ACTION-CHUNK PRUNING: LEARNING AN EFFICIENT EXECUTION SCHEDULER FOR EMBODIED POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied policies trained from expert demonstrations have achieved strong performance in many tasks. However, since imitation learning fundamentally binds policy behavior to demonstration quality, slow or overly cautious movements in human demonstrations often lead the learned policy to adopt corresponding behavioral patterns, resulting in low execution efficiency. Existing approaches typically address this issue through dataset refinement followed by policy retraining or additional online interaction, both of which incur high computational and data-collection costs. In this work, we introduce **A**ction-**C**hunk **P**runing (**ACP**), a lightweight,offline framework that enables **plug-and-play policy acceleration**. ACP formulates policy acceleration as an action-chunk pruning problem, aiming to accelerate execution by replacing redundant action segments with feasible shortcuts while preserving the original motion. Specifically, we use a learned world model to evaluate candidate action shortcuts on offline demonstrations and construct a shortcut Directed Acyclic Graph (DAG), where Dynamic Programming (DP) computes optimal next-hop decisions starting from each state to minimize the remaining number of execution steps. These pruning decisions are then amortized into a lightweight scheduler that predicts a stride for each position in the action chunk in a single forward pass, avoiding repeated world-model evaluation and online optimization. We demonstrate ACP in simulation and on real-world tasks, validating its capability to accelerate execution while preserving the base policy's performance.

1 INTRODUCTION

Robotic learning has achieved strong performance across a wide range of manipulation tasks, with many methods learning control policies directly from offline expert demonstrations (Zeng et al., 2021; Shridhar et al., 2022; Mandlekar et al., 2022; Zhao et al., 2023; Chi et al., 2023). As datasets and model capacity have continued to scale, Vision-Language-Action (VLA) policies have substantially improved the generality of this paradigm across diverse tasks and environments (Open X-Embodiment Collaboration, 2024; Khazatsky et al., 2024; Kim et al., 2025; Black et al., 2025b;a). Many of these policies predict action chunks from demonstration trajectories and execute them at control rates inherited from the data collection process (Zhao et al., 2023; Chi et al., 2023; Pertsch et al., 2025; Black et al., 2025b). Consequently, they may also inherit slow behavioral patterns and redundant actions, leading to inefficient execution (Guo et al., 2025; Wu et al., 2026). This raises a natural question: **how can we accelerate the execution of action chunks while preserving policy performance?**

Prior work has accelerated robot execution through demonstration retiming, adaptive speed control, and deployment-time action subsampling (Shi et al., 2023; Guo et al., 2025; Ramirez Sanchez et al., 2026; Wu et al., 2026). However, several of these approaches require modified training data, additional policy optimization, or online interaction, increasing the cost of adapting large pretrained policies (Guo et al., 2025; Ramirez Sanchez et al., 2026; Nam et al., 2025). Recent work formulates execution acceleration as a constrained optimization problem and learns a plug-and-play scheduler for action-chunk subsampling using offline RL on a frozen policy (Wu et al., 2026). Its execution decision, however, applies a uniform rate to the entire predicted chunk, without explicitly accounting for redundancy that varies across positions within the chunk. This motivates an execution strategy that determines where actions can be skipped while retaining dense execution where needed.

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.

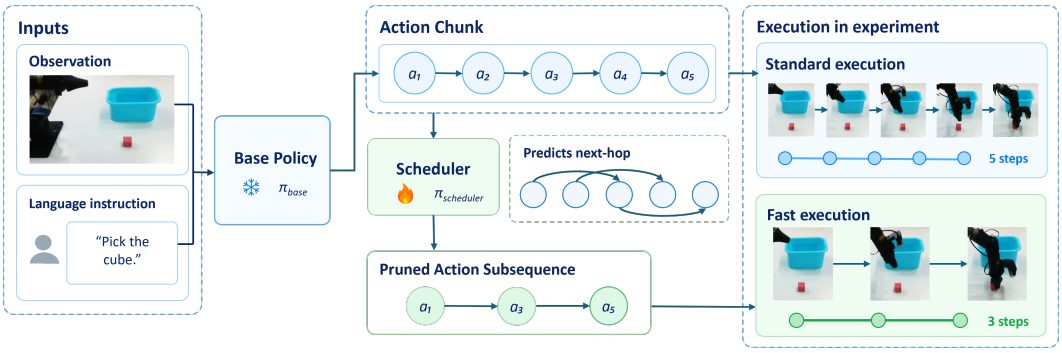


Figure 1: **Action-Chunk Pruning Scheduler.** ACP prunes redundant intermediate actions from an action chunk to accelerate execution while maintaining policy performance.

To address this gap, we introduce **A**ction-**C**hunk **P**runing (**ACP**), a lightweight framework that trains an external module offline to adaptively eliminate redundancy within action chunks. Specifically, we formulate policy acceleration as an action-chunk pruning problem, cast it as a shortest-path problem over a shortcut Directed Acyclic Graph (DAG) encoding feasible shortcuts, and amortize the resulting pruning decisions into a plug-and-play scheduler for real-time deployment, as illustrated in Figure 1. We first use a world model to evaluate candidate action shortcuts across offline demonstrations. Second, we construct a shortcut DAG where edges represent feasible action shortcuts, and use Dynamic Programming (DP) to find the minimum-hop feasible shortcut schedule from each state. Finally, we distill these pruning decisions into a lightweight scheduler that predicts next-hop strides for all chunk positions in a single forward pass. Since ACP is trained offline and operates externally to the base policy, it requires neither policy retraining nor online interaction while adding negligible latency during deployment.

