signal used for horizon selection rather than the number of generation steps. Full-10 includes the same late corrections, so its lower overall success indicates that adding early energy need not improve horizon selection. These results support separating trajectory corrections from overall transport and focusing on late corrections when choosing how far to execute.

6 Conclusion

We introduced KPE-AC, which brings Kinetic Path Energy to action generation for adaptive execution in flow-based VLA models. Our analysis connects correction energy along a single denoising trajectory to action uncertainty, providing a basis for selecting how much of a predicted chunk to execute. KPE-AC combines late-correction energy with cross-chunk consistency to adapt the execution horizon without additional action samples or policy training. Experiments on RoboCasa, LIBERO, and real-world manipulation demonstrate improved task success across multiple VLA backbones. These results highlight the value of the internal generation process for deciding when to observe and replan. Our method currently requires access to intermediate velocities from flow-based action generators. Extending this approach to other policy architectures remains a direction for future work.

AI use statement

We used generative AI tools to assist with literature search, manuscript drafting and editing, and result analysis and visualization. The authors take responsibility for the final content and claims.

REFERENCES

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

Kevin Black, Noah Brown, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization, 2025a. URL https://arxiv.org/abs/2504.16054.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, 2025b. URL https://arxiv.org/abs/2506.07339.

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, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. UniVLA: Learning to act anywhere with task-centric latent actions, 2025. URL https://arxiv.org/abs/2505.06111.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, Deli Zhao, and Hao Chen. WorldVLA: Towards autoregressive action world model, 2025. URL https://arxiv.org/abs/2506.21539.

Feng Chen, Xianghui Wang, Yuxuan Chen, Boying Li, Yefei He, Zeyu Zhang, and Yicheng Wu. Dynamic execution commitment of vision-language-action models, 2026. URL https://arxiv.org/abs/2605.11567.

Xiaoyu Chen, Hangxing Wei, Pushi Zhang, Chuheng Zhang, Kaixin Wang, Yanjiang Guo, Rushuai Yang, Yucen Wang, Xinquan Xiao, Li Zhao, Jianyu Chen, and Jiang Bian. villa-X: Enhancing latent action modeling in vision-language-action models, 2025. URL https://arxiv.org/abs/2507.23682.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion, 2023. URL https://arxiv.org/abs/2303.04137.

Chris Finlay, Joern-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ODE: the world of Jacobian and kinetic regularization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3154–3164. PMLR, 2020. URL https://proceedings.mlr.press/v119/finlay20a.html.

Yihang Fu, Zhenyu Zhang, Yifan Zhang, Zeyu Wang, Zhen Huang, and Yu Luo. MergeVLA: Cross-skill model merging toward a generalist vision-language-action agent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://arxiv.org/abs/2511.18810.

Herbert Goldstein, Charles P. Poole, and John L. Safko. *Classical Mechanics*. Addison-Wesley, 3 edition, 2002. ISBN 9780201657029. URL https://www.pearson.com/en-us/subject-catalog/p/classical-mechanics/P200000006871/9780201657029.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, 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, 2024. URL https://arxiv.org/abs/2406.09246.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

Ziyun Li and Henrik Boström. FlowGuard: Guarding flow matching via conformal sampling. In *Proceedings of the Fourteenth Symposium on Conformal and Probabilistic Prediction with Applications*, volume 266 of *Proceedings of Machine Learning Research*, pp. 775–777. PMLR, 2025. URL https://proceedings.mlr.press/v266/li25a.html.

Ziyun Li, Ben Dai, Huancheng Hu, Henrik Boström, and Soon Hoe Lim. EnfoPath: Energy-informed analysis of generative trajectories in flow matching, 2025. URL https://arxiv.org/abs/2511.19087.

Ziyun Li, Huancheng Hu, Soon Hoe Lim, Xuyu Li, Fei Gao, Enmao Diao, Zezhen Ding, Michalis Vazirgiannis, and Henrik Bostrom. A kinetic-energy perspective of flow matching, 2026. URL https://arxiv.org/abs/2602.07928.

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, 2026. URL https://arxiv.org/abs/2604.04161.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2022. URL https://arxiv.org/abs/2210.02747.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL https://arxiv.org/abs/2306.03310.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022. URL https://arxiv.org/abs/2209.03003.

Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via closed-loop resampling, 2024. URL https://arxiv.org/abs/2408.17355.

Lei Lv, Yunfei Li, Yu Luo, Fuchun Sun, and Xiao Ma. FLAC: Maximum entropy RL via kinetic energy regularized bridge matching, 2026. URL https://arxiv.org/abs/2602.12829.

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, 2024. URL https://arxiv.org/abs/2406.02523.

Kirill Neklyudov, Rob Brekelmans, Daniel Severo, and Alireza Makhzani. Action matching: Learning stochastic dynamics from samples. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 25858–25889. PMLR, 2023. URL https://proceedings.mlr.press/v202/neklyudov23a.html.

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, 2025. URL https://arxiv.org/abs/2503.14734.

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, 2024. URL https://arxiv.org/abs/2405.12213.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient action tokenization for vision-language-action models, 2025. URL https://arxiv.org/abs/2501.09747.

Physical Intelligence. OpenPI: LIBERO evaluation. Official code repository, 2025. URL https://github.com/Physical-Intelligence/openpi/tree/35106c9bea0f84584e1a9b3a9ee1ae7aba070532/examples/libero.

Neta Shaul, Ricky T. Q. Chen, Maximilian Nickel, Matthew Le, and Yaron Lipman. On kinetic optimal probability paths for generative models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 30883–30907. PMLR, 2023. URL https://proceedings.mlr.press/v202/shaul23a.html.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. Smolvla: A vision-language-action model for affordable and efficient robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Junhyuk So, Chiwoong Lee, Shinyoung Lee, Jungseul Ok, and Eunhyeok Park. Improving generative behavior cloning via self-guidance and adaptive chunking, 2025. URL https://arxiv.org/abs/2510.12392.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=CD9Snc73AW.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. VLA knows its limits, 2026a. URL https://arxiv.org/abs/2602.21445.

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*, 2026b. URL https://arxiv.org/abs/2601.20130.

Shuai Yang, Hao Li, Bin Wang, Yilun Chen, Yang Tian, Tai Wang, Hanqing Wang, Feng Zhao, Yiyi Liao, and Jiangmiao Pang. InstructVLA: Vision-language-action instruction tuning from understanding to manipulation, 2025. URL https://arxiv.org/abs/2507.17520v2.

Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Sejune Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, Lars Liden, Kimin Lee, Jianfeng Gao, Luke Zettlemoyer, Dieter Fox, and Minjoon Seo. Latent action pretraining from videos, 2024. URL https://arxiv.org/abs/2410.11758.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, Ankur Handa, Ming-Yu Liu, Donglai Xiang, Gordon Wetzstein, and Tsung-Yi Lin. CoT-VLA: Visual chain-of-thought reasoning for vision-language-action models, 2025. URL https://arxiv.org/abs/2503.22020.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL https://arxiv.org/abs/2304.13705.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=kt51kZH4aG.

