# ACCELERATING VLA INFERENCE WITH PERTURBATION INFORMED EXECUTION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Action-chunking vision-language-action (VLA) models predict sequences of future robot actions. Under fixed-horizon execution, the robot executes a fixed-length prefix of each predicted chunk before invoking the policy again, leaving the remaining predictions unused. For a given number of executed actions, short execution horizons require more policy calls. Extending the execution horizon amortizes policy inference cost over more actions. However, uniformly applying long horizons across episodes can limit opportunities for precise corrections. To address this limitation, we propose PICES (Perturbation-Informed Chunk and Execution Horizon Selection), a training-free method that jointly selects an action chunk and the number of actions executed before the next policy call. Within a single policy call, the action expert generates a batch of action chunks from multiple initial noise samples and their locally perturbed counterparts, using the same conditioning features across all samples. PICES exploits differences between corresponding actions in each nominal chunk and its counterpart to determine candidate-specific execution horizons and select a nominal chunk for execution. Experiments on simulation benchmarks and real-world manipulation tasks demonstrate that PICES achieves speedup by up to 2.3× in wall-clock policy inference time on Jetson Orin and Thor relative to fixed-horizon baselines while improving the success rate. The code is available at : `https://anonymous.4open.science/r/PICES_VLA-B1BF`

## 1 INTRODUCTION

Vision-language-action (VLA) models generate robot actions conditioned on visual observations, proprioceptive states, and language instructions (Brohan et al., 2022; Kim et al., 2024; 2025). Recent VLA architectures pair a vision-language backbone with a flow-matching action expert that transforms initial noise into an action chunk, a sequence of future actions (Bjorck et al., 2025; Black et al., 2024; Intelligence et al., 2025; Shukor et al., 2025). In conventional action chunking, the robot executes fixed-length actions of each predicted chunk before replanning as illustrated in Figure 1(a). The number of actions executed before update observations defines the execution horizon (Black et al., 2026). The cumulative policy invocations across policy rollout makes task-completion time dependent on both per-call latency and the number of policy calls (Zhou et al., 2026; Jing et al., 2025).

At comparable success rates, shorter task-completion times enable a robot to complete more tasks within a fixed time budget (Shukor et al., 2025; Huang et al., 2025). This makes cumulative inference latency as an important consideration for onboard VLA deployment, where edge platforms such as NVIDIA Jetson Orin and Thor operate under compute and memory-bandwidth constraints (Yang et al., 2026; Jiang et al., 2026). For example, GR00T N1.7 requires 80.4ms per policy call on Jetson Orin and 150.9 ms on Jetson Thor (Li et al., 2026; Bjorck et al., 2025), whereas observation acquisition and actuation can operate up to 60Hz and 1kHz, respectively (Jiang et al., 2026; Ma et al., 2025; Zhang et al., 2020). In observation-inference-execution pipeline, reducing the number of policy calls lowers cumulative inference latency and shorten task-completion time (Jing et al., 2025). For action chunking policies, executing only fraction of each predicted action chunk incurs inefficiency. However, applying long horizons uniformly across task phases delays the incorporation of updated observations when precise control is required. This motivates adapting execution horizons to reduce repeated inference while preserving task success.



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.

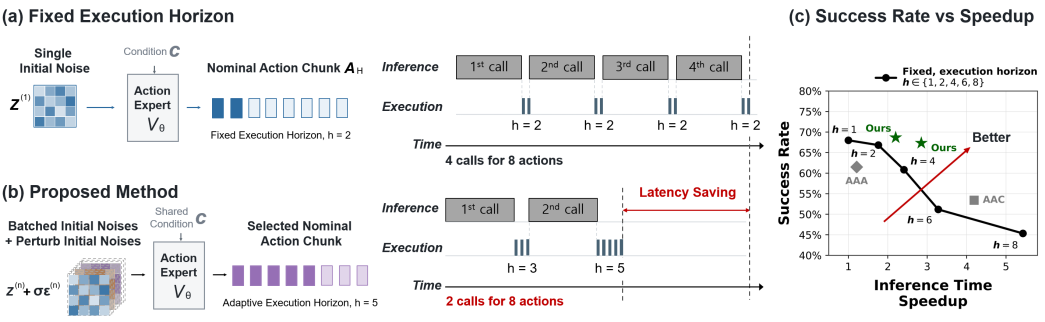


Figure 1: **High-level idea of the PICES.** (a) Fixed execution horizon generates a nominal action chunk from a single initial noise sample at each policy call and executes a fixed-length actions before replanning. (b) PICES uses multiple initial-noise samples and their perturbations to determine execution horizon of nominal action chunk. Our method reduces the number of policy calls required to complete the same episode, from four to two in the illustrated example. (c) Success rate versus speedup comparison across methods, showing that PICES achieves wall-clock inference time speedup while preserving success rate.

Prior works have explored diverse action-chunk execution strategies to balance temporal consistency and reactivity, including temporal ensembling, adaptive chunk updates, and guided candidate selection (Zhao et al., 2023; So et al., 2026; Liu et al., 2025a). AAC estimates action entropy of action candidates to decide a shared exceuction horizon (Liang et al., 2026). AutoHorizon derives the execution horizon from self-attention within a predicted chunk (Wang et al., 2026). $A^3$ selects a representative trajectory from multiple candidates and then performs additional flow-matching verification to determine execution horizon (Zhao et al., 2026). Our objective is to reduce wall-clock inference time while achieving comparable success rate by jointly selecting an action chunk and its execution horizon. This raises our key question: Given a bounded set of candidate action chunks generated in a single policy call, how can we select the action chunk and execution horizon to reduce the wall-clock inference time?

We propose PICES (Perturbation-Informed Chunk and Execution Horizon Selection), a training-free inference time method that jointly selects an action chunk and its execution horizon. For each initial-noise sample, PICES constructs a locally perturbed counterpart by adding a small Gaussian perturbation. A single batched policy call generates the corresponding nominal and auxiliary action chunks using shared conditioning features encoded by the vision-language-model (VLM). Based on the arithmetic intensity analysis of action expert and hardware ridge-point in our motivational analysis, we restrict total batch size to a range with negligible incremental latency. For each candidate, the stepwise differences between nominal and auxiliary action chunks yield a local sensitivity value. We exploit these values to retain candidates with low initial sensitivity and estimate execution horizon for each candidate. From the candidate pool, PICES selects and executes the nominal chunk with the longest execution horizon. Figure 1(b) illustrates the reduction in policy calls: two calls execute eight actions, compared with four calls with fixed horizon execution. Savings from fewer policy calls outweigh the additional cost of candidate generation, reducing cumulative inference time as shown in Figure 1(c).

Our key contributions are as follows:

- Through an open-loop study, we show that oracle selection from multiple action chunks can yield lower reference-action error over a longer execution horizon than a single-candidate, short-horizon baseline. We also characterize the cost of generating candidates in a batch.
- We propose PICES, a training-free method to reduce cumulative wall-clock inference time. PICES uses sensitivity to initial-noise perturbations to guide the joint selection of an action chunk and its execution horizon.
- We evaluate PICES across multiple VLA models in simulation and on real-world robot tasks. Compared with fixed-horizon baselines, PICES reduces policy calls by 25.4–60.3% and achieves 1.2–2.3× speedups in wall-clock inference time on Jetson Orin and Thor, while maintaining higher success rates.

## 2 Related Work

**Action Chunking in VLA Models.** Recent VLAs such as CogACT, $\pi$, and GR00T series execute only a fixed-length horizon of each predicted chunk (Li et al., 2024a; Black et al., 2024; Bjorck et al., 2025). Executing a long fixed-length actions reduces policy calls but limits the incorporation of new observations, motivating adaptive execution horizons (Liang et al., 2026; Wang et al., 2026; Chen et al., 2026) (see Appendix G for full discussion). In contrast, our method instead estimates a horizon for each candidate and uses the estimated execution length to choose the reliable action chunk to shorten cumulative inference time without training.

**Acceleration for VLA Models.** Efficient VLA inference seeks to reduce task completion time using various methods. To reduce inference overhead, previous work exploits redundancy in visual inputs and model computation through token pruning, channel pruning, and model scheduling (Chen et al., 2024; Shukor et al., 2025; Xu et al., 2026; Feng et al., 2026) (see Appendix G for full discussion). Our method is orthogonal to token pruning, which reduces computation within individual policy calls. PICES examines how candidate-specific horizon selection can reduce policy calls, while explicitly accounting for the additional cost of generating and assessing multiple action candidates.

## 3 Proposed Approach

### 3.1 Background

**Input Condition and Action Chunk.** Let $\boldsymbol{c} := \big(\boldsymbol{o}, \boldsymbol{q}, \ell\big) \in \mathcal{C}$ denote the conditioning input, comprising the current observation $\boldsymbol{o}$, the proprioceptive state $\boldsymbol{q}$, and the language instruction $\ell$. Let $V_\theta$ denote a pretrained VLA action generator parameterized by $\theta$, $V_\theta : \mathcal{C} \times \mathbb{R}^{d\times H} \longrightarrow \mathbb{R}^{d\times H}$. Here, $d$ is the dimension of each action and $H$ is the number of action steps generated in one action chunk. Given an initial noise sample $\boldsymbol{Z} \in \mathbb{R}^{d\times H}$, the model generates $\boldsymbol{A} = [\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{H-1}] \in \mathbb{R}^{d\times H}$ To generate $N$ candidate action chunks from the same current input, sample $N$ independent initial noise matrices $\boldsymbol{Z}^{(n)} \in \mathbb{R}^{d\times H}$. The $n$ indicates the candidate index. All candidates share the same conditioning prefix $\boldsymbol{c}$ and differ only in their initial noise samples:

$$\boldsymbol{A}^{(n)} = V_\theta\Big(\boldsymbol{c}, \boldsymbol{Z}^{(n)}\Big) = \Big[\boldsymbol{a}_0^{(n)}, \boldsymbol{a}_1^{(n)}, \ldots, \boldsymbol{a}_{H-1}^{(n)}\Big] \in \mathbb{R}^{d\times H}, \qquad n = 1, \ldots, N. \tag{1}$$

Equivalently, conditioned on the same current visual observation, proprioceptive state, and language instruction, $N$ independent source samples are mapped into $N$ candidate action chunks with $H$ number of actions in single action chunk, whereas $h \in \{1, \ldots, H\}$ denotes the execution horizon.

### 3.2 Motivational Study

We analyze whether sampling multiple action chunks under identical conditioning inputs can provide suitable candidates for longer execution horizon than fixed-length execution. To limit the latency overhead of this additional sampling, we explore affordable batch size during chunk generation. This motivates two questions: (1) Can multiple sampled chunks provide a lower reference-action error at a longer fixed execution horizon candidate than a single sampled chunk? (2) Under consideration of action expert's arithmetic intensity, how many candidate chunks can we expect to be generated in a batch with modest additional latency?