This paper makes the following three main contributions: 1) We formulate policy acceleration as an action-chunk pruning problem, seeking a minimum-step execution schedule composed of feasible shortcuts while preserving the original motion. 2) We propose ACP, an offline framework that operationalizes this formulation as a plug-and-play acceleration module for frozen policies, enabling fine-grained action-chunk pruning without policy retraining or online interaction. 3) We evaluate ACP across simulation and real-world manipulation tasks alongside systematic ablations. Results demonstrate that ACP achieves overall speedups of 1.49× and 1.70× while preserving policy performance with only 0.73ms of additional inference latency.

2 Preliminaries

Action-chunk policies. Modern embodied policies commonly adopt action chunking, predicting a sequence of future actions at each decision step rather than a single next action (Zhao et al., 2023; Chi et al., 2023; Haldar et al., 2024; Black et al., 2025b;a). Given the current observation o_t, we denote a pretrained base policy by π_{base}, which predicts an action chunk of horizon H:

$$A_t = \left(a_t^1, a_t^2, \ldots, a_t^H\right).$$

Here, $a_t^i \in \mathcal{A}$ denotes the action at relative step i within the chunk, and H denotes the chunk horizon. In standard chunk execution, these actions are issued sequentially in their predicted sequential order. In this work, we treat π_{base} as frozen and abstract away its internal architecture, focusing on the predicted action chunks used for execution.

Problem setting. Given a predicted H-step action chunk $A_t = (a_t^1, \ldots, a_t^H)$, ACP replaces contiguous action segments with shortcut actions, producing a shorter execution sequence

$$\tilde{A}_t = (\tilde{a}_t^1, \ldots, \tilde{a}_t^M). \tag{1}$$

Here, $M \leq H$ denotes the number of shortcut actions executed after pruning. Each shortcut action $\tilde{a}_t^m$ is obtained by applying a merge operator $\mathcal{M}(\cdot)$ to one contiguous segment of the original action

chunk. For absolute actions, $\mathcal{M}$ retains the final action of the segment; for delta actions, it sums the translational increments, composes the rotational increments on $\mathrm{SO}(3)$, and retains the final gripper command of the segment. We define the stride k of a shortcut as the number of original control steps it spans, while each shortcut execution constitutes one hop in the execution graph.

3 METHOD

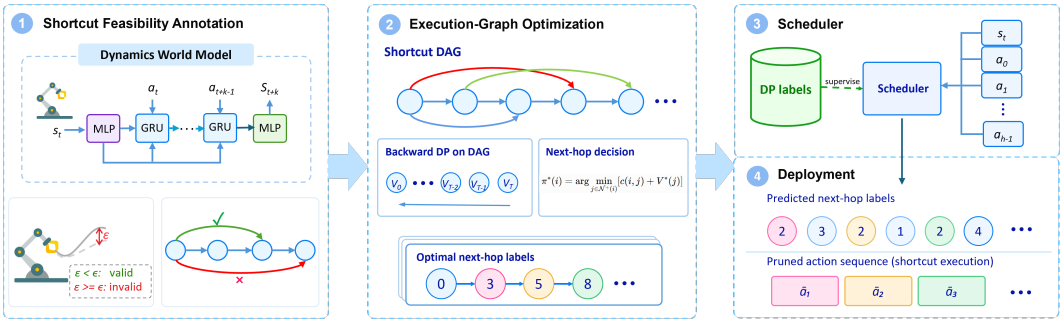


Figure 2: **Overview of ACP.** ACP first evaluates candidate action shortcuts on demonstrations and organizes feasible transitions into a DAG, where dynamic programming identifies the preferred next hop at each state. These offline decisions are distilled into a lightweight scheduler that predicts apointer table from s_t and A_t. At inference, ACP follows these pointers to assemble and execute a shortened action sequence, replanning when stride exceeds chunk horizon.

ACP separates structured schedule optimization from real-time deployment. Offline, a learned world model implemented as an Any-Step Dynamics Model (ADM) evaluates candidate action shortcuts along demonstration trajectories to construct a shortcut DAG, on which DP computes the minimum execution cost and corresponding next-hop decisions. The resulting next-hop labels are then used to supervise an amortized lightweight scheduler. At deployment, given an action chunk from the frozen base policy, the scheduler predicts a next-hop score matrix in a single forward pass, from which ACP decodes the pruned action sequence without invoking the ADM or solving online optimization problems. We provide the overview of ACP in Figure 2.

3.1 OFFLINE EXECUTION-GRAPH OPTIMIZATION

World-model as a feasibility evaluator. We train an Any-Step Dynamics Model $M_\theta(s_t, a)$ as an offline evaluator of candidate shortcuts (Chua et al., 2018; Janner et al., 2019; Hafner et al., 2023; Hansen et al., 2024). Given a robot state and a variable-length action sequence, the ADM predicts the corresponding future robot state. For a trajectory tuple $(s_t, A_t, s_{t+1:t+L}) \sim \mathcal{D}$, the ADM is trained via multi-step state prediction:

$$\mathcal{L}_{\mathrm{ADM}}(\theta) = \mathbb{E}_{\mathcal{D}}\left[\sum_{\ell=1}^{L} \|\hat{s}_{t+\ell} - s_{t+\ell}\|_2^2\right]. \tag{2}$$