A Detailed Comparison of Raw and Correction KPE

We compare raw and correction KPE for ten RoboCasa episodes to show the individual examples behind the average in Figure 2. For each example, both measures are computed from the same action chunk and denoising trajectory. The only difference is whether the mean velocity over the full trajectory is subtracted. Figures 5–9 each show two examples, with correction KPE on the left and raw KPE on the right for each task.

Energy Computation. For action i and a denoising interval starting at τ, we compute

$$e_i^{\text{raw}}(\tau) = \tfrac{1}{2}\|\mathbf{v}_i^\tau\|_2^2 \Delta\tau, \qquad e_i^{\text{corr}}(\tau) = \tfrac{1}{2}\|\mathbf{v}_i^\tau - \bar{\mathbf{v}}_i\|_2^2 \Delta\tau, \tag{10}$$

where $\bar{\mathbf{v}}_i$ is the mean velocity over the full trajectory and $\Delta\tau = 0.1$. Each column in a heatmap corresponds to a future action, and each row to a denoising interval. The vertical axis uses solver time $s = 1 - \tau$, decreasing from 1 (noise) at the top to 0 (actions) at the bottom. The red line at $s = 0.5$ separates the early and late halves. All twenty heatmaps use the same logarithmic color scale to show energy values. Figure 2 instead shows normalized energy shares.

Energy During Denoising. For a given action, raw KPE often changes slightly across denoising steps, producing vertical bands of similar color. Correction KPE shows clearer differences across both actions and denoising steps. In the lower halves of the correction plots, larger values indicate deviations from the mean velocity that remain late in generation. For example, `TurnOnStove` requires careful gripper positioning to turn a stove knob. In Figure 9 (right), correction KPE shows clear difference between future actions during late denoising, while raw KPE remains relatively uniform across denoising steps. These patterns are less apparent in raw KPE, supporting the use of correction KPE to examine how individual actions are generated. Table 6 compares task success when using the two energy measures for horizon selection.

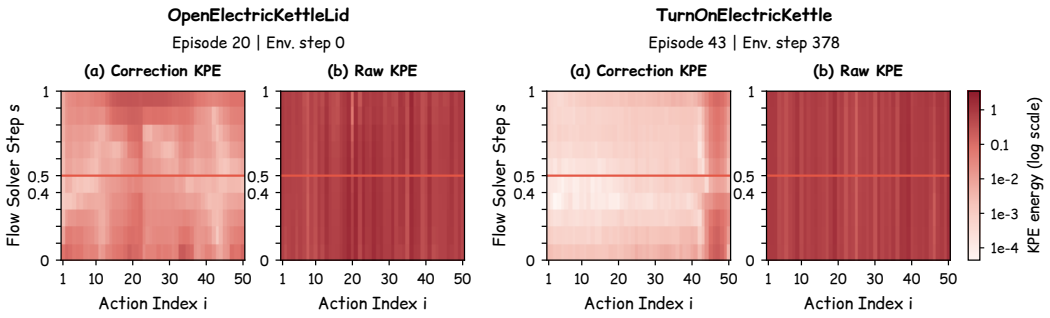

Figure 5: **Raw and correction KPE.** `OpenElectricKettleLid` (left pair) and `TurnOnElectricKettle` (right pair). Each pair compares (a) correction KPE and (b) raw KPE for the same action chunk.

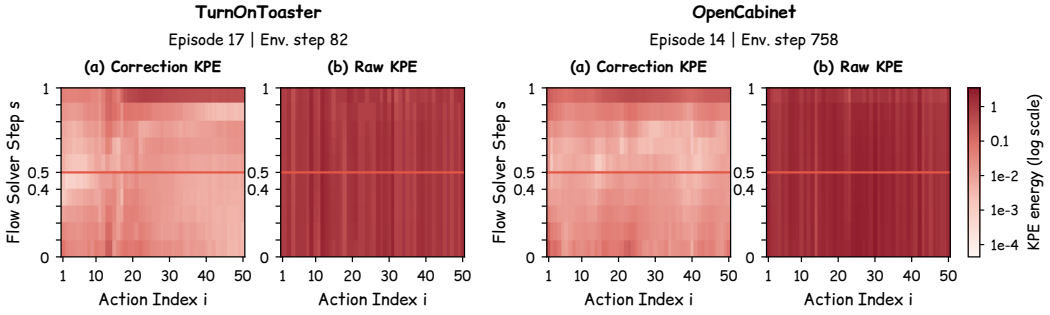

Figure 6: **Raw and correction KPE.** `TurnOnToaster` (left pair) and `OpenCabinet` (right pair). Each pair compares (a) correction KPE and (b) raw KPE for the same action chunk.

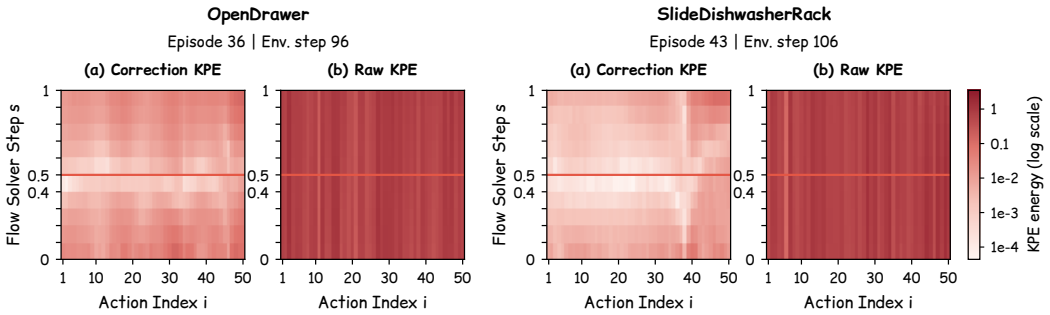

Figure 7: **Raw and correction KPE.** `OpenDrawer` (left pair) and `SlideDishwasherRack` (right pair). Each pair compares (a) correction KPE and (b) raw KPE for the same action chunk.

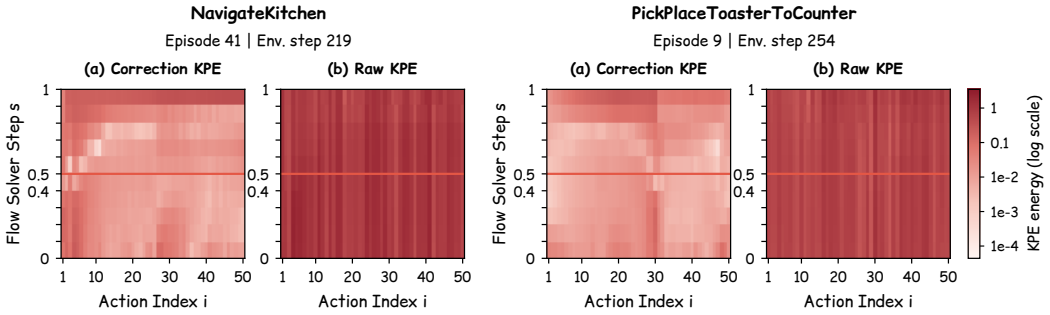

Figure 8: **Raw and correction KPE.** `NavigateKitchen` (left pair) and `PickPlaceToasterToCounter` (right pair). Each pair compares (a) correction KPE and (b) raw KPE for the same action chunk.