**Selection by minimum reference-action RMSE.** We first compare reference action root-mean-square-error (RMSE) across fixed execution horizons and candidate counts. We conduct the analysis with the RT-1 Fractal dataset and GR00T N1.7 (Bjorck et al., 2025; Brohan et al., 2022). For each condition $\boldsymbol{c}$, we generate $N \in 1, \ldots, 10$ candidate action chunks using independent initial noise and evaluate horizons $h \in \{1, \ldots, 8\}$. For each input and $(N, h)$ pair, an oracle selects the candidate with the lowest RMSE over its first $h$ actions relative to the corresponding reference actions. The RMSE metric is explained in Appendix C. Figure 2 shows that selection from four candidates at an eight-step horizon ($N = 4$, $h = 8$) achieves an RMSE of $0.1795$, compared with $0.1900$ for the single-candidate action chunk at one-step horizon ($N = 1$, $h = 1$). This result demonstrates that chunk selection from multiple candidates can yield lower reference-action error even when evaluated over a longer execution horizon, motivating an inference-time method for joint action chunk and execution horizon selection.

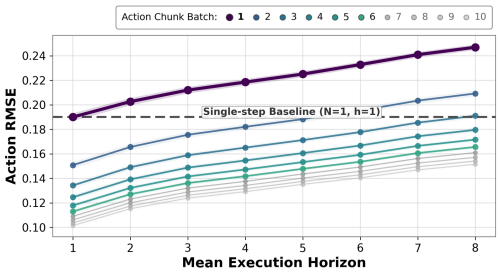


Figure 2: **Root-mean-square-error (RMSE) between reference and oracle-selected action chunk.** The dashed horizontal line marks the baseline single-candidate, one-step baselin ($N = 1, h = 1$).

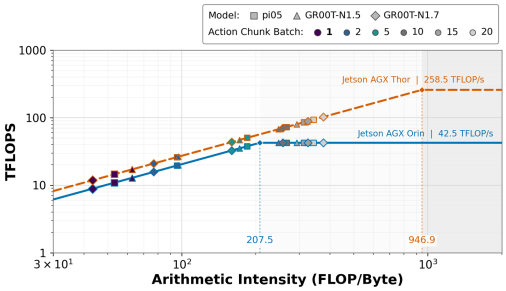


Figure 3: **Roofline analysis of batching action generation on Jetson Orin and Thor.** For each device, arithmetic intensities over its ridge point lie in the compute-bound region, where compute throughput determines the latency.

**Operation Characteristics of Action Expert.** Cumulative inference-time saving for the longer execution horizons requires generating action chunks with limited additional latency. Prior studies identify low arithmetic intensity (AI) of action generation in action expert (Jiang et al., 2026; Zhou et al., 2026). The roofline in Figure 3 shows the AI of the $\pi_{0.5}$, GR00T N1.5, and GR00T N1.7 action experts at batch sizes $B \in \{1, 2, 5, 10, 15, 20\}$. At single batch, the AI of all three action experts lies below the ridge points of both Jetson Orin and Thor, indicating memory-bound operation under the roofline model. When AI remains below the device's ridge point, the operation latency depends primarily on memory transfer latency. In such a case, the additional computation can be accommodated without a proportional increase in latency. On Orin, all three models approach the ridge point at a batch size of 6. On Thor, the action experts remain memory-bound even at a batch size of 20; however, beyond a batch size of 10, growing activation traffic incurs latency overhead. We therefore set the total generation batch-size limits to 6 on Orin and 10 on Thor to balance candidate diversity against per-call latency. Appendix B provides the further analysis of arithmetic intensity and measured latency on GPUs.

Together, these analyses suggest that a bounded set of action chunks can contain candidates suitable for longer execution horizons with modest cost for the batched generation of candidates. This raises the central question: how can we assess the reliability of each candidate's future action steps and jointly select a chunk and its execution horizon?

### 3.3 Perturbation-Informed Candidate Assessment

For each nominal action chunk, we measure changes in the action expert output under local perturbations of the corresponding initial noise. Local sensitivity quantifies output variation under small input changes, and exploiting this sensitivity has been used to encourage invariant representations (Rifai, 2011). In image generation, per-image perceptual path length quantifies how much the generated image changes perceptually under small perturbations of its latent code (Karras et al., 2020). Motivated by these observations, we use local stability under initial-noise perturbations as a proxy for reliability of each nominal action. We estimate sensitivity at each action timestep within a chunk and use these estimates to favor nominal candidates with lower sensitivity.

**Step 1 & 2. Local perturbations of the initial noise and multiple action chunks.** Comparing generator outputs at neighboring latent inputs provides a practical way to probe local sensitivity (Odena et al., 2018). Related analyses of diffusion sampling connect initial-noise perturbations to changes in the final generated output (Song et al., 2026). These analyses motivate our use of initial-noise perturbations to assess the local stability of nominal action chunks. For each nominal candidate $\boldsymbol{A}_t^{(n)} = V_\theta(\boldsymbol{c}, \boldsymbol{Z}^{(n)})$, we sample perturbation matrices $\boldsymbol{\epsilon}^{(n,r)} \in \mathbb{R}^{d \times H}$ with i.i.d. $\mathcal{N}(0, 1)$ entries. All perturbation matrices are mutually independent and independent of the nominal initial noises.

$$\widetilde{\boldsymbol{Z}}^{(n)} = \boldsymbol{Z}^{(n)} + \sigma \boldsymbol{\epsilon}^{(n)}, \qquad \widetilde{\boldsymbol{A}}_t^{(n)} = V_\theta\left(\boldsymbol{c}, \widetilde{\boldsymbol{Z}}^{(n)}\right), \tag{2}$$

where $\sigma > 0$ controls the perturbation scale. Perturbations are applied only to the initial noise before flow integration. We stack all nominal initial noises $\boldsymbol{Z}^{(n)}$ and their perturbed counterparts $\widetilde{\boldsymbol{Z}}^{(n)}$ into

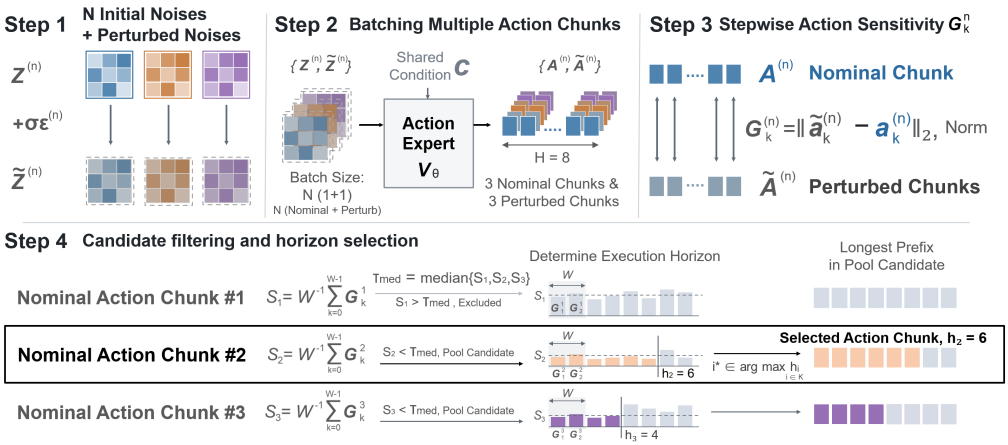


Figure 4: **Overview of PICES. Step 1:** Draw $N$ initial noise samples and construct a perturbed variant of each sample. **Step 2:** The action expert processes the resulting $N(1+1)$ inputs in a single batch under the shared condition $C$, producing $N$ nominal action chunks and their corresponding perturbed chunks. **Step 3:** Stepwise action sensitivity $G_k^{(n)}$ is computed by comparing each nominal chunk with its perturbed counterparts. **Step 4:** Compute a sensitivity score $S_i$ for each nominal chunk by averaging its sensitivity over the first $W$ steps and use this score to determine its execution horizon $h_i$. PICES retains candidates whose scores are at or below the median and selects the nominal chunk with the longest execution horizon from the pool.

a single batch. All nominal and perturbed initial-noise samples are processed in a single batched policy call with a shared conditioning input $\boldsymbol{c}$. For each candidate $n$, we estimate sensitivity at each action timestep by comparing the nominal chunk $\boldsymbol{A}^{(n)}$ with its auxiliary chunk $\widetilde{\boldsymbol{A}}^{(n)}$. Only actions from the unmodified nominal chunks are used for execution.

**Step 3. Stepwise action sensitivity score.** Write the nominal action chunk as $\boldsymbol{A}^{(n)} = [\boldsymbol{a}_0^{(n)}, \ldots, \boldsymbol{a}_{H-1}^{(n)}] \in \mathbb{R}^{d\times H}$, where $\boldsymbol{a}_k^{(n)} \in \mathbb{R}^d$ is the action at offset $k \in \{0, \ldots, H-1\}$. Define $\widetilde{\boldsymbol{a}}_k^{(n)}$ analogously from the auxiliary chunk $\widetilde{\boldsymbol{A}}^{(n)}$. The execution horizon $h$ denotes the number of leading actions executed from the selected chunk. With the conditioning input $\boldsymbol{c}$ fixed, we define

$$G_k^{(n)} = \frac{\left\| \widetilde{\boldsymbol{a}}_k^{(n)} - \boldsymbol{a}_k^{(n)} \right\|^2}{\sigma^2 \left\| \boldsymbol{\epsilon}^{(n)} \right\|_F^2}. \tag{3}$$

The score normalizes the action difference between the nominal and auxiliary actions by the squared displacement of the initial-noise perturbation. Appendix D provides the connection between squared norm of the directional derivative of the action at offset $k$ for sufficiently small perturbations.

We use the resulting stepwise scores $\{G_k^{(n)}\}_{k=0}^{H-1}$ to guide joint nominal action chunk and its execution horizon selection. The scores are computed from the paired outputs generated in Steps 1 and 2, without explicitly constructing Jacobians or training a separate evaluator.

### 3.4 Joint Action Chunk and Execution Horizon Selection

**Step 4. Candidate filtering and horizon selection.** We filter nominal candidates by their initial-window sensitivity, then select the candidate with the longest execution horizon among those retained. For each candidate $n$, define

$$S_n = \frac{1}{W} \sum_{k=0}^{W-1} G_k^{(n)}, \tag{4}$$

where $W$ is the number of initial action timesteps included in the average, with $1 \le W \le H$. The same averaging window is used for all candidates. We retain all candidates whose scores do not exceed the median across the $N$ nominal candidates:

$$\mathcal{K} = \{n \in \{1, \ldots, N\} : S_n \le \text{median}(S_1, \ldots, S_N)\}. \tag{5}$$

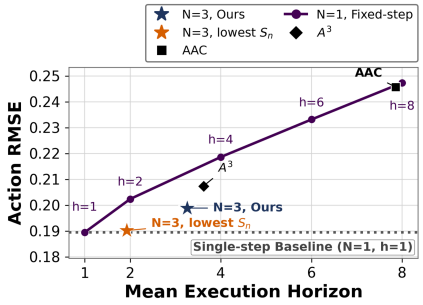


Figure 5: **Open-loop evaluation of selected action horizons.** Action RMSE is computed between each selected nominal action and the corresponding reference action, following the evaluation protocol of the motivational study. Each fixed-step point is based on the single action chunk. For $A^3$, AAC, PICES, and the lowest-$S_n$ variant, the horizontal coordinate indicates the average of the horizons selected for individual method. PICES and the lowest-$S_n$ variant use $N = 3$ nominal candidates; the latter selects the candidate with the smallest initial-window score ($S_n$).