For a trajectory state s_i and candidate stride $k \in \{1, \ldots, S\}$, we evaluate shortcut feasibility by comparing the unpruned reference trajectory against the accelerated shortcut motion. Querying M_θ with base action prefixes yields a reference rollout $\hat{s}^{\mathrm{ref}}_{i+r} = M_\theta(s_i, a_{i:i+r-1})$ for $r \in \{1, \ldots, k\}$. For the shortcut path, executing the merged action $\tilde{a}_{i,k} = \mathcal{M}(a_{i:i+k-1})$ produces the shortcut endpoint prediction $\hat{s}^{\mathrm{sc}}_{i,k} = M_\theta(s_i, \tilde{a}_{i,k})$, which is temporally interpolated across k steps to construct a length-matched shortcut trajectory $\tilde{s}^{\mathrm{sc}}_{i+1:i+k}$. We then measure the worst-case End-Effector (EEF) pose deviation over the entire horizon:

$$e_{i,k} = \max_{1 \le r \le k} d\big(\mathrm{EEF}(\hat{s}^{\mathrm{ref}}_{i+r}), \mathrm{EEF}(\tilde{s}^{\mathrm{sc}}_{i+r})\big), \tag{3}$$

where $\mathrm{EEF}(\cdot)$ extracts the robot end-effector pose, and $d(\cdot, \cdot)$ denotes the EEF pose distance metric measuring joint translational and rotational discrepancies (detailed definition provided in Appendix C.1).

A candidate shortcut is accepted only if its deviation satisfies $e_{i,k} \leq \epsilon$. Evaluating the maximum deviation over the rollout—rather than the average distance or endpoint error—provides a conservative feasibility criterion, strictly rejecting shortcuts that match the final pose but deviate substantially at intermediate states. Collecting all feasible edges across demonstrations constructs the execution DAG $G = (V, E)$.

Global Pruning via Dynamic Programming. Local shortcut feasibility does not determine the globally optimal pruning schedule, since a large feasible stride may reduce downstream shortcut opportunities. We therefore optimize shortcut selection over the DAG using DP. For each trajectory node i, the optimal cumulative cost $V^*(i)$ and the optimal pruning policy $\pi^*(i)$ can be computed exactly via backward dynamic programming (DP) starting from the end of the trajectory:

$$V^*(i) = \min_{j \in \mathcal{N}^+(i)} [c(i,j) + V^*(j)] \tag{4}$$

$$\pi^*(i) = \arg\min_{j \in \mathcal{N}^+(i)} [c(i,j) + V^*(j)] \tag{5}$$

where $\mathcal{N}^+(i)$ denotes the set of feasible successor nodes of node i. In this work, we uniformly set the edge cost to $c(i,j) = 1$, which makes minimizing the total cost equivalent to minimizing the number of execution hops on the graph.

3.2 Amortized Scheduler

Optimal Next-Hop Labels. Dynamic programming provides the long-horizon optimal cost of every feasible outgoing shortcut. We convert these values into a unique next-hop label for each demonstration state. Specifically, we select the optimal stride $k_i^\star$ from feasible strides $\mathcal{F}_i$ by lexicographically minimizing the optimal cost-to-go, shortcut deviation, and negative stride length:

$$k_i^\star = \arg\min_{k \in \mathcal{F}i} \left(c(i, i+k) + V^*(i+k), e_{i,k}, -k\right), \tag{6}$$

where the tuple minimization is performed in lexicographical order. This produces one deterministic next-hop label at every valid demonstration state. For a training chunk starting at state s_t, the labels at its successive positions provide supervision for all corresponding rows of the scheduler output.

Amortized Scheduler Learning. We train a lightweight scheduler f_ϕ to predict the DP-derived next-hop decisions directly from the current robot state and the action chunk generated by the frozen base policy. Given robot state s_t and $A_t = \pi_{\text{base}}(o_t)$, the scheduler outputs

$$Z = f_\phi(s_t, A_t) \in \mathbb{R}^{H \times S}, \tag{7}$$

where row i contains the logits for the candidate strides at position i of the action chunk, and S denotes the maximum shortcut stride. For each supervised position, the scheduler is trained to classify the optimal stride $k_i^\star$ using cross-entropy:

$$\mathcal{L}_{\text{sched}} = -\frac{1}{|\mathcal{I}_{\text{valid}}|} \sum_{i \in \mathcal{I}_{\text{valid}}} \log \frac{\exp(z_{i,k_i^\star})}{\sum_{k=1}^{S} \exp(z_{i,k})}, \tag{8}$$

where $\mathcal{I}_{\text{valid}}$ denotes the supervised chunk positions. Each position is classified over the full set of candidate strides $\{1, \ldots, S\}$. Shortcut feasibility is used offline to construct the DP labels, it is not required at deployment.

Next-Hop Schedule Decoding and Execution. At deployment, the frozen base policy first predicts an action chunk $A_t = \pi_{\text{base}}(o_t)$, and the scheduler produces the complete score matrix $Z = f_\phi(s_t, A_t)$ in a single forward pass. Each chunk position is assigned a predicted stride

$$\hat{k}_i = \arg\max_{k \in \{1,\ldots,S\}} z_{i,k}. \tag{9}$$

Starting from $i_0 = 0$, ACP follows these predictions recursively. If $i_m + \hat{k}_{i_m} \leq H$, the hop is accepted and $i_{m+1} = i_m + \hat{k}_{i_m}$; otherwise, decoding terminates. The accepted hops are merged into consecutive segments and executed sequentially. After the accepted shortcuts are executed, ACP queries the base policy from the resulting observation to obtain a new action chunk and scheduler prediction. All next-hop decisions within the current chunk are read from the same matrix Z, requiring no additional scheduler forward passes during decoding.

4 EXPERIMENTS

In this section, we evaluate our framework in simulation (Section 4.1) and on real-world manipulation tasks (Section 4.2). We then visualize how the predictive partition adapts its stride along a trajectory (Section 4.3), and study the key design choices of ACP (Section 4.4). The simulation and real-world evaluation settings are illustrated in Figure 3.