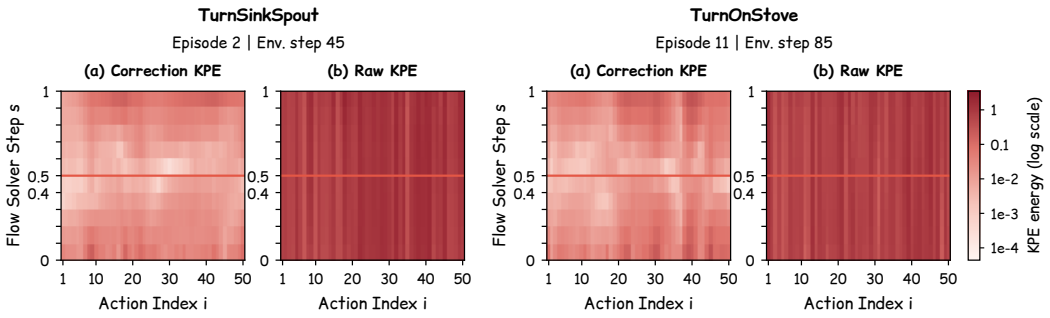

Figure 9: **Raw and correction KPE.** `TurnSinkSpout` (left pair) and `TurnOnStove` (right pair). Each pair compares (a) correction KPE and (b) raw KPE for the same action chunk.

B Correction Energy and Conditional Action Uncertainty

Section 4.1 uses changes within one generation as a signal of variation across possible actions. We make this connection explicit in a conditional Gaussian setting, where the mean action and its uncertainty can be varied separately. The derivation shows why subtracting the mean velocity removes the former while retaining a dependence on the latter. We then examine the corresponding energy–uncertainty relationship in a trained VLA.

Assumptions. Fix the observation o. Assume a scalar action $A \mid o \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu \in \mathbb{R}$ and $\sigma > 0$, and draw $Z \sim \mathcal{N}(0,1)$ independently of A given o. For the training interpolation $\widetilde{X}_\tau = (1-\tau)Z + \tau A$, use the population-optimal field of the squared-error flow-matching objective,

$$v^\star(x,\tau) = \mathbb{E}[A - Z \mid \widetilde{X}_\tau = x, o]. \tag{11}$$

Let x_τ be the exact solution of $\dot{x}_\tau = v^\star(x_\tau, \tau)$ initialized at a fresh noise sample $x_0 = \xi \sim \mathcal{N}(0,1)$. Along this inference trajectory, write $v_\tau = v^\star(x_\tau, \tau)$ and $\bar{v} = \int_0^1 v_\tau \, \mathrm{d}\tau$. The full-trajectory

correction energy is

$$E^{\mathrm{corr}}(\xi) = \frac{1}{2}\int_0^1 (v_\tau - \bar{v})^2 \,\mathrm{d}\tau. \tag{12}$$

All expectations below are conditional on the fixed observation; we suppress o where unambiguous.

Theorem B.1 (Correction energy and action uncertainty)**.** *Under the assumptions above, the expected correction energy is independent of the mean action μ and proportional to its conditional standard deviation σ:*

$$\mathbb{E}_\xi\left[E^{\mathrm{corr}}(\xi) \mid o\right] = \left(1 - \frac{\pi}{4}\right)\sigma. \tag{13}$$

More precisely, each initial value $\xi \in \mathbb{R}$ gives $E^{\mathrm{corr}}(\xi) = (1-\pi/4)\sigma\xi^2$.

The proof separates the generation path into its mean and scale. Lemma B.2 gives this path under the optimal velocity field. Lemma B.3 then measures how its scale changes relative to a constant-velocity reference, yielding the correction energy.

Lemma B.2 (Gaussian generation trajectory)**.** *Let $r(\tau) = \sqrt{(1-\tau)^2 + \sigma^2\tau^2}$. Under the assumptions above, the optimal velocity field is*

$$v^\star(x,\tau) = \mu + \frac{\tau\sigma^2 - (1-\tau)}{r(\tau)^2}(x - \tau\mu), \tag{14}$$

and its inference trajectory satisfies

$$x_\tau = \tau\mu + r(\tau)\xi, \qquad 0 \le \tau \le 1. \tag{15}$$

Proof. The variables $A - Z$ and $\widetilde{X}_\tau$ are jointly Gaussian, with

$$\begin{aligned} \mathbb{E}[A - Z] &= \mu, & \mathbb{E}[\widetilde{X}_\tau] &= \tau\mu, \\ \mathrm{Var}(\widetilde{X}_\tau) &= r(\tau)^2, & \mathrm{Cov}(A - Z, \widetilde{X}_\tau) &= \tau\sigma^2 - (1-\tau). \end{aligned}$$

The Gaussian conditional-mean formula therefore gives Eq. (14). Since $\sigma > 0$, $r(\tau)$ is positive throughout $[0,1]$, and

$$\frac{r'(\tau)}{r(\tau)} = \frac{\tau\sigma^2 - (1-\tau)}{r(\tau)^2}.$$

Setting $y_\tau = x_\tau - \tau\mu$ reduces the inference ODE to $\dot{y}_\tau = [r'(\tau)/r(\tau)]y_\tau$. Thus $y_\tau/r(\tau)$ is constant. The initial conditions $r(0) = 1$ and $y_0 = \xi$ give $y_\tau = r(\tau)\xi$, proving Eq. (15). □

In particular, $x_1 = \mu + \sigma\xi$ has the required action distribution. The energy is evaluated along this inference trajectory, which differs from the straight interpolation of an individual training pair.

Lemma B.3 (Integrated variation of the Gaussian scale)**.** *For $r(\tau) = \sqrt{(1-\tau)^2 + \sigma^2\tau^2}$ with $\sigma > 0$,*

$$\int_0^1 \left[r'(\tau) - (\sigma - 1)\right]^2 \,\mathrm{d}\tau = \left(2 - \frac{\pi}{2}\right)\sigma. \tag{16}$$

Proof. The endpoints $r(0) = 1$ and $r(1) = \sigma$ imply $\int_0^1 r'(\tau)\,\mathrm{d}\tau = \sigma - 1$. Expanding the square therefore gives

$$\int_0^1 \left[r'(\tau) - (\sigma-1)\right]^2 \,\mathrm{d}\tau = \int_0^1 r'(\tau)^2\,\mathrm{d}\tau - (\sigma-1)^2.$$

Write $q(\tau) = r(\tau)^2$. Direct differentiation and simplification yield

$$r'(\tau)^2 = \frac{\left[(1+\sigma^2)\tau - 1\right]^2}{q(\tau)} = 1 + \sigma^2 - \frac{\sigma^2}{q(\tau)}.$$

With $u = [(1+\sigma^2)\tau - 1]/\sigma$, the remaining integral is

$$\int_0^1 \frac{\mathrm{d}\tau}{q(\tau)} = \frac{1}{\sigma}\int_{-1/\sigma}^{\sigma} \frac{\mathrm{d}u}{1+u^2} = \frac{\arctan\sigma + \arctan(1/\sigma)}{\sigma} = \frac{\pi}{2\sigma}.$$