For each retained candidate, we determine an execution horizon using its initial-window average $S_n$ as a sensitivity threshold. Let $H$ denote the maximum execution length, we define

$$h_n = \min\big(\{k \in \{W, \dots, H-1\} : G_k^{(n)} > S_n\} \cup \{H\}\big) \tag{6}$$

If $k$ is the first offset in $\{W, \dots, H-1\}$ satisfying $G_k^{(n)} > S_n$, we set $h_n = k$. If no crossing occurs before the execution cap, $h_n = H$. The selected nominal chunk is $\boldsymbol{A}^{(n^\star)}$, and its execution horizon is $h^\star = h_{n^\star}$. We use the first $h^\star$ actions of the chunk, $[\boldsymbol{a}_0^{(n^\star)}, \dots, \boldsymbol{a}_{h^\star-1}^{(n^\star)}]$, as the control sequence. After executing the action sequence, we obtain an updated observation and repeat the candidate generation, assessment, and selection procedure. Each policy call thus provides $h^\star$ actions before replanning (Algorithm in Appendix H).

### 3.5 Open-loop Analysis of Chunk and Horizon Selection

**Effect of the candidate selection rule.** We evaluate open-loop action prediction error across methods with fixed and adaptive execution horizons. For each method, we compute RMSE between the first $h$ predicted actions and the corresponding recorded actions, following the evaluation protocol of the motivational study. The fixed-step baseline uses a prescribed horizon with single action chunk. Figure 5 reports action prediction RMSE against the average execution horizon. The lowest-$S_i$ method obtains reference-action RMSE close to the single-step baseline while selecting approximately 1.9 actions on average. This observation supports the usefulness of sensitivity-guided candidate assessment for identifying action chunks with low reference-action error. PICES retains candidates with low initial sensitivity and selects the candidate with the longest estimated execution horizon. The resulting average horizon increases to 3.3 actions, while the observed RMSE remains below that of the single-candidate, two-step execution horizon baseline.

## 4 Experiments

### 4.1 Experimental Setups

**Models and benchmarks.** In simulation, we evaluate PICES with GR00T N1.7 on SimplerEnv–Fractal, $\pi_{0.5}$ and GR00T N1.5 on LIBERO. For each model–benchmark configuration, we retain the action dimensionality specified by the corresponding official repository and use default execution horizon for the fixed-horizon baseline. We further evaluate $\pi_{0.5}$ on three real-world manipulation tasks using a UR10e robot arm.

**Baselines and PICES configuration.** We compare PICES with fixed-horizon execution, AAC, and $A^3$ based on each official repository. We also evaluate AAC$^*$ and $A^{3*}$, whose generation budgets match PICES's total number of nominal and perturbed action chunks. Unless otherwise specified, PICES uses $N = 3$ nominal candidates, perturbation scale of $\sigma = 0.25$ and reference-window $W$ as default execution horizon specified by each model's official repository for the corresponding benchmark. Full model configurations, benchmark descriptions, evaluation protocols, and real-world task settings are provided in Appendix E.

### 4.2 Quantitative Analyses

**PICES improves task success while reducing total inference time.** On SimplerEnv–Fractal with GR00T N1.7, PICES ($N = 3$) achieves $68.7\%$ success with $60.3\%$ fewer policy calls, yielding inference-time speedups of $2.2\times$ on Jetson Orin and $2.2\times$ on Jetson Thor relative to the single-step baseline (Table 1). The reported speedups account for each method's measured per-call GPU

Table 1: **Main results on SimplerEnv–Fractal with GR00T N1.7.** We evaluate six Google Robot tasks with 100 episodes per task. We report task success rates (%) and mean policy calls over successful episodes, together with per-call GPU latency and inference-time speedup on Jetson Orin and Thor. Speedup is relative to default fixed execution horizon. The best success rates and speedups are highlighted in bold.

| Method | Success rate (%) ↑ | | | | | | | Policy calls ↓ | Jetson Orin | | Jetson Thor | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pick Coke can | Pick object | Move near | Open drawer | Close drawer | Place in closed drawer | **Average** | | Single call (ms) ↓ | Speedup ↑ | Single call (ms) ↓ | Speedup ↑ |
| Fixed ($h = 1$) | **99** | **83** | **97** | 54 | 65 | 13 | 68.5 | 46.8 | | 1.0 | | 1.0 |
| Fixed ($h = 2$) | **99** | 80 | 92 | 57 | 67 | 15 | 68.3 | 26.6 | | 1.8 | | 1.8 |
| Fixed ($h = 4$) | 82 | 64 | 72 | 52 | 77 | **18** | 60.8 | 19.4 | 209.9 | 2.4 | 101.1 | 2.4 |
| Fixed ($h = 6$) | 66 | 48 | 55 | 47 | 77 | 14 | 51.2 | 14.3 | | 3.3 | | 3.3 |
| Fixed ($h = 8$) | 65 | 42 | 44 | 35 | 78 | 8 | 45.3 | 8.6 | | 5.4 | | 5.4 |
| AAC | 58 | 42 | 56 | 43 | 76 | 9 | 47.3 | 10.1 | 417.1 | 2.3 | 181.7 | 2.6 |
| AAC* | 58 | 36 | 64 | 48 | 77 | 17 | 50.0 | 9.7 | 256.7 | **3.9** | 132.3 | **3.7** |
| AutoHorizon | 67 | 47 | 62 | 44 | 74 | 8 | 50.3 | 12.3 | 212.6 | 3.7 | 102.6 | **3.7** |
| $A^3$ | 79 | 56 | 71 | **56** | 75 | 13 | 58.3 | 18.8 | 552.8 | 0.9 | 234.8 | 1.1 |
| $A^{3*}$ | 92 | 66 | 80 | 44 | 71 | 16 | 61.5 | 20.4 | 398.1 | 1.2 | 192.2 | 1.2 |
| **PICES** | 97 | 82 | 91 | 53 | **79** | 10 | **68.7** | 18.6 | 242.7 | 2.2 | 113.4 | 2.3 |
| **PICES**** | **99** | 74 | 90 | 52 | **79** | 10 | 67.3 | 14.3 | 242.7 | 2.8 | 113.4 | 2.9 |

*Same total number of batched action samples as PICES, including nominal and perturbed chunks.
**PICES with $W = 2$.

latency. Increasing the fixed execution horizon to $h = 8$, success rate falls to $45.3\%$, a decrease of $23.2\%$ from the single-step baseline. Compared with the fixed-horizon baseline ($h = 2$), PICES achieves inference-time speedups of $1.3\times$ on both Jetson Orin and Thor. These results demonstrate PICES improves the trade-off between task success and inference time relative to the fixed-horizon baselines.

**Comparison with dynamic execution baselines.** PICES also achieves higher task success than the competing adaptive execution methods. AAC and $A^3$ reduce the average number of policy calls to $10.1$ and $18.8$, respectively. However, their success rates are $47.3\%$ and $58.3\%$, compared with $68.7\%$ for PICES ($N = 3$). The success-rate gap persists under matching sampling action budgets to PICES (including nominal and auxiliary chunks). AAC* and $A^{3*}$ achieve success rates of $50.0\%$ and $61.5\%$, respectively, remaining $18.7\%$ and $7.2\%$ below PICES.

Table 2: **Results on LIBERO with $\pi_{0.5}$ and GR00T N1.5.** We report average success rates (SR, %) and policy calls across the four LIBERO suites, together with per-call GPU latency and inference-time speedup on Jetson Orin and Thor.

| Method | $\pi_{0.5}$ | | | | | | GR00T N1.5 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Avg. SR (%)** ↑ | **Avg. Policy Calls** ↓ | **Jetson Orin** | | **Jetson Thor** | | **Avg. SR (%)** ↑ | **Avg. Policy Calls** ↓ | **Jetson Orin** | | **Jetson Thor** | |
| | | | Single call (ms) ↓ | Speedup ↑ | Single call (ms) ↓ | Speedup ↑ | | | Single call (ms) ↓ | Speedup ↑ | Single call (ms) ↓ | Speedup ↑ |
| Fixed ($h = 5$) | 96.5 | 30.2 | 297.2 | 1.0 | 126.7 | 1.0 | 92.9 | 30.1 | 149.3 | 1.0 | 62.8 | 1.0 |
| AAC | 96.4 | 20.6 | 359.2 | 1.2 | 152.4 | 1.2 | 93.3 | 24.2 | 282.5 | 0.7 | 107.2 | 0.7 |
| AAC* | 96.0 | 21.4 | 316.0 | 1.3 | 137.4 | 1.3 | 92.8 | 25.8 | 185.9 | 0.9 | 77.7 | 0.9 |
| AutoHorizon | 97.2 | 21.6 | 300.4 | 1.4 | 127.8 | 1.4 | – | – | – | – | – | – |
| $A^3$ | 97.2 | **15.3** | 455.8 | 1.3 | 199.7 | 1.3 | **94.9** | 29.7 | 277.7 | 0.5 | 124.9 | 0.5 |
| $A^{3*}$ | 96.0 | 29.2 | 421.6 | 0.7 | 187.3 | 0.7 | 93.0 | 50.2 | 235.9 | 0.4 | 100.6 | 0.4 |
| **PICES** | 97.3 | 19.9 | 311.4 | 1.5 | 131.6 | 1.5 | 94.5 | 21.4 | 178.5 | 1.2 | 74.8 | 1.2 |
| **PICES**** | **97.5** | 16.4 | 311.4 | **1.8** | 131.6 | **1.8** | 94.4 | **15.4** | 178.5 | **1.6** | 74.8 | **1.6** |

*Same total number of batched action samples as PICES, including nominal and perturbed chunks.
**PICES with $W = 8$.

**Evaluation on LIBERO.** On LIBERO, PICES improves success over Fixed ($h = 5$) from 96.5% to 97.3% with $\pi_{0.5}$ and from 92.9% to 94.5% with GR00T N1.5 (Table 2). Average policy calls decrease by 34.1% and 28.9%, respectively. These reductions yield estimated inference-time speedups of $1.5\times$ on both devices with $\pi_{0.5}$, and $1.2\times$ with GR00T N1.5. With $\pi_{0.5}$, $A^3$ achieves the lowest policy calls, but additional flow-matching computations for verification increase per-call latency 110 ms on Orin and 58 ms on Thor. This overhead limits inference-time speedup to $1.3\times$ on both devices, below the $1.5\times$ achieved by PICES. With GR00T N1.5 AAC achieves a slightly higher success rate than fixed-horizon baseline, but requires higher per-call latency, resulting in $0.7\times$ speedup on Orin and Thor. PICES improves success rate while achiving speedup on both devices.