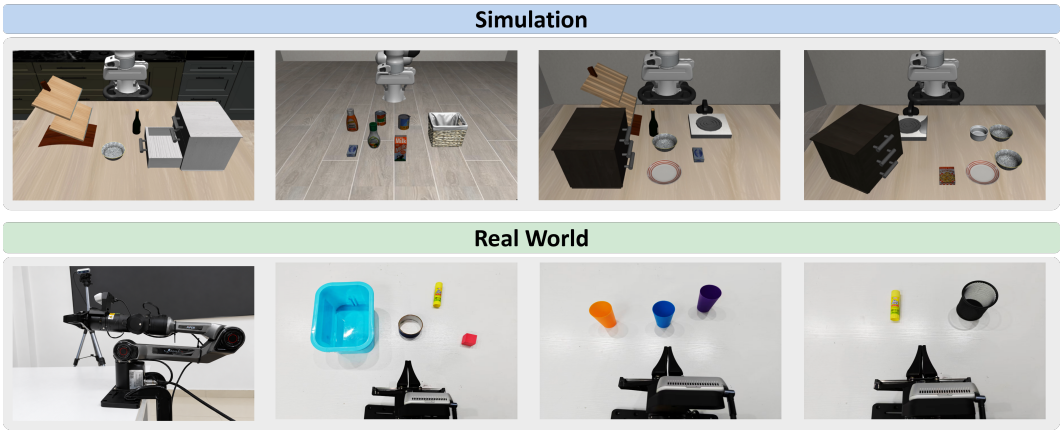


Figure 3: **Simulation and real-world evaluation environments.** We evaluate ACP on all four LIBERO suites in simulation and on three manipulation tasks in the real world. These settings provide complementary evaluation across diverse manipulation scenarios and demonstrate the effectiveness of ACP in both simulated and real-world environments.

4.1 SIMULATION BENCHMARK

We first evaluate ACP on simulation benchmarks to examine whether it can accelerate policy execution while preserving task performance. We conduct experiments on all four task suites of LIBERO, a benchmark designed for multi-task and lifelong robot learning (Liu et al., 2023). In all experiments, ACP is applied to frozen pretrained policies without modifying their parameters.

We compare ACP against three baselines: (1) the frozen base policy without execution acceleration, (2) Vanilla Downsampling(-ds*), which applies a fixed downsampling rate to every action chunk, and (3) SuP, which adaptively selects the downsampling rate of each action chunk based on world-model-estimated trajectory deviation (Wu et al., 2026). We report task success rate, episode length of successful rollouts, execution speedup ratio, and online scheduler inference latency.

Table 1: **Performance on LIBERO**. We report success rate and average successful control steps, across four LIBERO suites, alongside the average success rate and execution speedup. For each metric column, the best and second-best values are highlighted in bold.

Base Policy	Method	Spatial	Long	Goal	Object	Average
$\pi_{0.5}$	-base	**0.988**, 105.3	0.924, 267.9	0.980, 113.1	0.982, 138.1	0.969, 1.00×
	-ds2	0.914, **67.9**	0.874, **153.4**	0.952, **67.6**	0.970, **75.0**	0.928, **1.72**×
	+SuP	**0.972**, **70.4**	**0.940**, 215.2	**0.986**, 93.4	**0.994**, 83.0	**0.973**, 1.35×
	+ACP (Ours)	0.968, 74.9	**0.942**, **179.5**	**0.986**, **84.4**	**0.986**, **80.5**	**0.971**, **1.49**×

As shown in Table 1, ACP accelerates the frozen $\pi_{0.5}$ policy without sacrificing its task performance, achieving $1.49\times$ execution speedup while maintaining essentially the same average success rate as the base policy (0.971 vs. 0.969). In contrast, ds2 downsampling attains a higher $1.72\times$ speedup but reduces the average success rate to 0.928. This trade-off highlights a key limitation of uniform execution-rate reduction: different parts of an action chunk tolerate different degrees of pruning, and applying the same stride throughout the chunk can remove actions required near precision-critical interactions. ACP instead adjusts the shortcut stride locally, retaining dense execution where necessary while skipping more aggressively in redundant regions.

Compared with SuP, ACP achieves comparable task success (0.971 vs. 0.973) while improving the overall execution speedup from $1.35\times$ to $1.49\times$. The improvement is observed on three of the four suites, with the largest reduction in successful trajectory length occurring on LIBERO-Long. Unlike chunk-level rate selection, which applies a single downsampling pattern to an entire action chunk, ACP can compose shortcuts of different lengths within the same chunk. The results therefore suggest that explicitly modeling non-uniform redundancy inside action chunks provides additional room for acceleration while preserving the reliability of the underlying policy.

Table 2: Inference latency of ACP in LIBERO.

Component	Train. Params	Latency
VLM	3B	70.6 ms
Action Expert	300 M	33.3 ms
Scheduler (Ours)	**0.27** M	**0.73** ms

We further analyze the inference latency introduced by ACP on LIBERO with $\pi_{0.5}$, evaluated on a single NVIDIA RTX 4090 GPU. As shown in Table 2, the VLM contains approximately 3B parameters and requires 70.6 ms per inference, while the 300M-parameter action expert requires 33.3 ms. Due to the flow-matching formulation of $\pi_{0.5}$, the reported action-expert latency includes the 10 forward passes used to generate each action chunk. In comparison, our scheduler contains only 0.27M parameters and requires 0.73 ms for a single forward pass that predicts the complete execution schedule. This corresponds to less than 1% additional latency relative to the base-policy inference cost, indicating that ACP introduces negligible computational overhead while reducing the number of executed control steps.