Hence $\int_0^1 r'(\tau)^2\,\mathrm{d}\tau = 1 + \sigma^2 - \pi\sigma/2$. Subtracting $(\sigma-1)^2$ gives Eq. (16). □

Proof of Theorem B.1. By Lemma B.2, the velocity along the inference path and its mean are

$$v_\tau = \mu + r'(\tau)\xi, \qquad \bar{v} = x_1 - x_0 = \mu + (\sigma-1)\xi.$$

Their difference is $v_\tau - \bar{v} = [r'(\tau) - (\sigma-1)]\xi$, so the mean action μ cancels from every trajectory. Substituting into the energy definition and applying Lemma B.3 gives

$$\begin{aligned} E^{\mathrm{corr}}(\xi) &= \frac{\xi^2}{2}\int_0^1 \left[r'(\tau) - (\sigma-1)\right]^2\,\mathrm{d}\tau \\ &= \left(1 - \frac{\pi}{4}\right)\sigma\xi^2. \end{aligned}$$

Taking the expectation over $\xi \sim \mathcal{N}(0,1)$ and using $\mathbb{E}[\xi^2] = 1$ proves Eq. (13). □

Relation to Action Uncertainty. The cancellation of μ is central to the result: shifting the mean action leaves the expected correction energy unchanged, while increasing its conditional action standard deviation increases that energy linearly. Individual generations still depend on the initial noise through ξ^2, so the proportionality concerns their expectation. Under the stated Gaussian and optimal-field assumptions, this establishes a link between full-path correction energy and action uncertainty. The next analysis asks whether the late-correction signal used by our method retains a positive association with uncertainty in a trained policy.

B.1 Action Uncertainty in a Trained VLA

Sampling. We analyze 237 observations from 60 reference episodes across 20 RoboCasa tasks, with five tasks per task group and three evaluation seeds per task. Holding the images, robot state, and instruction fixed, we vary only the initial noise. For each of the 50 action positions, we estimate uncertainty from the sample standard deviation of each normalized action coordinate over 32 generations, averaged across the 12 active coordinates. We measure correction KPE from one pre-selected, separate generation, summing energy over the final five of ten denoising intervals after subtracting the full-path mean velocity. Thus, the energy sample is independent of the samples used to estimate uncertainty. Repeated generation is used only for this analysis; KPE-AC uses energy from a single generation to select the execution horizon.

Energy and Action Variability. Figure 10 tests whether this single-generation signal identifies actions with greater variability across independent predictions. In (a), we sort the 50 action positions by energy into ten equally sized groups and divide their uncertainty by its observation mean. A value above one therefore indicates greater uncertainty than the observation's average. In (b), we define high-uncertainty actions as the ten positions with the largest reference uncertainty and report their frequency within each energy group. Their overall prevalence is 20%. The annotations compare uncertainty in the highest and lowest energy deciles and report the high-uncertainty fraction among the top 20% of energy. These quantiles are used to analyze the signal, separately from the horizon-selection threshold in Section 4.2.

Detecting High-Uncertainty Actions. Figure 10(c) and (d) sweep the energy threshold to assess how well KPE ranks high-uncertainty positions above the remaining positions. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR); its diagonal represents random ranking. The precision–recall curve measures the fraction of selected positions that have high uncertainty and the fraction of all high-uncertainty positions recovered. AP denotes average precision, computed as the non-interpolated area under this curve. The horizontal references in (b) and (d) mark the 20% positive prevalence.

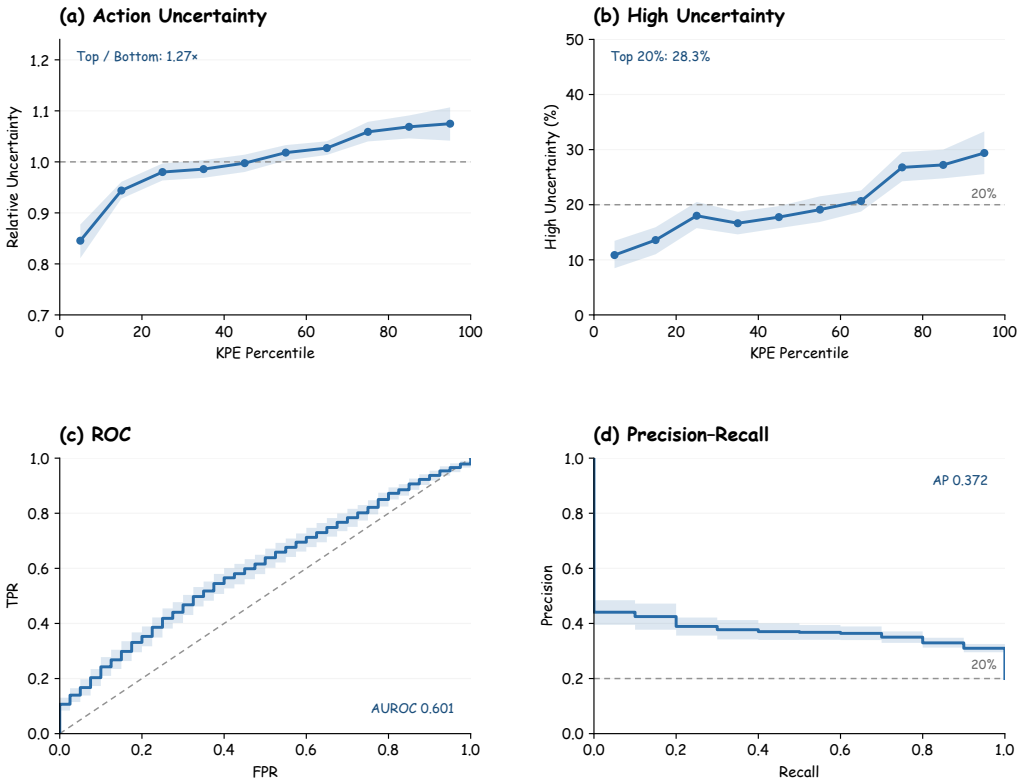

Figure 10: **Single-generation KPE and action uncertainty.** (a) Relative uncertainty across energy deciles. (b) Frequency of high-uncertainty actions within each decile. (c,d) ROC and precision–recall curves for identifying these actions. Uncertainty is estimated from 32 independent generations. Shading denotes pointwise 95% task-bootstrap confidence intervals.

Analysis. We compute each curve and metric within an observation, then average across observations within episodes, episodes within tasks, and tasks within each task group, assigning equal weight to each group. Shading shows pointwise 95% confidence intervals from 10,000 bootstrap resamples of whole tasks within groups. Action positions from the same observation are kept together rather than treated as independent trials. Higher-energy positions show greater action variability and contain high-uncertainty actions more often. Selecting the top 20% of energy increases their fraction from 20.0% to 28.3%. The AUROC is 0.601, with a 95% confidence interval of $[0.575, 0.626]$, above the random-ranking reference of 0.5. Together, these results show that the relationship extends beyond a rank correlation: energy from a single generation helps identify where independent action predictions vary more. This supports using late-correction KPE to compare uncertainty within a chunk. The analysis measures conditional action variability; the effect on task success is evaluated in our manipulation experiments.