Table 3: **Real-world task results using UR10e arm with $\pi_{0.5}$.** We report per-task and average success rates (SR, %) and policy calls for three tasks, together with per-call GPU latency and inference-time speedup on Jetson Orin and Thor. Inference-time speedup is computed from the product of average policy calls and per-call GPU latency on each device, relative to Fixed ($h = 8$).

| Method | Success rate (%) ↑ | | | | Policy calls ↓ | | | | Jetson Orin | | Jetson Thor | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Tape | Stapler | Cube | **Average** | Tape | Stapler | Cube | **Average** | Single call (ms) ↓ | Speedup ↑ | Single call (ms) ↓ | Speedup ↑ |
| Fixed ($h = 4$) | **97.2** | 77.8 | 58.3 | 77.8 | 614.8 | 236.8 | 485.9 | 445.8 | 298.5 | 0.39 | 126.7 | 0.39 |
| Fixed ($h = 8$) | 86.1 | **94.4** | 91.7 | 90.7 | 222.9 | 104.8 | 196.9 | 174.9 | | 1.00 | | 1.00 |
| Fixed ($h = 12$) | 88.9 | 80.6 | 88.9 | 86.1 | 120.8 | 74.4 | 111.1 | 102.1 | | 1.71 | | 1.71 |
| Fixed ($h = 16$) | 77.8 | 66.7 | 86.1 | 76.9 | 95.8 | 68.6 | 74.8 | 76.4 | | 2.29 | | 2.29 |
| **PICES** | 88.9 | **94.4** | **97.2** | **93.5** | 148.9 | 72.6 | 134.5 | 118.7 | 314.7 | 1.39 | 137.3 | 1.36 |

**Evaluation on Real-world tasks.** We evaluate PICES with $\pi_{0.5}$ on three manipulation tasks using a UR10e robot arm. Among the tested fixed-horizon baselines, $h = 8$ achieves the highest average success rate of 90.7%, as shown in Table 3. We set the reference-window length of PICES to $W = 8$ and use Fixed ($h = 8$) as the baseline for comparisons. PICES increases the average success rate to 93.5%, an improvement of 2.8 percentage points. PICES also reduces the average number of policy calls per episode from 174.9 to 118.7, a reduction of 32.1%. Consequently, the reduction in policy calls yields the speedups of 1.39× and 1.36× on Orin and Thor, respectively, relative to Fixed ($h = 8$). These results demonstrate that the reduction at inference demand extends across the evaluated VLA backbones and manipulation benchmarks.

### 4.3 Additional Analyses

Table 4: **Effect of the number of nominal candidates on SimplerEnv–Fractal with GR00T N1.7.** We report average task success rates (%) and average policy calls, together with per-call GPU latency and inference-time speedup relative to fixed execution horizon on Jetson Orin and Thor.

| Method | Avg. SR (%) ↑ | Avg. Policy Calls ↓ | Jetson Orin | | Jetson Thor | |
|---|---|---|---|---|---|---|
| | | | Single call (ms) ↓ | Speedup ↑ | Single call (ms) ↓ | Speedup ↑ |
| Fixed ($h = 1$) | 68.5 | 46.7 | 209.8 | 1.0 | 101.1 | 1.0 |
| **PICES**, $N = 1$ | 67.3 | 23.4 | 214.3 | 2.0 | 106.9 | 1.9 |
| **PICES**, $N = 3$ | 68.6 | 18.5 | 242.6 | **2.2** | 113.4 | **2.3** |
| **PICES**, $N = 5$ | 68.5 | 17.4 | 303.6 | 1.9 | 129.8 | 2.1 |
| **PICES**, $N = 7$ | 68.0 | 16.8 | 357.7 | 1.6 | 153.0 | 1.8 |
| **PICES**, $N = 9$ | **69.0** | **16.1** | 397.8 | 1.5 | 161.8 | 1.8 |

**Effect of the number of nominal candidates.** Table 4 evaluates PICES with $N \in \{1, 3, 5, 7, 9\}$ on SimplerEnv–Fractal using GR00T N1.7. Each policy call generates $2N$ chunks, comprising $N$ nominal and $N$ auxiliary chunks. As $N$ increases from 1 to 9, the average number of policy calls per successful episode decreases from 23.4 to 16.1, a reduction of 31.1%. On Orin, $N = 3$ adds 5.1 ms per action sample generation. At $N = 9$, increased per action sample costs (10.7 ms) outweigh further policy-call savings, consistent with the compute-bound behavior on Orin. Consequently, $N = 3$ achieves 2.2× speedup on Orin and 2.3× on Thor.

Table 5: **Compatibility of PICES with VLA acceleration methods on $\pi_{0.5}$.** We report success rates (%) and average policy calls per episode across four LIBERO suites, together with per-call GPU latency and total inference-time speedup on Jetson Orin and Thor. Speedup is measured relative to Fixed ($h = 5$) without visual token compression on the same device.

| Acceleration Method | Success rate (%) ↑ | | | | | Policy calls ↓ | | | | | Jetson Orin | | Jetson Thor | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Spatial | Goal | Object | Long | **Avg.** | Spatial | Goal | Object | Long | **Avg.** | Single call (ms) ↓ | Speedup ↑ | Single call (ms) ↓ | Speedup ↑ |
| Baseline, Fixed | 99.0 | 97.4 | 97.2 | 92.2 | 96.5 | 20.6 | 21.8 | 27.7 | 50.6 | 30.2 | 297.2 | 1.0 | 126.7 | 1.0 |
| **PICES** | 99.0 | 98.4 | 99.6 | 92.2 | **97.3** | 13.9 | 14.4 | 18.0 | 33.4 | **19.9** | 311.4 | 1.5 | 131.6 | 1.5 |
| FastV (40%) | 99.0 | 97.4 | 98.0 | 89.6 | 96.0 | 20.6 | 27.7 | 22.3 | 50.1 | 30.2 | 237.5 | 1.3 | 106.3 | 1.2 |
| FastV (40%) + **PICES** | 98.4 | 99.2 | 97.0 | 92.2 | 96.7 | 14.0 | 18.0 | 14.7 | 33.5 | 20.1 | 247.6 | 1.8 | 111.8 | 1.7 |
| FastV (50%) | 97.6 | 98.8 | 95.2 | 90.6 | 95.6 | 21.2 | 27.6 | 22.3 | 51.2 | 30.6 | 217.1 | 1.4 | 97.3 | 1.3 |
| FastV (50%) + **PICES** | 98.0 | 98.2 | 95.4 | 92.0 | 95.9 | 14.1 | 18.2 | 14.9 | 33.8 | 20.2 | 228.5 | 1.9 | 100.8 | 1.9 |
| $\text{GridS}_{16}$ | 96.2 | 96.4 | 98.2 | 90.4 | 95.3 | 21.4 | 23.1 | 28.4 | 52.7 | 31.1 | 151.9 | 1.9 | 71.9 | 1.7 |
| $\text{GridS}_{16}$ + **PICES** | 98.2 | 98.8 | 96.2 | 92.4 | 96.4 | 14.2 | 18.2 | 14.6 | 33.8 | 20.2 | 160.5 | **2.8** | 75.4 | **2.5** |

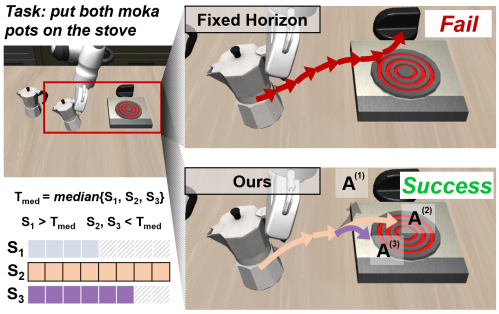


Figure 6: **Sensitivity-guided chunk selection in LIBERO–Long.** We compare Fixed ($h = 5$, top) and PICES (bottom) using GR00T N1.5. Colored arrows illustrate candidate motions, and the outlined bar identifies the selected execution horizon. PICES selects $\boldsymbol{A}^{(2)}$ over $\boldsymbol{A}^{(3)}$ for the longer execution horizon ($h_2 = 10$).

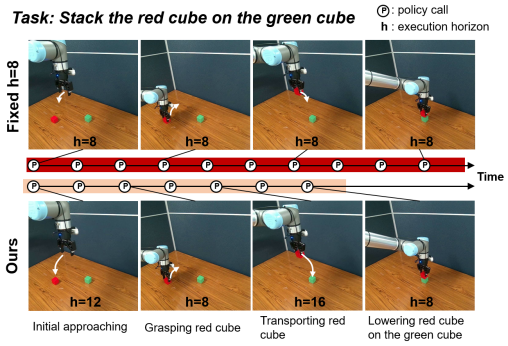


Figure 7: **Execution horizons across phases of real-world cube stacking.** We compare Fixed ($h = 8$, top) and PICES (bottom) using $\pi_{0.5}$ on the UR10e. White arrows indicate robot motion, and circled P markers denote policy calls. Circled P markers indicate policy calls, and $h$ denotes the execution horizon.

**Compatibility with VLA Acceleration Methods** We evaluate PICES combined with FastV and GridS on LIBERO using $\pi_{0.5}$ with a fixed-horizon $h = 5$ as the baseline. Both methods reduce per-call policy inference latency through visual token compression, using attention-based pruning for FastV and differentiable grid sampling for GridS. For FastV, we evaluate pruning ratios of 40% and 50%; For GridS, we use $\text{GridS}_{16}$, which samples 16 visual tokens per image.

All speedups are relative to the fixed-horizon baseline without token pruning. Table 5 shows that PICES further reduces policy calls while maintaining or improving average success rates across all configurations. Combined with FastV (40%), PICES achieves cumulative inference-time speedups of up to $1.8\times$ on Jetson Orin and $1.7\times$ on Jetson Thor; combined with $\text{GridS}_{16}$, the speedups reach $2.8\times$ and $2.5\times$, respectively. These results demonstrate that PICES provides additional efficiency gains on top of visual token compression.

### 4.4 Qualitative Analyses

**Sensitivity-guided chunk selection from action candidates.** Figure 6 compares Fixed ($h = 5$) and PICES using GR00T N1.5 on the LIBERO–Long task *put both moka pots on the stove*. The fixed-horizon baseline executes five actions from each sampled chunk, but the resulting motion fails to place the moka pot on the stove. PICES evaluates three nominal chunks $N = 3$ using the initial-window sensitivity scores, filtering out $\boldsymbol{A}^{(1)}$ with $S_1$ above the median. Between $\boldsymbol{A}^{(2)}$ and $\boldsymbol{A}^{(3)}$, PICES selects $\boldsymbol{A}^{(2)}$ with longer execution horizon ($h_2 = 10$). PICES executes the first ten actions of this nominal chunk, resulting in successful placement in the shown rollout.