4.2 Real-World Performance

We evaluate ACP on an AgileX PiPER robotic arm across three manipulation tasks: **Clean Table**, where the robot places different objects from the tabletop into a box; **Marker Insertion**, where it places a marker into a pen holder; and **Stack Cups**, where it sequentially grasps and stacks cups. These tasks span free-space transport, precise object placement, and sequential contact-rich manipulation. For each task, we collect 50 trajectories and fine-tune $\pi_{0.5}$ policy using LoRA ($r = 32$), ensuring a consistent base policy across all evaluations. All experiments are conducted on a single NVIDIA RTX 4090 GPU. We report task success and the number of control steps for the same fine-tuned policy with and without ACP.

Table 3: **Real-world performance** We report task success and average successful control steps across three real-world manipulation tasks, alongside the average success rate and execution speedup. For each metric column, the best and second-best values are highlighted in bold.

Base Policy	Method	Clean Table	Stack Cups	Marker Insertion	Average
$\pi_{0.5}$	-base	27/30, 979.2	**20/30**, 628.3	**16/30**, 527.9	**70.0%**, 1.00×
	+ds2	26/30, **594.4**	**19/30**, **392.5**	**17/30**, 332.4	**68.9%**, **1.62**×
	+Sup	26/30, 620.4	18/30, 431.5	**17/30**, **283.1**	67.8%, 1.60×
	+ACP (Ours)	**28/30**, **575.3**	**19/30**, **410.7**	15/30, **270.8**	**68.9%**, **1.70**×

Across the three real-world tasks, ACP achieves the highest overall execution speedup of $1.70\times$ while maintaining an average success rate of 68.9%, close to the 70.0% of the unmodified base policy. This behavior remains consistent across manipulation regimes involving long-range transport, precise terminal placement, and repeated grasping and contact transitions. Compared with fixed stride-2, ACP preserves the same average success while improving the speedup from $1.62\times$ to $1.70\times$; compared with SuP, it improves both average success (68.9% vs. 67.8%) and execution speedup ($1.70\times$ vs. $1.60\times$). These results suggest that fine-grained, within-chunk stride adaptation provides a more effective execution trade-off than either uniform downsampling or chunk-level rate selection on real hardware.

4.3 CASE STUDY

To examine how ACP adapts execution strides within a trajectory, we visualize the predicted strides over a representative manipulation episode from the LIBERO-LONG suite in Figure. 4. ACP exhibits a clear phase-dependent schedule: it uses larger strides during relatively smooth motions such as approaching and transporting the object, while reducing the stride to one around precision-critical interactions such as grasping and placement. The scheduler also varies the stride within individual phases rather than assigning a uniform execution rate, illustrating the non-uniform redundancy that motivates action-chunk pruning.

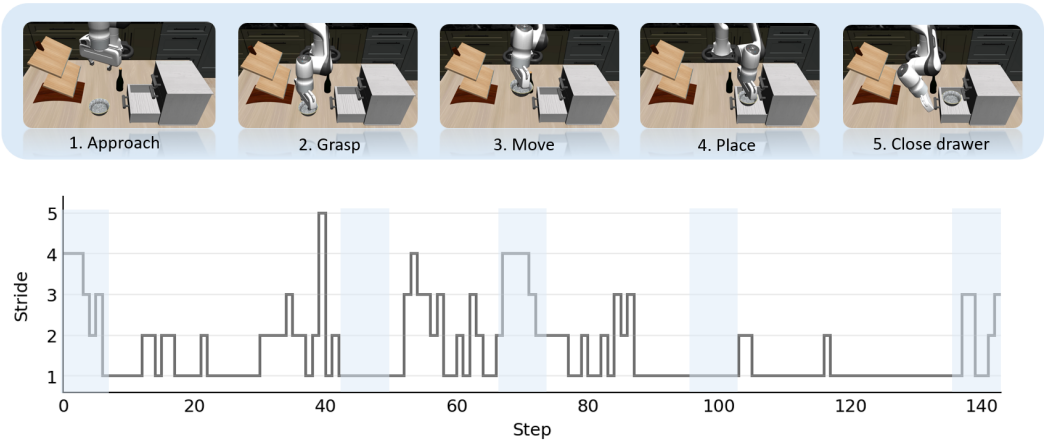


Figure 4: **Case Study.** Predicted shortcut strides for the task "*put the black bowl in the bottom drawer of the cabinet and close it.*" Shaded regions correspond to the manipulation phases shown above. ACP uses smaller strides near precision-critical interactions and larger strides during smoother motions.

4.4 ABLATION STUDIES

Impact of Amortized Scheduler. We first compare Greedy, Online Pruning, and ACP to evaluate different shortcut scheduling strategies. Greedy selects the longest admissible stride at each node, while Online Pruning predicts future nodes with the ADM, evaluates shortcut feasibility from these predicted states, and applies DP to obtain the minimum-hop path. As shown in Table 4, Greedy achieves a $1.83\times$ speedup but reduces success from 0.924 to 0.892, while Online Pruning reaches a similar $1.84\times$ speedup with a further drop to 0.830. This suggests that explicit planning over ADM-predicted future states can suffer from model error. ACP instead amortizes the offline DP supervision into a lightweight scheduler, achieving the highest success rate of 0.942 while retaining a $1.49\times$ speedup, thereby avoiding recursive world-model rollout at deployment while preserving useful long-horizon pruning preferences.

Table 4: **Impact of Amortized Scheduler.** We compare local greedy pruning, online pruning: online DAG optimization over ADM-predicted future states, and the amortized ACP scheduler.