C Inference Procedure

Algorithm 1 combines the two views of execution developed in Sections 4.2 and 4.3. At each query, KPE examines the denoising path of one newly generated chunk and proposes a candidate prefix. CCC then compares this plan with the unexecuted part of the previous chunk, aligning actions that refer to the same future time steps. Greater disagreement shortens the candidate and brings the next observation forward. The predicted actions themselves are unchanged. KPE-AC retains only the previous chunk and its executed length between queries. At the first query of an episode, or when no unexecuted overlap remains, it sets $P_t = 0$ and uses the KPE candidate directly. Clearing this state at each episode boundary keeps the comparison within a single rollout. Thus, both execution signals are obtained from the current generation and the stored plan, without additional policy calls.

Algorithm 1 KPE-guided action chunking

Require: Current policy inputs; execution bounds $h_{\min} < H$
Require: Previous chunk $\mathbf{A}_{t-1}$ and executed length h_{t-1}, if available
Ensure: Executed action prefix and updated state for the next query
Generate one chunk $\mathbf{A}_t$ using Eq. (2); retain solver velocities
Compute late-correction energies using Eq. (6)
Set threshold γ_t using Eq. (7)
Select candidate h_t^{KPE} using Eq. (8)
Initialize $P_t \leftarrow 0$
if a previous chunk has unexecuted actions **then**
    Set overlap length $L_t \leftarrow H - h_{t-1}$
    Align $\mathbf{A}_{t-1}[h_{t-1} : H]$ with $\mathbf{A}_t[0 : L_t]$
    Compute the CCC discrepancy score P_t using Eq. (9)
end if
Set $h_t \leftarrow \mathrm{round}\big[h_{\min} + (1 - P_t)(h_t^{\mathrm{KPE}} - h_{\min})\big]$
Execute the first h_t actions of $\mathbf{A}_t$
Store $\mathbf{A}_t$ and h_t for the next query

D Simulation Experiments on RoboCasa

The RoboCasa experiments examine how execution decisions affect a fixed policy. This appendix gives the evaluation and compute conventions, then compares the aggregate results to horizon distributions and individual rollouts. These views complement one another: the benchmark measures task success, the distributions summarize the selected execution lengths, and the rollouts show how those choices change during manipulation.

D.1 Evaluation Protocol

Our main evaluation uses $\pi_{0.5}$ as the base policy on 28 RoboCasa tasks (Nasiriany et al., 2024; Black et al., 2025a), divided into four groups of seven: Relocation, Container, Rotation, and Button. Each method is evaluated for 50 episodes per task, giving 1,400 episodes. We compare KPE-AC with the $\pi_{0.5}$ baselines at execution horizons $h \in \{5, 10, 30, 50\}$, AAC with 20 sampled chunks per query (Liang et al., 2026), and AutoHorizon (Wang et al., 2026a). Each fixed-horizon baseline executes the first h actions of every generated chunk before querying the policy again.

AutoHorizon. We adapt the released AutoHorizon method to our JAX-based RoboCasa policy, using the same checkpoint and ten-step generation process as the other methods. We extract action-token self-attention at the third denoising step, average over layers and heads, and apply the released bidirectional selector with entropy quantile 0.9, hold threshold 0.3, and execution bounds $[1, 50]$. Each query generates one chunk. The success rates in Table 1 and both graphs of Figure 11 use this same evaluation.

Metrics. We report task success, generated action chunks, and estimated policy FLOPs per 100 environment steps. Chunk counts include all 20 candidates generated by each AAC query, whereas AutoHorizon and KPE-AC generate one. We normalize aggregate costs by the total number of executed steps across both successful and failed episodes.

D.2 Inference Cost Estimation

Inference Cost. Adaptive execution affects inference demand through both the frequency of policy queries and the work performed at each query. We report generated chunks to measure sampling demand and estimated policy FLOPs to include the different costs of image processing, action generation, and horizon selection. Both metrics are normalized by executed environment steps, allowing methods with different episode lengths to be compared on the same scale.

The FLOP estimate includes image encoding, vision–language prefix processing, all denoising steps, and method-specific horizon selection. We count matrix products, convolutions, and elementwise arithmetic, including normalization and softmax, from the traced JAX graph and selector implemen-

tation. A multiply–add counts as two FLOPs; elementary functions count as one logical operation, except reciprocal square roots, which count as two. Dense padding is included. Common pre/post-processing, random-number generation, integer operations, comparisons, logging, communication, and simulator I/O are excluded. This defines an arithmetic-cost comparison independent of wall-clock execution and communication overhead.

Computational Cost per Policy Query. The RoboCasa policy processes three 224×224 images and a state-conditioned prompt padded to 200 tokens, giving a 968-token prefix. It generates a 50×32 action chunk with ten denoising steps. All 32 model action dimensions are counted, including the dimensions not executed by the robot. Let F_{pre} denote image encoding and prefix processing, F_{den} the cost of generating one candidate through all denoising steps, and $C_{m,e,q}$ the additional arithmetic for method m at query q of episode e. Then

$$F_{m,e,q} = F_{\mathrm{pre}} + N_m F_{\mathrm{den}} + C_{m,e,q}, \tag{17}$$

where $N_m = 20$ for AAC and $N_m = 1$ for the other evaluated methods. The decomposition separates visual-language processing, which occurs once per query, from action generation, which scales with the number of candidates. AAC shares the prefix pass across its batch of candidates, so generating 20 chunks does not multiply the entire policy cost by 20. The resulting backbone costs are 5.0310 TFLOPs for one candidate and 12.4372 TFLOPs for AAC, before adding selector arithmetic.

The term $C_{m,e,q}$ accounts for correction-energy and cross-chunk calculations in KPE-AC, attention extraction and horizon selection in AutoHorizon, and entropy and cumulative-motion calculations in AAC. For AutoHorizon, this includes the additional query–key product, full-key softmax, layer/head reduction, and bidirectional selector used by our implementation. Recorded overlap lengths determine KPE-AC's query-dependent costs. For AAC and AutoHorizon selector arithmetic, lower and upper bounds yield the same reported TFLOPs to two decimal places.

Summation. Let $Q_{m,e}$ and $S_{m,e}$ denote the number of policy queries and executed environment steps in episode e. We normalize total cost by total executed steps:

$$\begin{aligned} \mathrm{TFLOPs}/100 &= \frac{100 \sum_e \sum_{q=1}^{Q_{m,e}} F_{m,e,q}}{10^{12} \sum_e S_{m,e}}, \\ \mathrm{Chunks}/100 &= \frac{100 N_m \sum_e Q_{m,e}}{\sum_e S_{m,e}}. \end{aligned} \tag{18}$$

Both metrics pool all 1,400 RoboCasa episodes, including failures, and count every generated candidate. Dividing the total cost by the total executed steps gives equal weight to each environment step. A final chunk is counted even when the episode ends before it is fully executed, which explains small differences between the measured chunk count for a fixed horizon h and $100/h$. We aggregate unrounded costs and round only the reported results. These quantities determine the horizontal positions and circle areas in Figure 11(a).