**Execution horizons across phases.** Figure 7 compares fixed-horizon execution and PICES on *stack the red cube on the green cube* using $\pi_{0.5}$ with the UR10e arm. Given a prediction horizon of $H = 16$, the baseline uses $h = 8$ throughout execution, whereas PICES selects $h = 12$ during the initial approach and $h = 16$ while transporting the red cube. During fine-grained phases such as near grasping and placement, PICES selects $h = 8$, allowing replanning after fewer actions than coarse movement phases. Adaptive horizon execution beyond the baseline allows reducing the policy calls to shorten task completion relative to the fixed horizon.

## 5 Conclusion

We presented PICES, a training-free inference method for flow-matching VLA policies that jointly selects an action chunk and execution horizon. PICES estimates stepwise action sensitivity by comparing outputs generated from each initial noise and locally perturbed counterpart. PICES exploits these scores to filter candidates, determine execution horizon, and select the nominal chunk with the longest execution horizon. Experiments across multiple VLA backbones, simulation benchmarks, and real-world robot tasks show that PICES achieves higher task success rates while reducing cumulative inference time relative to the fixed-horizon baselines.

### AI USE STATEMENT

The research objectives, core ideas, and proposed method were developed by the authors without the use of generative AI tools. After independently formulating the research ideas, we used generative AI tools to help identify relevant prior work. We also used these tools to refine the manuscript's language and clarity and to assist with code implementation.

The authors reviewed and verified all AI-generated outputs. In particular, AI-assisted code was reviewed and checked for correctness by at least two authors. We take full responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the assistance of generative AI.

### REPRODUCIBILITY STATEMENT

The source code is available at `https://anonymous.4open.science/r/PICES_VLA-B1BF`. The joint action chunk and execution horizon selection procedure is described in Section 3.4, with Algorithm 1 provided in Appendix H. Model configurations, baseline settings, and benchmark evaluation protocols are documented in Appendix E. The open-loop evaluation procedure and reference-action error metric are detailed in Appendix C. GPU profiling settings and latency measurements are provided in Appendix B. The real-world platform, manipulation tasks, demonstration collection, policy fine-tuning, and evaluation protocol are described in Appendix F.

## REFERENCES

Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi_0$: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Feng Chen, Xianghui Wang, Yuxuan Chen, Boying Li, Yefei He, Zeyu Zhang, and Yicheng Wu. Dynamic execution commitment of vision-language-action models. *arXiv preprint arXiv:2605.11567*, 2026.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, pp. 19–35. Springer, 2024.

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. *arXiv preprint arXiv:2605.11817*, 2026.

Yiyang Huang, Yuhui Hao, Bo Yu, Feng Yan, Yuxin Yang, Feng Min, Yinhe Han, Lin Ma, Shaoshan Liu, Qiang Liu, et al. Dadu-corki: Algorithm-architecture co-design for embodied ai-powered robotic manipulation. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 327–343, 2025.

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

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_\{0.5\}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Wenqi Jiang, Jason Clemons, Karu Sankaralingam, and Christos Kozyrakis. How fast can i run my vla? demystifying vla inference performance with vla-perf. *arXiv preprint arXiv:2602.18397*, 2026.

Dong Jing, Gang Wang, Jiaqi Liu, Weiliang Tang, Zelong Sun, Yunchao Yao, Zhenyu Wei, Yunhui Liu, Zhiwu Lu, and Mingyu Ding. Mixture of horizons in action chunking. *arXiv preprint arXiv:2511.19433*, 2025.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8107–8116. IEEE, 2020.

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

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024a.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024b.

Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shu-Tao Xia, Zhi Wang, and Wenwu Zhu. Sp-vla: A joint model scheduling and token pruning approach for vla model acceleration. In *The Fourteenth International Conference on Learning Representations*, 2025.

Yinhao Li, Weixin Mao, Zihan Lan, Jikun Rong, Minzhao Zhu, Yiming Mao, Bowen Shen, and Xu Huang. FluxVLA Engine: A One-Stop VLA Engineering Platform for Embodied Intelligence, April 2026. URL `https://github.com/FluxVLA/FluxVLA`.

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. *arXiv preprint arXiv:2604.04161*, 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Yuejiang Liu, Jubayer Hamid, Annie Xie, Yoonho Lee, Max Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via guided test-time sampling. In *International Conference on Learning Representations*, volume 2025, pp. 4594–4627, 2025a.

Ziyan Liu, Yeqiu Chen, Hongyi Cai, Tao Lin, Shuo Yang, Zheng Liu, and Bo Zhao. Vla-pruner: Temporal-aware dual-level visual token pruning for efficient vision-language-action inference. *arXiv e-prints*, pp. arXiv–2511, 2025b.

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

Augustus Odena, Jacob Buckman, Catherine Olsson, Tom Brown, Christopher Olah, Colin Raffel, and Ian Goodfellow. Is generator conditioning causally related to gan performance? In *International conference on machine learning*, pp. 3849–3858. PMLR, 2018.

Physical Intelligence. Openpi: Open-source infrastructure for vision-language-action models. https://github.com/Physical-Intelligence/openpi, 2024.

Salah Rifai. Contractive auto-encoders: Explicit invariance during feature extraction. 2011.

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

Junhyuk So, Chiwoong Lee, Shinyoung Lee, Jungseul Ok, and Eunhyeok Park. Improving generative behavior cloning via self-guidance and adaptive chunking. *Advances in Neural Information Processing Systems*, 38:84432–84456, 2026.

Bowen Song, Zecheng Zhang, Zhaoxu Luo, Jason Hu, Wei Yuan, Jing Jia, Zhengxu Tang, Guanyang Wang, and Liyue Shen. Ccs: Controllable and constrained sampling with diffusion models via initial noise perturbation. *Advances in Neural Information Processing Systems*, 38:70516–70552, 2026.

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. Specprune-vla: Accelerating vision-language-action models via action-aware self-speculative pruning. *arXiv preprint arXiv:2509.05614*, 2025.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. Vla knows its limits: Adaptive execution horizons for robot policies. *arXiv preprint arXiv:2602.21445*, 2026.

Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators, 2023.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *Advances in Neural Information Processing Systems*, 38:164448–164473, 2026.

Zebin Yang, Qi Wang, Yunhe Wang, Xiurui Guo, Bo Yu, Shaoshan Liu, Jiafeng Xu, Hao Dong, and Meng Li. Jetson-pi: Towards onboard real-time robot control via foresight-aligned asynchronous inference. *arXiv preprint arXiv:2607.12659*, 2026.

Jiyao Zhang, Zimu Han, Junhan Wang, Xionghao Wu, Shihong Lin, Jinzhou Li, Hongwei Fan, Ruihai Wu, Dongjiang Li, and Hao Dong. Hipolicy: Hierarchical multi-frequency action chunking for policy learning. *arXiv preprint arXiv:2604.06067*, 2026.

Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *arXiv preprint arXiv:2011.02398*, 2020.

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

Yuchi Zhao, Miroslav Bogdanovic, Arjun Sohal, Liyu Tao, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Dynamic execution horizon prediction for chunk-based robot policies. *arXiv preprint arXiv:2606.11408*, 2026.

Kaijun Zhou, Qiwei Chen, Da Peng, Zhiyang Li, Xijun Li, and Jinyu Gu. Characterizing vision-language-action models across xpus: Constraints and acceleration for on-robot deployment. *arXiv preprint arXiv:2604.24447*, 2026.

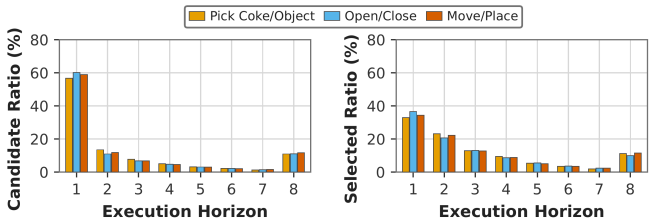


Figure 8: **Candidate and selected execution horizons in SimplerEnv ($N = 3$). Left:** Execution horizons assigned to all nominal candidates. **Right:** Execution horizons of the nominal chunks selected by PICES at the same policy calls.

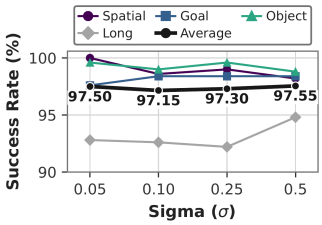


Figure 9: **Effect of the perturbation scale on LIBERO success rate.** We evaluate PICES at $\sigma \in \{0.05, 0.10, 0.25, 0.50\}$.

## A Horizon Distributions and Perturbation Sensitivity

**Distribution of execution horizons.** To examine how PICES selects action chunks and execution horizons, we analyze the policy call records collected during closed-loop evaluation of GR00T N1.7 on SimplerEnv–Fractal with three $N = 3$ nominal candidates. We group tasks by manipulation objective into three categories: Pick Coke/Object (Pick Coke Can and Pick Object), Open/Close (Open Drawer and Close Drawer), and Move/Place (Move Near and Place in Closed Drawer).

Figure 8 compares the execution horizons assigned to all nominal candidates with those of the chunks selected by PICES at the same recorded policy calls. For this analysis, we apply the horizon-estimation rule to all three nominal candidates, including those excluded by the initial-sensitivity filter. Each call contributes three candidate horizons (left) and one selected horizon (right). Across all groups, the selected distribution exhibits a lower proportion of one-step horizons and a higher proportion of intermediate horizons, particularly two to four steps. The proportion of one-step horizons decreases from $59.07\%$ among nominal candidates to $34.93\%$ among PICES selections. The corresponding average horizon increases from $2.49$ to $3.01$ actions. The consistent shift across the three groups shows that candidate selection increases the execution length on average across different manipulation objectives. The resulting distribution includes both short and long horizons, reflecting that PICES makes candidate-specific decisions during rollout.

**Effect of the perturbation scale.** We examine the sensitivity of PICES's task performance to the perturbation scale $\sigma$ on LIBERO. With $N = 3$, we evaluate $\sigma \in \{0.05, 0.10, 0.25, 0.50\}$ across the four task suites. As shown in Figure 9, the average success rate across suites ranges from $97.15\%$ to $97.55\%$, a difference of $0.40$ percentage points. Although individual suites exhibit variation, the average success rate changes little across the sigmas.

# B GPU INFERENCE COST ANALYSIS

**Operation Characteristics of Action Expert.** Realizing benefits of multi-action chunks requires generating multiple candidate chunks without a proportional increase in inference latency in GPUs. At the single batch, the arithmetic intensities (AI) of the $\pi_{0.5}$, GR00T N1.5, and GR00T N1.7 action experts are 59, 63, and 43 FLOP/byte, respectively. When AI remains smaller than ridge-point of each GPU, inference remains on the memory-bound condition. The single-batch AI of VLA models lies in the memory-bound region of the roofline models for Jetson Orin and Thor, indicating that single-candidate generation does not fully utilize their compute throughput.