Method	Success rate	Steps	Speedup
-base	0.924	267.9	1.00×
+Greedy	0.892	146.6	1.83×
+Online Pruning	0.830	**145.5**	**1.84**×
+ACP (Ours)	**0.942**	179.5	1.49×

Impact of Maximum stride. We next vary the maximum candidate stride S to examine how the shortcut search range affects ACP. As shown in Figure 5, increasing S from 2 to 3 substantially shortens successful trajectories, while further enlarging the candidate set yields only marginal changes, with average trajectory length remaining around 178–180 steps. In contrast to the clear efficiency

saturation, the success rate stays stable at approximately 0.94 across all settings, indicating that ACP is relatively insensitive to the maximum stride once a moderate candidate range is available. We use $S = 5$ as the default, achieving a success rate of 0.942 with an average trajectory length of 179.5 steps while still allowing occasional longer shortcuts.

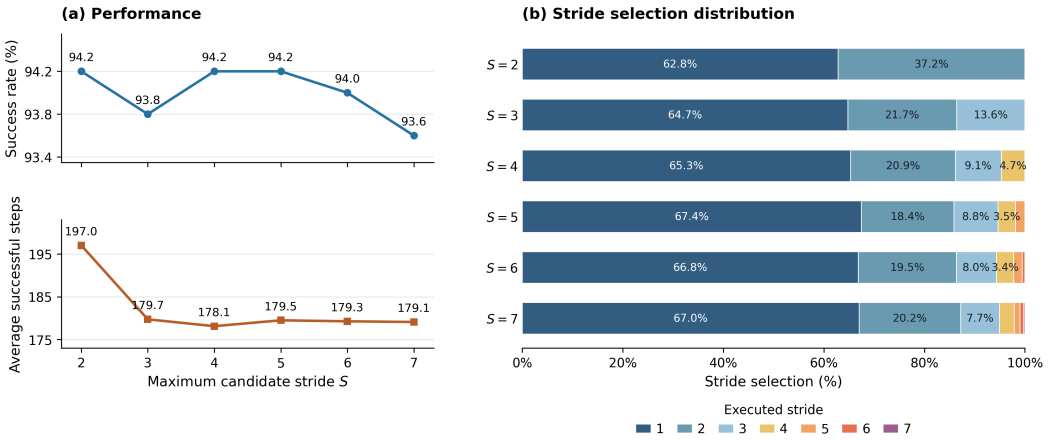


Figure 5: **Impact of Maximum stride.** (a) Success rate and average successful trajectory length for $S \in \{2, \ldots, 7\}$. (b) Stride selection distributions across different maximum strides. Performance largely saturates beyond moderate stride ranges, with execution dominated by short strides.

Sensitivity on ϵ. Finally, we vary the shortcut feasibility threshold ϵ to study its effect on pruning aggressiveness. As shown in Table 5, increasing ϵ consistently reduces the number of executed steps, from 232.6 at $\epsilon = 0.005$ to 142.8 at $\epsilon = 0.015$, corresponding to a speedup increase from $1.15\times$ to $1.88\times$. This gain comes with a clear performance trade-off: success decreases from 0.954 to 0.868 as increasingly permissive shortcuts are admitted. We use $\epsilon = 0.01$ as the default, which retains a success rate of 0.942 while achieving a $1.49\times$ speedup.

Table 5: **Sensitivity on ϵ.** Performance comparison across different feasibility tolerance values ϵ for trajectory deviation.

Ablation	Success Rate	Steps	Speedup
-base	0.924	267.9	1.00×
+ACP-0.005	0.954	232.6	1.15×
+ACP-0.01	0.942	179.5	1.49×
+ACP-0.015	0.868	142.8	1.88×

5 Related Work

Imitation Learning for Embodied Policies. Imitation learning has become a dominant paradigm for robotic manipulation (Mandlekar et al., 2022; 2023), enabling policies to acquire complex visuomotor behaviors from expert demonstrations (Khazatsky et al., 2024; Open X-Embodiment Collaboration, 2024). Recent advances in action chunking (Zhao et al., 2023; Haldar et al., 2024), generative policies (Chi et al., 2023; Ze et al., 2024), and vision-language-action models (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2025a) have substantially improved the capability and generalization of imitation-trained policies (Liu et al., 2025; Ghosh et al., 2024). However, because these policies learn from demonstrated behaviors, their execution characteristics can remain coupled to the temporal structure of the training data, including motion tempo and waypoint density (Shi et al., 2023; Guo et al., 2025). This issue is particularly relevant for human teleoperation data, where slow and cautious demonstrations can lead learned policies to inherit similarly inefficient execution patterns (Wu et al., 2026; Ramirez Sanchez et al., 2026).

Embodied Policy Acceleration. Accelerating embodied policy execution has been explored across different abstraction levels (Prasad et al., 2024; Zhang et al., 2025; Black et al., 2025c), ranging from dataset refinement (Shi et al., 2023) to chunk-level scheduling (Liang et al., 2026; Xu et al., 2026). Dataset-level approaches identify portions of demonstrations that can be safely accelerated and construct temporally compressed training data (Guo et al., 2025; Ramirez Sanchez et al., 2026), allowing imitation policies to learn faster behaviors after retraining. Policy-level approaches instead optimize execution speed through additional reinforcement learning or policy adaptation, enabling behaviors beyond the tempo of the base demonstrations but requiring further policy optimization or online interaction (Nam et al., 2025). More recently, policy acceleration has been formulated as a constrained Markov decision process, with an external module predicting a downsampling rate for each action chunk to enable state-dependent acceleration without modifying the base policy (Wu et al., 2026). This rate-based formulation applies a uniform temporal transformation to an entire chunk, which may be restrictive when redundant actions are distributed non-uniformly within the chunk. In contrast, ACP formulates policy acceleration as action-chunk pruning, selectively bypassing redundant intermediate actions to permit non-uniform execution within each predicted chunk.