D.3 Selected-Horizon Distributions

Figure 11 connects task success to the amount and timing of policy inference. Figure 11(a) places each method by its success rate and estimated compute, with circle area indicating generated chunks. The separation between KPE-AC and AAC shows the benefit of using an existing generation path as an execution signal: KPE-AC achieves higher success while avoiding repeated candidate generation. AutoHorizon makes a different trade-off, using fewer policy queries at lower task success. The fixed-horizon results also show that more frequent inference does not necessarily improve success. KPE-AC achieves higher success than $h = 5$ and $h = 10$ with lower compute, supporting the value of adapting when to request new observations. Together with the comparison to AAC, these results show that useful execution guidance can be obtained from the existing generation process without additional action samples.

Figure 11(b) summarizes one selected horizon per policy query over all 1,400 episodes. We use five-action bins and normalize by each method's total query count, so each curve sums to 100%. Each AAC query contributes one selection despite generating 20 candidates; KPE-AC contributes the horizon after CCC adjustment. The allowed ranges are 2–50 for AAC, 1–50 for AutoHorizon,

and 5–50 for KPE-AC. Horizons are recorded before truncation at episode termination. Thus, the curves describe query-weighted policy decisions across the evaluated tasks.

AAC tends to select shorter horizons, while AutoHorizon concentrates its selections around 16–20 actions and the maximum horizon. These intermediate horizons allow KPE-AC to execute more of the predicted sequence while still requesting a new observation before the full chunk ends. Its higher success is therefore achieved without consistently reducing execution to the shortest possible prefix. Together with Figure 11(a), this behavior shows that KPE-AC can retain longer action sequences while keeping inference costs below those of frequent replanning. This distribution complements its task success by showing that each method uses both frequent feedback and longer execution prefixes across the benchmark. The following rollout examples reveal how these choices vary within individual tasks.

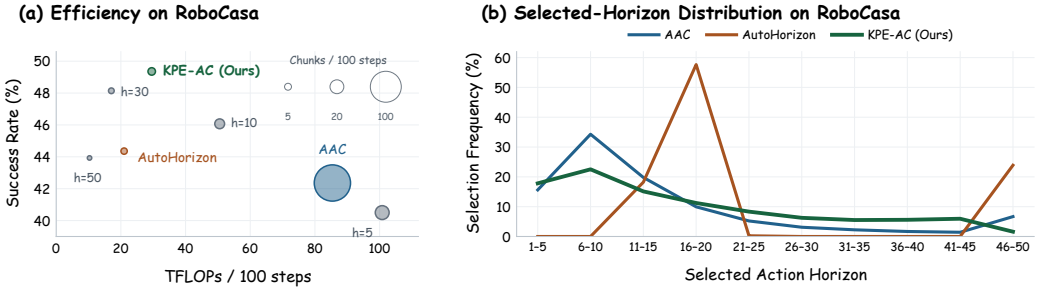

Figure 11: **RoboCasa efficiency and horizon selection.** (a) Success versus estimated TFLOPs; circle area represents generated chunks. Costs are per 100 executed steps. (b) Query-weighted horizon frequencies in five-action bins. All results correspond to the evaluations in Table 1.

D.4 Additional Qualitative Results

Figures 12–17 extend the main-text example to six selected successful rollouts covering object placement, cabinet opening, stove control, and appliance activation. Each figure aligns observations with the KPE candidate horizon (orange) and the executed horizon after CCC adjustment (blue). The gap between the curves shows how much cross-chunk disagreement shortens the candidate. Stage labels describe the observed motion and are provided only to help read the figures.

Across these examples, some parts follow the expected pattern of longer execution during sustained motion and shorter prefixes near alignment or contact. Other parts contain repeated shortening and lengthening within the same stage. KPE-AC responds to the current generation and its agreement with the previous plan, so a single stage need not have one characteristic horizon. All displayed rollouts reach their task goals despite this variation. Together, they illustrate how changing execution lengths accompany successful manipulation across different tasks, complementing the aggregate success rates in Table 1.

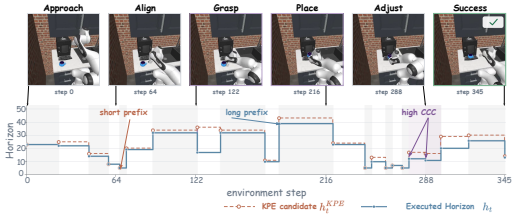

Figure 12: KPE-AC horizon adaptation during a `CoffeeSetupMug` rollout in RoboCasa.

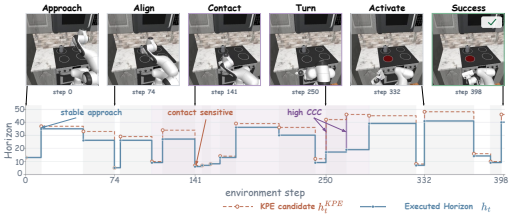

Figure 13: KPE-AC horizon adaptation during a `TurnOnStove` rollout in RoboCasa.

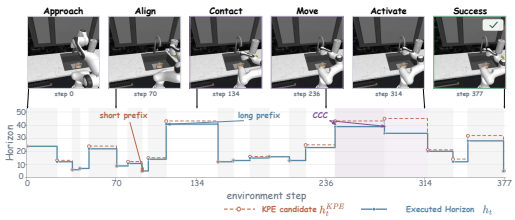

Figure 14: KPE-AC horizon adaptation during a `TurnOnSinkFaucet` rollout in RoboCasa.

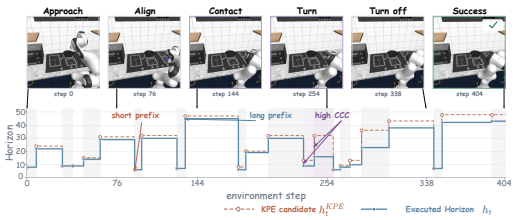

Figure 15: KPE-AC horizon adaptation during a `TurnOffStove` rollout in RoboCasa.

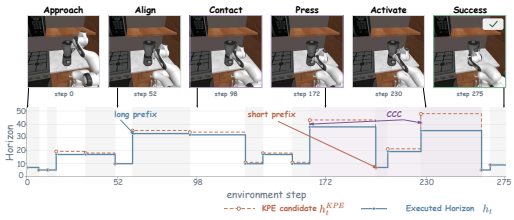

Figure 16: KPE-AC horizon adaptation during a `TurnOnBlender` rollout in RoboCasa.

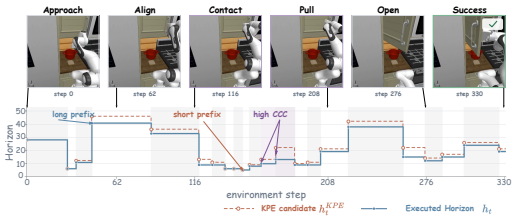

Figure 17: KPE-AC horizon adaptation during an `OpenCabinet` rollout in RoboCasa.

E Transfer Across Flow-Based VLAs

The transfer experiments test whether the execution signal remains useful across policy architectures and action chunk lengths. We describe the shared evaluation protocol for Table 2 and provide a separate comparison with published LIBERO results.