However, for the mulit-action chunk generation, $B$ initial-noise tensors are stacked along the batch dimension and processed by the action expert at each flow-matching step, avoiding repeated execution of the full VLA inference pipeline. At a batch size of 6, the AI of $\pi_{0.5}$, GR00T N1.5, and GR00T N1.7 increases to 205.6, 190.6, and 182.4 FLOP/byte, respectively. Increasing the batch size to 8 further raises these values to 239.9, 222.1, and 223.0 FLOP/byte, exceeding the ridge point 207.5 FLOP/byte of Jetson Orin. Once the AI exceeds the ridge point, the operation turns into a compute-bound characteristic, where additional computation increases latency.

Accordingly, for Jetson Orin, all three models approach the ridge point at a batch size of 6, without increasing latency due to additional computation. For Jetson Thor, the operations remain memory-bound even at a batch size of 20, indicating that the available compute throughput is still not fully utilized. However, beyond a batch size of 10, the growing cost of loading activations, in addition to the cost of loading action expert weights, leads to a smooth increase in latency. Considering both hardware constraints and operation AI, these observations support setting the maximum batch size to 6 for Jetson Orin and 10 for Jetson Thor.

## B.1 INFERENCE LATENCY AND BATCHING OVERHEAD

Table 6: **GPU latency across batch sizes on Jetson Orin and Jetson Thor.** We report the total VLA inference latency (ms) for batch sizes $B \in \{1, 5, 10, 15, 20\}$.

| **Model** | **Jetson Orin** | | | | | **Jetson Thor** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $B=1$ | $B=5$ | $B=10$ | $B=15$ | $B=20$ | $B=1$ | $B=5$ | $B=10$ | $B=15$ | $B=20$ |
| $\pi_{0.5}$ | 310.4 | 361.6 | 464.1 | 581.1 | 676.6 | 127.2 | 149.8 | 179.3 | 215.3 | 267.0 |
| GR00T N1.5 | 149.3 | 162.1 | 192.2 | 230.8 | 273.4 | 62.8 | 67.2 | 73.7 | 82.7 | 90.8 |
| GR00T N1.7 | 209.9 | 235.4 | 301.6 | 361.6 | 401.1 | 101.1 | 111.5 | 124.5 | 148.3 | 166.2 |

Table 7: **Additional latency per batch across batch sizes on Jetson Orin and Jetson Thor.** We report per batch additional latency (ms) for batch sizes $B \in \{1, 5, 10, 15, 20\}$.

| **Model** | **Jetson Orin** | | | | | **Jetson Thor** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $B=1$ | $B=5$ | $B=10$ | $B=15$ | $B=20$ | $B=1$ | $B=5$ | $B=10$ | $B=15$ | $B=20$ |
| $\pi_{0.5}$ | 9.03 | 10.25 | 15.37 | 18.05 | 18.31 | 4.90 | 4.53 | 5.21 | 5.87 | 6.99 |
| GR00T N1.5 | 1.08 | 2.56 | 4.29 | 5.43 | 6.20 | 0.96 | 0.87 | 1.08 | 1.32 | 1.40 |
| GR00T N1.7 | 2.91 | 5.10 | 9.17 | 10.11 | 9.56 | 2.74 | 2.08 | 2.34 | 3.14 | 3.25 |

**Total inference latency and additional latency per batch.** Table 6 reports measured total inference latency. On Jetson Orin, increasing the batch size $B = 1$ to 5 results in latency increases of 3.5%, 8.6%, 12.1% for $\pi_{0.5}$, GR00T N1.5, and GR00T N1.7, respectively. Batch size between $B = 10$ to 20, total latency increases by 10.9%, 42.2%, and 33.0% relative to $B = 10$, respectively. Table 7 reports additional latency per batch, to quantify the cost of adding single action chunk. Specifically, increasing the baseline batch size from $B = 5$ to $B = 10$ raises the incremental latency from 10.25 to 15.37ms for $\pi_{0.5}$, from 2.56 to 4.29 ms for GR00T N1.5, and from 5.10 to 9.17ms for GR00T N1.7. Together with the roofline analysis, the measurements provide analytical and empirical support for the chosen generation budget.

### B.2 GPU PROFILING SETUP

Table 8: GPU environments and evaluation settings.

| Setting | Jetson AGX Orin | Jetson AGX Thor |
|---|---|---|
| Jetson Linux | R36.5.0 | R38.4.0 |
| CUDA | 12.6 | 13.0 |
| PyTorch | 2.8.0 | 2.13.0+cu130 |
| TensorRT | 10.3.0.30 | 10.13.3.9 |
| Power mode | MAXN | MAXN |
| Clock settings | Fixed GPU: 1300.5 MHz<br>Fixed EMC: 3199 MHz | Fixed GPU: 1575 MHz<br>Fixed EMC: 4266 MHz |
| Warm-up calls ($\pi_{0.5}$/GR00T) | 5/10 | 5/10 |

**Environment setup.** VLA inference on edge device can become a latency bottleneck for responsive robot control, motivating inference acceleration. We therefore evaluate NVIDIA Jetson AGX Orin and Jetson AGX Thor as representative edge deployment platforms. Software environments and operating settings are listed in Table 8.

**Inference Configurations** All models use `bfloat16` without quantization. For each model-device pair, PICES and all baselines, including AAC and $A^3$, are evaluated on the same optimized model implementation. The $\pi_{0.5}$ action expert uses device-tuned Triton kernels with CUDA graphs. GR00T System 1 uses the TensorRT deployment path provided by the official GR00T repository, while System 2 uses eager PyTorch with SDPA. We use 10 flow-matching steps for $\pi_{0.5}$ and 4 for both GR00T models. We evaluate $\pi_{0.5}$/GR00T inference latency after a 90 s burn-in and 5/10 warm-up calls. Timing covers backbone inference and action generation from GPU-resident inputs, excluding input reprocessing, observation host-to-device transfer, environment interaction, and one-time initialization. PICES latency additionally includes probe construction and associated transfers, sensitivity computation, and candidate selection. Each PICES call generates $N$ nominal and $N$ perturbed chunks, with conditioning features computed once per call.

## C Open-loop Evaluation of Reference-Action Agreement

**Evaluation contexts and action chunks.** We evaluate open-loop action prediction error using the GR00T N1.7 SimplerEnv–Fractal checkpoint on the Fractal/RT-1 dataset. Concretely, we uniformly sample 1,000 contexts without replacement from valid windows containing eight consecutive actions within a single episode. Each context consists of the current image, robot state, and language instruction, together with the corresponding recorded action sequence. The recorded actions serve as the reference action chunk for evaluation of one to eight action horizon.

**Candidate sampling.** We sample candidate chunks under identical conditioning inputs by drawing independent Gaussian initializations for each context. For PICES and the lowest-$S_n$ variant, we generate one auxiliary chunk per candidate by adding Gaussian noise of scale $\sigma = 0.25$ to its initial noise. These auxiliary chunks provide sensitivity estimates, while the action execution uses nominal chunks.

**Root-Mean-Square-Error to measure action error.** Let $\hat{\boldsymbol{a}}_k$ and $\boldsymbol{a}_k$ denote the normalized prediction and reference vectors at action offset $k$. For a prefix of length $h$, the error is

$$\mathrm{RMSE}(h) = \sqrt{\frac{1}{dh} \sum_{k=0}^{h-1} \|\hat{\boldsymbol{a}}_k - \boldsymbol{a}_k\|^2} \tag{7}$$

where $d$ is the action dimension. All action components and evaluated timesteps receive equal weight. We report means of the per-evaluation RMSEs.

**Oracle selected action chunk.** The motivational study varies the number of nominal candidates from one to ten and evaluates fixed horizons of one to eight execution horizon. For each context and candidate set, the oracle selects the candidate with the lowest RMSE at the specified horizon. We then average these minimum errors to obtain the curves in Figure 2. Selection is performed separately at each horizon, so the chosen candidate may change along a curve. The oracle uses reference actions only for this analysis, and the dashed line marks the single-candidate, one-step baseline.

**Fixed and adaptive horizon methods evaluation.** The fixed-step baseline evaluates a single nominal chunk at a prescribed horizon. PICES follows the selection procedure in Section 3.4, while the lowest-$S_n$ variant selects the candidate with the lowest initial sensitivity and applies the same horizon-estimation rule. Both variants use an initial sensitivity window of one step. For each adaptive method, we compute prefix RMSE at the horizon selected for each context, then average the errors and selected horizons separately. Figure 5 reports the resulting mean RMSE against the mean selected horizon. We refer to Appendix E for the AAC and $A^3$ configurations.

## D Interpretation of Stepwise Action Sensitivity

**Setup.** We fix the conditioning input $\boldsymbol{c}$ and model parameters. Conditional on these fixed quantities, we sample $\boldsymbol{\epsilon}^{(n)}$ with independent standard Gaussian entries. Let $D = dH$ denote the dimension of the vectorized initial-noise matrix, and define

$$\rho^{(n)} = \|\boldsymbol{\epsilon}^{(n)}\|_F, \qquad \boldsymbol{u}^{(n)} = \frac{\mathrm{vec}(\boldsymbol{\epsilon}^{(n)})}{\rho^{(n)}}, \qquad \delta^{(n)} = \sigma\rho^{(n)}. \tag{8}$$

Here, $\boldsymbol{u}^{(n)} \in \mathbb{R}^D$ is the unit perturbation direction, and $\delta^{(n)}$ is the perturbation magnitude. Let $\boldsymbol{J}_k^{(n)} \in \mathbb{R}^{d\times D}$ be the Jacobian of the action at offset $k$ with respect to the vectorized initial noise, evaluated at $\boldsymbol{Z}^{(n)}$. This Jacobian differentiates the complete implemented action generator with respect to all initial-noise entries, including those associated with other action offsets. The index $k$ denotes a future action offset.

**Local directional interpretation.** Normalizing output differences by input displacement provides a finite-difference assessment of generator sensitivity, as used in Jacobian Clamping (Odena et al., 2018). Assume that the action-output map is differentiable at the nominal initialization. For a fixed sampled probe, a first-order expansion as $\sigma \to 0$ gives

$$\widetilde{\boldsymbol{a}}_k^{(n)} - \boldsymbol{a}_k^{(n)} = \delta^{(n)}\boldsymbol{J}_k^{(n)}\boldsymbol{u}^{(n)} + o\Big(\delta^{(n)}\Big). \tag{9}$$

Such Taylor expansions are standard in analyses of small input perturbations (Bishop, 1995). Substituting into Eq. equation 3 yields

$$\begin{aligned} G_k^{(n)} &= \frac{\|\widetilde{\boldsymbol{a}}_k^{(n)} - \boldsymbol{a}_k^{(n)}\|^2}{(\delta^{(n)})^2} \\ &= \|\boldsymbol{J}_k^{(n)}\boldsymbol{u}^{(n)}\|^2 + o(1). \end{aligned} \tag{10}$$

Thus, for sufficiently small realized perturbations, the score approximates the squared norm of the directional derivative of the action at offset $k$.