6 Conclusion

In this paper, we present **ACP**, a lightweight framework for accelerating action-chunking policies through action-chunk pruning. ACP evaluates feasible shortcuts along offline demonstration trajectories, computes globally optimized next-hop decisions over the resulting shortcut DAG, and amortizes these decisions into a plug-and-play scheduler for real-time execution. Simulated and real-world experiments show that ACP reduces execution steps while preserving policy performance, demonstrating the benefit of adapting execution density within individual action chunks. These findings suggest that optimizing action-chunk execution offers a practical avenue for improving execution efficiency.

AI use statement

In this work, we used generative AI tools to assist with research ideation and execution, including providing feedback on experimental design, troubleshooting implementation issues, and assisting with the interpretation of experimental results. We also used generative AI tools for literature retrieval and discovery, including identifying potentially relevant prior work and summarizing existing literature, as well as for polishing parts of the manuscript. All AI-assisted research suggestions were independently reviewed by the authors; implementation-related suggestions were verified against the actual code and experimental results, literature suggestions were checked against the original sources, and all AI-assisted text was manually reviewed and revised. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

Reproducibility statement

We provide the implementation and experimental details necessary to reproduce our results throughout the paper and appendix. The main text describes the ACP formulation, training procedure, and evaluation protocol. Appendix A provides the simulation benchmark and base-policy inference settings, Appendix B details the real-world platform, tasks, data collection, and policy fine-tuning setup, and Appendix C specifies the deviation metric, gripper compensation, network architectures, and ACP hyperparameters. Together, these sections document the model configuration, training settings, and evaluation procedures used in our experiments.

References

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, et al. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. In *Proceedings of the Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, et al. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, 2025b. doi: 10.15607/RSS.2025.XXI.010.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies. In *Advances in Neural Information Processing Systems*, 2025c.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1724–1734, 2014. doi: 10.3115/v1/D14-1179.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems*, 2018.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, et al. Octo: An Open-Source Generalist Robot Policy. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.090.

Lingxiao Guo, Zhengrong Xue, Zijing Xu, and Huazhe Xu. DemoSpeedup: Accelerating Visuomotor Policies via Entropy-Guided Demonstration Acceleration. In *Proceedings of the Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 599–609. PMLR, 2025.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering Diverse Domains through World Models. arXiv preprint arXiv:2301.04104, 2023.

Siddhant Haldar, Zhuoran Peng, and Lerrel Pinto. BAKU: An Efficient Transformer for Multi-Task Policy Learning. In *Advances in Neural Information Processing Systems*, 2024.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, Robust World Models for Continuous Control. In *International Conference on Learning Representations*, 2024.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to Trust Your Model: Model-Based Policy Optimization. In *Advances in Neural Information Processing Systems*, 2019.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, et al. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.120.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, et al. OpenVLA: An Open-Source Vision-Language-Action Model. In *Proceedings of the Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20802–20811, 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Advances in Neural Information Processing Systems*, 2023.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. RDT-1B: A diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, 2025.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *Proceedings of the Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1678–1690. PMLR, 2022.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations. In *Proceedings of the Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1820–1864. PMLR, 2023.

Taewook Nam, Junmo Cho, Youngsoo Jang, and Sung Ju Hwang. SpeedAug: Policy Acceleration via Tempo-Enriched Policy and RL Fine-Tuning. arXiv preprint arXiv:2512.00062, 2025.

Open X-Embodiment Collaboration. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024. doi: 10.1109/ICRA57147.2024.10611477.

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. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.012.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency Policy: Accelerated Visuomotor Policies via Consistency Distillation. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.071.

Robert Ramirez Sanchez, Daniel J. Evans, Dylan P. Losey, and Siddarth Jain. VOLT: Vision and Language Trajectory Segmentation for Faster-than-Demonstration Policies. arXiv preprint arXiv:2606.06323, 2026.

Lucy Xiaoyang Shi, Archit Sharma, Tony Z. Zhao, and Chelsea Finn. Waypoint-Based Imitation Learning for Robotic Manipulation. In *Proceedings of the Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2195–2209. PMLR, 2023.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. CLIPort: What and Where Pathways for Robotic Manipulation. In *Proceedings of the Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 894–906. PMLR, 2022.

Zhichao Wu, Junyin Ye, Zhilong Zhang, Yihao Sun, Haoxin Lin, Jiaheng Luo, Haoxiang Ren, Lei Yuan, and Yang Yu. Speedup Patch: Learning a plug-and-play policy to accelerate embodied manipulation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Runze Xu, Xiaolong Shan, Shuang Dai, Yu Wang, and Jincheng Yu. Knowing When to Stop: Adaptive Action Chunking via Internal Cross-Attention Dynamics in VLAs. arXiv preprint arXiv:2609.00908, 2026.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3D Diffusion Policy: Generalizable Visuomotor Policy Learning via Simple 3D Representations. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.067.

Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, et al. Transporter Networks: Rearranging the Visual World for Robotic Manipulation. In *Proceedings of the Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pp. 726–747. PMLR, 2021.

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. FlowPolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14754–14762, 2025. doi: 10.1609/aaai.v39i14.33617.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Proceedings of the Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023.