E.1 Evaluation Protocol

We evaluate the baseline, AAC (Liang et al., 2026), AutoHorizon (Wang et al., 2026a), and KPE-AC on LIBERO's Spatial, Object, Goal, and Long suites, each with ten tasks (Liu et al., 2023). Every method uses three environment seeds (7, 17, and 27), with 50 episodes per task and seed, totaling 6,000 episodes per method and backbone. The π_0 and $\pi_{0.5}$ policies generate chunks of 50 and 10 actions, respectively. GR00T N1.5 generates and, in its baseline, executes 16 actions per query. We use the released OpenPI LIBERO checkpoints for π_0 and $\pi_{0.5}$ (Black et al., 2024; 2025a; Physical Intelligence, 2025), and a publicly available GR00T N1.5 checkpoint jointly fine-tuned on all four LIBERO suites[1]. Within each backbone, all methods share the same weights and evaluation settings, isolating the effect of execution strategy. No policy weights are updated for horizon selection.

E.2 Comparison with Prior Works on LIBERO

Table 5 places our $\pi_{0.5}$ result alongside previously reported LIBERO suite scores, retaining the training and evaluation settings of each source. The table notes distinguish the original model references from the sources of the reported results. The CoT-VLA average is recomputed from its four suite scores because the original table reports an inconsistent average; this exception is marked in the table. AAC and AutoHorizon use their reported GR00T N1.5 and $\pi_{0.5}$ backbones, respectively (Liang et al., 2026; Wang et al., 2026a). These results provide broader benchmark context alongside the controlled comparisons in Table 2.

F Real-World Experiments

The real-world experiments examine adaptive execution under two manipulation requirements: placing different instructed objects into a receptacle and placing an object at a specified position and orientation. This appendix details the shared training and evaluation conditions and illustrates successful execution with paired camera views.

[1] `https://huggingface.co/ryanhoangt/gr00t-n1.5-libero-joint-4-subsets-ckpt-60k`

Table 5: **LIBERO success rates (%).** Checkpoints and evaluation protocols vary by method. **Bold** and underlined values mark the best and second-best results per column.

Method	Spatial	Object	Goal	Long	Average
Representative VLA policies					
LAPA (Ye et al., 2024)	73.8	74.6	58.8	55.4	65.7
CoT-VLA (Zhao et al., 2025)	87.5	91.6	87.6	69.0	83.9†
WorldVLA (Cen et al., 2025)	87.6	96.2	83.4	60.0	81.8
π_0-FAST (Pertsch et al., 2025)	96.4	96.8	88.6	60.2	85.5
VILLA-X (Chen et al., 2025)	97.5	97.0	91.5	74.5	90.1
GR00T N1 (NVIDIA et al., 2025)	94.4	97.6	93.0	90.6	93.9
π_0 (Black et al., 2024)	96.8	98.8	95.8	85.2	94.2
$\pi_{0.5}$ baseline (Black et al., 2025a)	98.8	98.2	**98.0**	92.4	<u>96.9</u>
UniVLA (Bu et al., 2025)	96.5	96.8	95.6	92.0	95.2
MergeVLA (Fu et al., 2026)	98.0	98.6	95.0	<u>95.0</u>	96.7
Adaptive models					
GR00T N1.5 + AAC (Liang et al., 2026)	94.4	98.6	94.2	92.8	95.0
$\pi_{0.5}$ + AutoHorizon (Wang et al., 2026a)	<u>99.1</u>	<u>99.2</u>	<u>97.5</u>	91.6	<u>96.9</u>
$\pi_{0.5}$ **+ KPE-AC (ours)**	**99.3**	**99.3**	**98.0**	**95.1**	**97.9**

LAPA results are from UniVLA (Table I; Bu et al., 2025); π_0 and π_0-FAST results are from OpenVLA-OFT (Table I; Kim et al., 2025); GR00T N1 results are from InstructVLA (Table 10; Yang et al., 2025). The $\pi_{0.5}$ results are from the official OpenPI LIBERO evaluation (Physical Intelligence, 2025), with its reported average of 96.85% rounded to 96.9%.
† CoT-VLA's suite scores average to 83.925%; its original Table 1 reports 81.13%. We report the recomputed mean.

Training and Evaluation. Each task uses a separate $\pi_{0.5}$ policy fine-tuned on 48 demonstrations, with 12 additional demonstrations held out for validation. Training uses an effective batch size of eight for 7,000 optimizer steps, and all execution methods use the 6,000-step checkpoint. Within a task, the checkpoint, observation inputs, control rate, and timeout are shared. Thus, the comparisons in Table 3 vary the execution strategy while retaining the learned manipulation policy.

For T1, each object (banana, box, and beans) is evaluated in ten layouts with varied object positions and distractors, with two trials per layout. This gives 20 trials per object and 60 per method. For T2, we evaluate three orientations ($-30°$, $0°$, and $+30°$) twice at each location. Four locations used for training form the ID set (24 trials per method), while two locations excluded from training form the OOD set (12 trials per method). Success requires the target condition to remain satisfied after release. Success rates pool all trials within each task: T1 gives equal weight to the three objects, while T2 weights the ID rate twice as heavily as the OOD rate. This aggregation preserves each trial's equal contribution to the reported task success.

Qualitative Behavior. KPE-AC is more selective about continuing a predicted action sequence than the fixed-horizon baselines. When KPE or CCC indicates a shorter execution prefix, our method requests a new observation earlier, allowing the policy to adjust its next actions. This provides opportunities to correct gripper alignment before grasping or release, while retaining longer prefixes when the execution signals support them. Figures 18 and 19 show selected successful rollouts: the robot maintains its grasp during transport and aligns the object with the destination before release, including from a held-out initial object position. With the same task-specific checkpoints, KPE-AC completes grasp-and-place sequences more reliably than both fixed-horizon baselines across all three T1 objects and both T2 location sets (Table 3). The examples illustrate this successful execution, while the table provides the comparison across all trials. Each example presents paired front and wrist views at ten corresponding times from left to right. The wrist views show gripper alignment in detail and are rotated and cropped for display; T1 uses recorded video frames and T2 uses saved camera observations.

G Additional Ablation Studies

The component analysis in Section 5.4 examines the complementary roles of KPE and CCC. Here, we specify the CCC-only method, then examine mean-velocity subtraction and the denoising win-