**Interpretation at finite perturbation magnitude.** For the finite perturbation used in generation, $G_k^{(n)}$ measures the normalized squared action change along the sampled displacement. Its interpretation in terms of a directional derivative applies when the first-order approximation at the realized magnitude $\delta^{(n)} = \sigma\|\boldsymbol{\epsilon}^{(n)}\|_F$. The nonlinear remainder can affect the score at finite $\delta^{(n)}$, and normalization does not remove this effect. All action offsets of a candidate share the same perturbation, so the profile $\{G_k^{(n)}\}_{k=0}^{H-1}$ describes how the corresponding actions respond to that perturbation. These scores are computed from the paired outputs without explicitly constructing the Jacobian. This local interpretation does not provide a bound on action prediction error. The usefulness of the profile for joint action-chunk and execution-horizon selection is evaluated empirically in Section 3.5.

# E Detailed Experimental Setup and Benchmarks

**Models and implementation.** We evaluate PICES with three action-chunking VLA models: $\pi_{0.5}$, GR00T N1.5, and GR00T N1.7 (Intelligence et al., 2025; Bjorck et al., 2025). We use their official implementations and apply PICES at inference time without additional training. Specifically, we evaluate GR00T N1.7 on SimplerEnv–Fractal (Li et al., 2024b), $\pi_{0.5}$ and GR00T N1.5 on LIBERO (Liu et al., 2023). For each model–benchmark configuration, we retain the action dimensionality specified by the corresponding official repository and use its default execution horizon for the fixed-horizon baseline.

**Benchmarks and evaluation protocol.** We conduct simulation experiments on SimplerEnv and LIBERO. For SimplerEnv, we evaluate six manipulation tasks in the Fractal (Google Robot) setting, with 100 episodes per task. For LIBERO, we evaluate four task suites: LIBERO-Spatial, LIBERO-Goal, LIBERO-Object, and LIBERO-Long (LIBERO-10), with 10 tasks per suite and 50 evaluation episodes per task.

- $\pi_{0.5}$ **on LIBERO.** We use the released checkpoint and evaluation code provided by the official `openpi` repository, available at `https://github.com/Physical-Intelligence/openpi`.
- **GR00T N1.5 on LIBERO.** We use the released checkpoints for the four LIBERO suites and the evaluation scripts documented in the `n1.5-release` branch of the official `Isaac-GR00T` repository, available at `https://github.com/NVIDIA/Isaac-GR00T/tree/n1.5-release`.
- **GR00T N1.7 on SimplerEnv–Fractal.** We use the released Fractal checkpoint and evaluation scripts documented in the official `Isaac-GR00T` SimplerEnv example, available at `https://github.com/NVIDIA/Isaac-GR00T/blob/main`.

**Real-world setup** We evaluate our method across three manipulation tasks using a 6-DoF UR10e robot arm equipped with a Robotiq 2F-85 gripper: `Tape`, `Stapler`, and `Cube`. For each task, we collect 24 teleoperated demonstrations (72 in total) spanning varied initial object configurations. We evaluate each policy over 36 independent trials per task with randomized object placements. To assess both task effectiveness and computational efficiency, we report the Success Rate (%) and the average number of policy calls per episode. See Appendix F for the details.

**Baseline and PICES configurations.** We reproduce AAC, Autohorizon and $A^3$ using the code released in their respective repositories, retaining the action-sampling budgets specified by the released implementations. We additionally evaluate sample-matched variants that generate the same number of action samples per batch as PICES to align the GPU latency of batched action generation. These variants are denoted by AAC$^*$ and $A^{3*}$. The matched sampling budget includes all $N(1+1)$ nominal and perturbed action chunks generated by PICES.

Unless specified, PICES uses $N = 3$ nominal candidates and a perturbation scale of $\sigma = 0.25$. We use the default execution horizon specified by each model's official repository as the reference-window length $W$. We set the reference-window length $W$ to the default execution horizon specified in each model's official repository for the corresponding benchmark. For GR00T N1.5 on LIBERO, whose default execution horizon is one step, we use five action steps, matching the default execution horizon of $\pi_{0.5}$ on the same benchmark. This window defines the initial sensitivity reference for candidate assessment, while PICES adaptively selects the execution horizon $h$ between $W$ and the prediction horizon $H$.

# F Real-World Experimental Setup

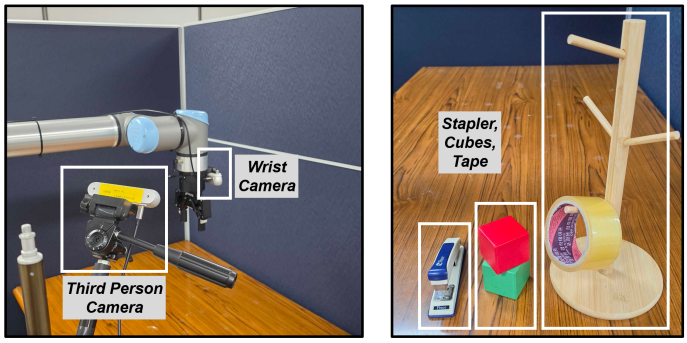


Figure 10: **Real-world experimental setup and task objects.** Left: The UR10e robot with wrist-mounted and third-person cameras. Right: A tape and stand, a stapler, red and green cubes used for Tape Hanging, Stapler Pressing, and Cube Stacking, respectively.

**Robot platform and control.** As shown in Figure 10, our real-world experimental platform comprises a 6-DoF UR10e arm equipped with a Robotiq 2F-85 gripper. Visual observations are captured using RGB cameras streaming at 30 Hz. The robot is operated in operational space at a control frequency of 30 Hz, with actions parameterized as 7-dimensional vectors $[\Delta x, \Delta y, \Delta z, \Delta\text{roll}, \Delta\text{pitch}, \Delta\text{yaw}, g_t] \in \mathbb{R}^7$ representing end-effector Cartesian displacements, Euler angle rotations, and continuous gripper commands.

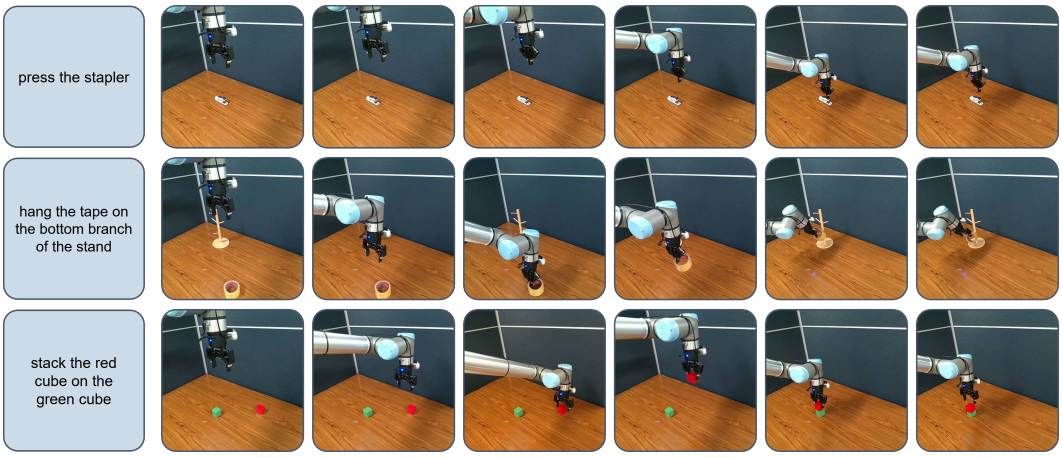


Figure 11: Task examples for real-world experiments

**Manipulation Tasks.** We consider three contact-rich and fine-grained manipulation tasks: *Tape Hanging*, *Stapler Pressing*, and *Cube Stacking*, which are abbreviated by their respective object names as `Tape`, `Stapler`, and `Cube` in the main text, as shown in Figure 11. *Tape Hanging* requires picking up a tape from the table and hanging it onto the bottom branch of a multi-branch stand. *Stapler Pressing* requires positioning the end-effector over a desktop stapler and applying downward force to depress the top lever and actuate the stapling mechanism. *Cube Stacking* requires grasping the red cube, precisely stacking it atop the green cube, and retracting the end-effector upward with the gripper open, such that the tower remains stable without toppling.

**Fine-tuning Configuration.** All demonstration trajectories are recorded at 30 Hz using a GELLO (Wu et al., 2023). For each task, we collect 24 teleoperated demonstrations (amounting to 72 trajectories in total) across diversified initial object configurations. We initialize our policy from the official $\pi_{0.5}$-base checkpoint provided by OpenPI (Physical Intelligence, 2024) and perform full-parameter fine-tuning on 4 $\times$ NVIDIA RTX PRO 6000 Blackwell GPUs. The model is

Table 9: Training hyperparameters and real-world experiment settings for fine-tuning $\pi_{0.5}$ on the UR10e platform.

| **Hyperparameter / Setting** | **Value** |
|---|---|
| Base checkpoint | $\pi_{0.5}$-base (OpenPI) |
| # Training GPUs | 4 $\times$ NVIDIA RTX PRO 6000 Blackwell |
| Total batch size | 128 (32 per GPU) |
| # Train steps | 5,000 steps ($\approx$ 17.7 epochs) |
| Fine-tuning mode | Full fine-tuning (all parameters) |
| Action chunk horizon | 16 steps |
| Control frequency | 30 Hz |
| Action representation | 7-DoF: $[\Delta x, \Delta y, \Delta z, \Delta\text{roll}, \Delta\text{pitch}, \Delta\text{yaw}, \text{gripper}]$ |
| Teleoperation interface | GELLO (collected at 30 Hz) |
| # Demonstrations | 24 per task (72 total across *Cube Stacking*, *Tape Hanging*, *Stapler Pressing*) |
| Evaluation protocol | 36 trials per task (108 rollouts total) under perturbed object poses |
| Evaluation metrics | Success Rate (%), Average policy calls |

trained for 5,000 steps with an effective batch size of 128 (32 per GPU), corresponding to approximately 17.7 training epochs, using an action chunk prediction horizon of $H = 16$.

**Evaluation Protocol.** Each policy is evaluated on 36 independent trials per task (108 rollouts total) with 12 distinct initial object placements. We report the task success rate (%) and the average number of policy calls per episode to measure execution reliability and inference efficiency. Detailed training and deployment configurations are summarized in Table 9.

## G Additional Related Work

**Action Chunking in VLA Models.** Action chunking policy predicts a sequence of future actions conditioned on current observation, improving temporal consistency. Recent VLAs such as CogACT, $\pi$, and GR00T series use transformer-based action experts, implemented as Diffusion Transformers (DiTs), to model multimodal distributions over continuous action chunks (Li et al., 2024a; Black et al., 2024; Bjorck et al., 2025). At inference time, such policies execute only a fixed-length horizon of each predicted chunk. Executing a long fixed-length actions reduces policy calls but limits the incorporation of new observations, motivating adaptive execution horizons (Liang et al., 2026; Wang et al., 2026; Chen et al., 2026; Zhang et al., 2026). MoH requires auxiliary module for multiple horizons and uses cross-horizon agreement to determine an executable actions (Jing et al., 2025). DEHP uses a lightweight predictor trained with online reinforcement learning to adapt the execution horizon of a frozen pretrained action-chunking policy (Zhao et al., 2026). AAC estimates a single execution horizon from cross-sample action entropy (Liang et al., 2026). AutoHorizon estimates an execution horizon for each predicted chunk using the self-attention patterns (Wang et al., 2026). $A^3$ selects a representative draft from sampled trajectories and determines execution horizon through an additional batched verification pass (Chen et al., 2026). In contrast, our method instead estimates a horizon for each candidate and uses the estimated execution length to jointly choose the trajectory and its execution extent in inferece time without training.

**Acceleration for vision-language-action models** Recent works on efficient VLA inference seek to reduce cumulative inference time using various methods. SmolVLA overlaps action prediction with execution and evaluates efficiency using task-completion time and the number of successful pick-and-place operations within a fixed time budget (Shukor et al., 2025). To reduce inference overhead, prior work exploits redundancy in visual inputs and model computation through token pruning, channel pruning, and model scheduling (Wang et al., 2025; Li et al., 2025; Liu et al., 2025b; Huang et al., 2025; 2026; Feng et al., 2026). FastV, originally developed for vision-language models, prunes visual tokens using attention scores from early layers to reduce computation in subsequent layers (Chen et al., 2024). VLA-Cache exploits temporal redundancy by reusing cached key-value representations of minimally changed visual tokens across adjacent observations (Xu et al., 2026). GridS reduces the computational cost of vision-language-action models by learning task-aware sampling coordinates and extracting pruned visual tokens via differentiable bilinear interpolation (Feng et al., 2026). Our method is orthogonal to token pruning, which reduces computation within individual policy calls. DaDu-Corki predicts future trajectories to reduce expensive model invocations (Huang et al., 2025). PICES examines how candidate-specific horizon selection can reduce policy calls, while explicitly accounting for the additional cost of generating and assessing multiple action candidates.

## H PICES ALGORITHM

**Algorithm 1** PICES: joint chunk and execution-horizon selection

**Input:** Conditioning input $\boldsymbol{c}$, generator $V_\theta$, and parameters $N, W, \sigma$
**Output:** Selected nominal chunk–horizon pair $(\boldsymbol{A}^{(n^\star)}, h^\star)$

*// Candidate generation and sensitivity assessment*
Independently sample $\boldsymbol{Z}^{(n)}, \boldsymbol{\epsilon}^{(n)} \in \mathbb{R}^{d\times H}$ with i.i.d. $\mathcal{N}(0,1)$ entries, $n = 1, \dots, N$
$\widetilde{\boldsymbol{Z}}^{(n)} \leftarrow \boldsymbol{Z}^{(n)} + \sigma\boldsymbol{\epsilon}^{(n)}$, *// Perturb initial noise*
Generate $\boldsymbol{A}^{(n)} = V_\theta(\boldsymbol{c}, \boldsymbol{Z}^{(n)})$ and $\widetilde{\boldsymbol{A}}^{(n)} = V_\theta(\boldsymbol{c}, \widetilde{\boldsymbol{Z}}^{(n)})$ for all $n$ in batched policy call
**for** $n = 1, \dots, N$ **do**
    Compute $G_k^{(n)}$ for $k = 0, \dots, H-1$ using equation 3 *// Stepwise sensitivity*
    $S_n \leftarrow W^{-1} \sum_{k=0}^{W-1} G_k^{(n)}$ *// Initial-window sensitivity*
**end for**

*// Candidate filtering and horizon selection*
$\mathcal{K} \leftarrow \{n \in \{1, \dots, N\} : S_n \le \mathrm{median}(S_1, \dots, S_N)\}$
**for** each $n \in \mathcal{K}$ **do**
    $h_n \leftarrow \min\big(\{k \in \{W, \dots, H-1\} : G_k^{(n)} > S_n\} \cup \{H\}\big)$ *// Execution horizon selection*
**end for**
$n^\star \in \arg\max_{n\in\mathcal{K}} h_n$, $h^\star \leftarrow h_{n^\star}$ *// Action chunk selection*
**Return:** $(\boldsymbol{A}^{(n^\star)}, h^\star)$

# I Detailed Simulation Results on LIBERO

Table 10: **LIBERO results with $\pi_{0.5}$ and GR00T N1.5.** We report per-suite and average success rates (%) and policy calls, together with single-call GPU latency and inference-time speedup on Jetson Orin and Thor. Fixed ($h = x$) executes $x$ actions per policy call; single-call latency is shared across the five fixed-horizon settings for each model and device. Speedups are reported relative to Fixed ($h = 5$).

| Method | Success rate (%) ↑ | | | | | Policy calls ↓ | | | | | Jetson Orin | | Jetson Thor | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Spatial | Goal | Object | Long | **Average** | Spatial | Goal | Object | Long | **Average** | Single call (ms) ↓ | Speedup ↑ | Single call (ms) ↓ | Speedup ↑ |
| **$\pi_{0.5}$** | | | | | | | | | | | | | | |
| Fixed ($h = 1$) | 99.0 | 92.8 | 91.4 | 84.4 | 91.9 | 102.8 | 110.6 | 145.7 | 272.9 | 158.0 | | 0.2 | | 0.2 |
| Fixed ($h = 2$) | 99.0 | 95.4 | 94.8 | 86.6 | 94.0 | 51.2 | 55.3 | 71.7 | 132.1 | 77.6 | | 0.4 | | 0.4 |
| Fixed ($h = 5$) | 98.0 | 97.6 | 98.2 | 92.0 | 96.5 | 20.6 | 21.8 | 27.7 | 50.6 | 30.2 | 297.2 | 1.0 | 126.7 | 1.0 |
| Fixed ($h = 8$) | **99.4** | 98.6 | 98.2 | 93.4 | 97.4 | 13.1 | 13.7 | 17.2 | 31.5 | 18.9 | | 1.6 | | 1.6 |
| Fixed ($h = 10$) | 98.2 | 98.2 | 97.6 | 93.8 | 97.0 | 10.5 | 11.1 | 13.7 | 25.6 | **15.2** | | **2.0** | | **2.0** |
| AAC | 97.6 | 94.2 | 98.8 | 94.8 | 96.4 | 15.5 | 15.6 | 18.8 | 32.5 | 20.6 | 359.2 | 1.2 | 152.4 | 1.2 |
| AAC* | 97.4 | 94.2 | 99.0 | 93.2 | 96.0 | 16.3 | 16.3 | 19.8 | 33.7 | 21.4 | 316.0 | 1.3 | 137.4 | 1.3 |
| AutoHorizon | 98.6 | 98.2 | 97.0 | **95.2** | 97.2 | 15.2 | 19.5 | 16.0 | 35.6 | 21.6 | 300.4 | 1.4 | 127.8 | 1.4 |
| $A^3$ | 99.0 | **99.0** | 97.0 | 93.8 | 97.2 | 10.6 | 11.2 | 13.9 | 25.5 | 15.3 | 455.8 | 1.3 | 199.7 | 1.3 |
| $A^{3*}$ | 98.8 | 97.4 | 97.6 | 90.4 | 96.0 | 19.7 | 20.8 | 26.9 | 49.3 | 29.2 | 421.6 | 0.7 | 187.3 | 0.7 |
| **PICES, $N = 3$** | 99.0 | 98.4 | **99.6** | 92.2 | 97.3 | 13.9 | 14.4 | 18.0 | 33.4 | 19.9 | 311.4 | 1.5 | 131.6 | 1.5 |
| **PICES**, $N = 3$** | 98.4 | 98.6 | 99.4 | 93.6 | **97.5** | 11.5 | 11.9 | 14.8 | 27.4 | 16.4 | 311.4 | 1.8 | 131.6 | 1.8 |
| **GR00T N1.5** | | | | | | | | | | | | | | |
| Fixed ($h = 1$) | 89.6 | 90.0 | 92.6 | 71.4 | 85.9 | 108.6 | 115.3 | 140.7 | 259.1 | 155.9 | | 0.2 | | 0.2 |
| Fixed ($h = 2$) | 91.4 | 91.0 | 93.8 | 78.0 | 88.6 | 53.7 | 56.7 | 69.6 | 129.0 | 77.2 | | 0.4 | | 0.4 |
| Fixed ($h = 5$) | 94.8 | 95.0 | 95.8 | 85.8 | 92.9 | 21.3 | 22.5 | 27.9 | 50.5 | 30.6 | 149.3 | 1.0 | 62.8 | 1.0 |
| Fixed ($h = 8$) | 95.2 | **95.8** | 95.6 | 86.4 | 93.3 | 13.5 | 14.3 | 17.5 | 34.2 | 19.9 | | 1.5 | | 1.6 |
| Fixed ($h = 16$) | 94.0 | 93.8 | 97.0 | 86.0 | 92.7 | 7.0 | 7.3 | 9.0 | 16.2 | **9.9** | | **3.1** | | **2.0** |
| AAC | 94.0 | 92.8 | 97.8 | 88.6 | 93.3 | 18.6 | 15.7 | 25.1 | 37.9 | 24.2 | 282.5 | 0.7 | 107.2 | 0.7 |
| AAC* | 94.0 | 92.0 | 96.6 | 88.6 | 92.8 | 19.7 | 17.0 | 26.5 | 39.9 | 25.8 | 185.9 | 0.9 | 77.7 | 0.9 |
| $A^3$ | 94.6 | 95.6 | **99.6** | **90.0** | **94.9** | 21.2 | 19.4 | 27.7 | 50.7 | 29.7 | 277.7 | 0.5 | 124.9 | 0.5 |
| $A^{3*}$ | 94.8 | 93.0 | 94.5 | 89.6 | 93.0 | 35.3 | 33.1 | 46.4 | 85.9 | 50.2 | 235.9 | 0.4 | 100.6 | 0.4 |
| **PICES, $N = 3$** | **97.4** | 95.6 | 95.6 | 89.4 | 94.5 | 15.7 | 15.5 | 19.6 | 36.3 | 21.4 | 178.5 | 1.2 | 74.8 | 1.2 |
| **PICES**, $N = 3$** | 95.2 | 94.8 | 97.6 | **90.0** | 94.4 | 10.8 | 11.1 | 14.0 | 25.8 | 15.4 | 178.5 | 1.6 | 74.8 | 1.6 |

*Same number of action samples as PICES.
**PICES with $W = 8$.