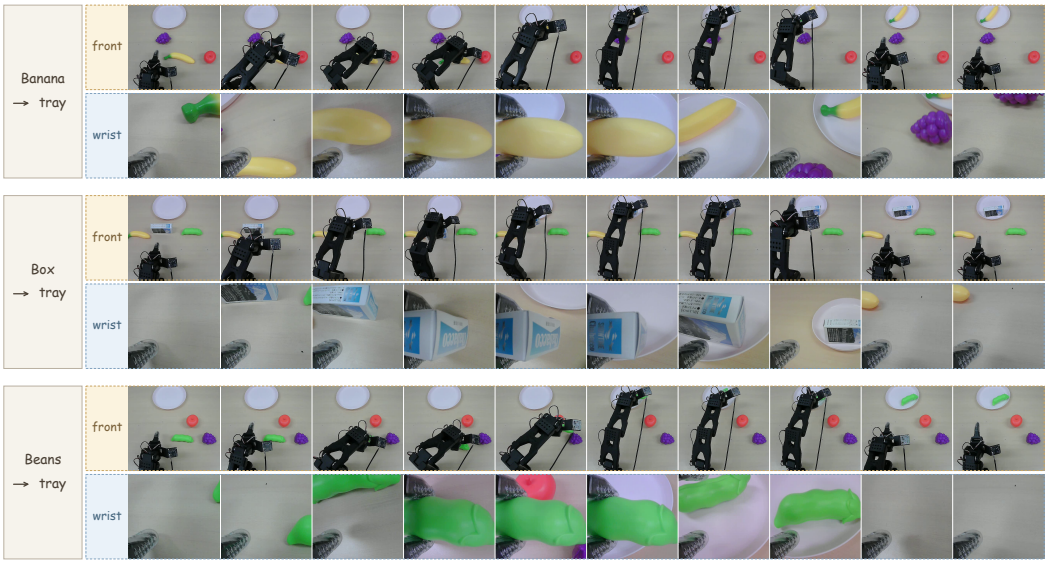

Figure 18: **Successful Pick-and-Place (T1) rollouts on SO-101.** From top to bottom, KPE-AC places the banana, box, and beans in the target tray. The paired views show the approach, grasp, transport, and release of the instructed object among distractors.

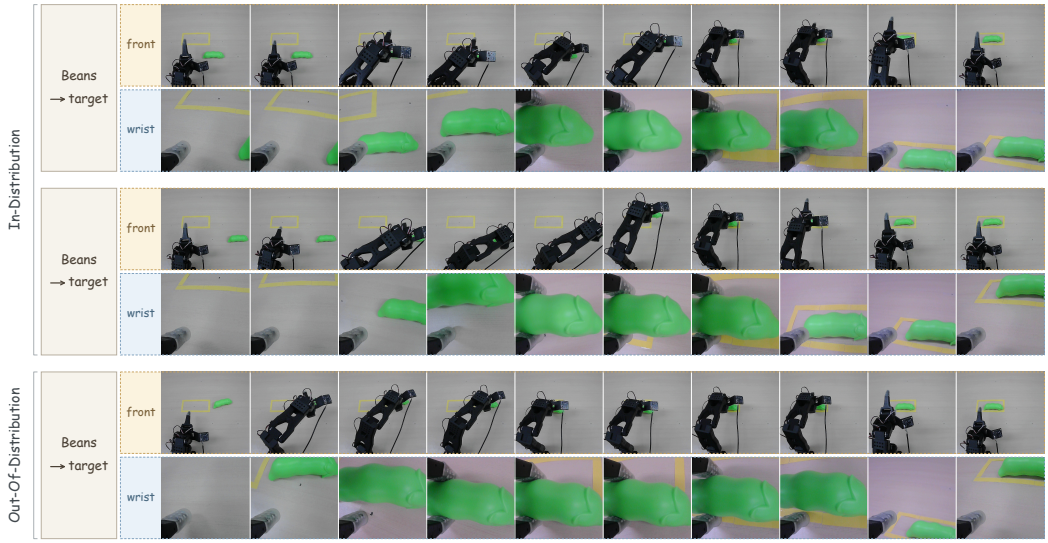

Figure 19: **Successful Target Placement (T2) rollouts on SO-101.** KPE-AC places the beans inside the yellow marked rectangle. The first two examples start at training locations (In-Distribution); the last starts at a held-out location (Out-Of-Distribution), illustrating successful execution under the spatial split defined in Section 5.3.

Table 6: **Effect of KPE correction on RoboCasa.** Raw late KPE omits mean-velocity subtraction; all other settings are identical. **Bold** marks the best result per column.

	Success Rate (%)				
KPE signal	Relocation	Container	Rotation	Button	Overall
Raw late KPE	53.71	54.57	**43.71**	**30.57**	45.64
Corrected late KPE	**64.86**	**60.57**	42.29	29.71	**49.36**

Table 7: **Effect of the denoising window on RoboCasa.** Normalized time τ runs from noise (0) to actions (1). **Bold** marks the best result per column.

		Success Rate (%)				
Window	Time range	Relocation	Container	Rotation	Button	Overall
Late-5 (Ours)	$[0.5, 1]$	**64.86**	**60.57**	42.29	**29.71**	**49.36**
Full-10	$[0, 1]$	58.57	60.00	**45.71**	26.86	47.79

dow. All comparisons use the same $\pi_{0.5}$ checkpoint and runtime as the main RoboCasa evaluation, with 50 episodes per task across 28 tasks, totaling 1,400 episodes per variant.

CCC-Only Horizon Selection. For CCC only in Table 4, the reference horizon is the full chunk length $H = 50$. When consecutive chunks overlap, we compute P_t using Equation 9 and execute $h_t = \lfloor 5 + 45(1 - P_t) + \frac{1}{2} \rfloor$ actions. This implements nearest-integer rounding with half-integers rounded up. Otherwise, including at the first query and after a fully executed chunk, we execute five actions to establish overlap for the next query. No KPE signal enters this decision.

G.1 Effect of Correcting KPE

Raw KPE includes the energy needed to transport noise to the final action, whereas correction KPE measures deviations from the mean velocity along that path. To test the effect of this distinction on execution, we compare raw and corrected late KPE while holding the denoising window, horizon-selection rule, and CCC adjustment fixed. The only change is whether the full-trajectory mean velocity is subtracted before computing energy. Table 6 shows that correction improves overall success from 45.64% to 49.36%, with gains on Relocation and Container tasks. Together with the matched-path visualizations in Appendix A, this result supports separating overall transport from trajectory corrections when selecting the execution horizon.

G.2 Effect of the Denoising Window

We compare the default Late-5 window, $\tau \in [0.5, 1]$, with Full-10, $\tau \in [0, 1]$, to examine which part of the generation path provides the execution signal (Table 7). Both variants retain all ten denoising updates, subtract the full-path mean velocity, and use the same execution bounds $[5, 50]$ and CCC adjustment. Thus, the comparison changes the energy profile used for horizon selection without reducing the number of sampling steps. Late-5 achieves higher overall success than Full-10 (49.36% versus 47.79%), with gains in three of the four task groups. Because the generation process is unchanged, this comparison isolates the denoising interval used to guide execution. The higher aggregate success supports focusing on corrections near the final action predictions when selecting the horizon.

Denoising Steps. Index the ten Euler updates by $k = 0, \ldots, 9$ in generation order. Under the paper's noise-to-action convention, update k evaluates velocity at $\tau_k = k/10$ and advances to τ_{k+1}. Late-5 uses $k = 5, \ldots, 9$, covering $[0.5, 1]$; Full-10 uses $k = 0, \ldots, 9$, covering $[0, 1]$. The implementation's solver time decreases from 1 to 0 and equals $1 - \tau$, so the Late-5 velocity evaluations occur at solver times $0.5, 0.4, 0.3, 0.2, 0.1$.

H Limitations and Future Work

KPE-AC is designed for flow-based VLA policies and requires access to intermediate velocities during action generation. Extending this approach to other policy architectures would require iden-

tifying suitable internal signals for execution control. Our method also focuses on horizon selection and leaves the generated actions unchanged, so its effectiveness depends on the capabilities of the underlying policy. Combining generation-based execution signals with action refinement or recovery mechanisms is a promising direction for extending the approach to more demanding manipulation tasks.