A Details of Simulation Experiment

A.1 LIBERO Benchmark

LIBERO (Liu et al., 2023) is a tabletop manipulation benchmark with four commonly used 10-task suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. Spatial varies object locations to test spatial reasoning, Object varies manipulated objects under similar task structures, Goal changes task objectives within similar scenes, and Long contains longer-horizon multi-stage manipulation tasks. LIBERO is simulated in MuJoCo with a Franka Panda robot at 20 Hz. An episode succeeds when the task goal is satisfied and fails if the goal is not reached within the evaluation horizon; all methods use the same benchmark-provided initial states.

Each observation contains two RGB views, robot proprioception, and the task language instruction, formally defined as

$$o_t = \left(I_t^{\text{side}},\, I_t^{\text{wr}},\, s_t,\, l_t\right), \tag{10}$$

where I_t^{side} and I_t^{wr} represent the side and wrist RGB images, s_t denotes the robot proprioceptive state, and l_t is the task language instruction. The robot state is defined as:

$$s_t = [p_t,\, r_t,\, g_t] \in \mathbb{R}^8, \tag{11}$$

where $p_t \in \mathbb{R}^3$ is the end-effector position, $r_t \in \mathbb{R}^3$ is the axis-angle orientation, and $g_t \in \mathbb{R}^2$ denotes the gripper joint positions. The action is defined as:

$$a_t = [\Delta p_t,\, \Delta r_t,\, u_t] \in \mathbb{R}^7, \tag{12}$$

where $\Delta p_t, \Delta r_t \in \mathbb{R}^3$ are end-effector translation and rotation increments, respectively, and u_t is the gripper command.

We use the official $\pi_{0.5}$ LIBERO checkpoint from OpenPI (Black et al., 2025a) and follow its standard inference setup. Each policy query predicts an action chunk of horizon $H = 10$.

B Details of Real-World Experiment

B.1 Hardware Setup

Our real-world experiments are conducted on an AgileX PiPER robotic arm, as shown in Figure. 3. We collect demonstrations through leader–follower teleoperation and use two Intel RealSense D435i cameras, mounted at the robot wrist and beside the workspace, to provide RGB observations for policy inference.

B.2 Real-World Tasks

Table Cleanup. The robot collects multiple objects scattered on the tabletop and places them into a storage box. The task involves repeated grasping, free-space transport, and object placement.

Marker Insertion. The robot grasps a marker from the table and places it into a pen holder. Successful execution requires accurate alignment and placement near the end of the trajectory.

Cup Stacking. The robot sequentially grasps and stacks multiple cups. The task contains repeated grasping and transport.

For each task, we collect 50 demonstration trajectories and independently fine-tune the $\pi_{0.5}$ policy using LoRA ($r = 32$).

C Details of ACP

C.1 Calculation of Deviation

To evaluate whether a candidate shortcut action can faithfully approximate the motion generated by the base policy, we formulate a state deviation metric as the core feasibility criterion.

For a robot state s, we extract its end-effector (EEF) pose in the world frame as

$$\mathrm{EEF}(s) = (\mathbf{p}, \mathbf{r}), \tag{13}$$

where $\mathbf{p} \in \mathbb{R}^3$ denotes the Cartesian position and $\mathbf{r} \in \mathbb{R}^3$ is the axis-angle representation of the EEF orientation. For trajectory interpolation and rotational-distance computation, we convert $\mathbf{r}$ to its corresponding unit quaternion $\mathbf{q} \in \mathbb{R}^4$.

For a candidate shortcut with stride k ($i \to i + k$), the model first queries the original base actions to obtain the reference trajectory ($\hat{\mathbf{p}}^{\mathrm{ref}}_{i,j}$ and $\hat{\mathbf{q}}^{\mathrm{ref}}_{i,j}$) for $j = 1, \ldots, k$. Meanwhile, the shortcut action predicts the shortcut endpoint, which is then interpolated from the initial pose to its predicted endpoint over k steps (using linear interpolation for translation and normalized linear interpolation for orientation) to form the shortcut trajectory ($\tilde{\mathbf{p}}^{\mathrm{sc}}_{i,j}$ and $\tilde{\mathbf{q}}^{\mathrm{sc}}_{i,j}$). To compare the two motions at the same temporal resolution, the shortcut deviation is defined as the maximum aligned EEF pose discrepancy:

$$d(i,k) = \max_{1 \le j \le k} \left[\frac{1}{2} \left\| \hat{\mathbf{p}}^{\mathrm{ref}}_{i,j} - \tilde{\mathbf{p}}^{\mathrm{sc}}_{i,j} \right\|_2 + \arccos\left(\left| (\hat{\mathbf{q}}^{\mathrm{ref}}_{i,j})^\top \tilde{\mathbf{q}}^{\mathrm{sc}}_{i,j} \right| \right) \right]. \tag{14}$$

A shortcut is considered feasible when $d(i,k) \le \epsilon$.

C.2 Gripper Compensation

Our framework adaptively selects varying strides within action chunks to construct shortcuts. However, gripper actions require separate consideration due to the discrete nature of their control and specific physical constraints.

Shortcut execution compresses multiple base control steps into a single shortcut action, reducing the total number of commands applied over the same motion segment. Because of the discrete control nature in simulation environments, this compression causes a mismatch in the cumulative physical displacement of the gripper. If a full grasp requires several consecutive closure commands, reducing the command frequency via shortcuts results in the gripper failing to reach the intended state in time, leading to failed grasps.

To resolve this inconsistency and maintain task success rates, we applied task-specific compensations. Specifically, for **LIBERO**, we doubled the magnitude (velocity) of each gripper action command in the shortcut. For instance, if a single base action resulted in a 0.1 cm closure, the adjusted shortcut action for a stride of 2 produces a 0.2 cm closure, ensuring the gripper responds aggressively enough to maintain physical state consistency. Conversely, for **Real-World** deployment, since we use continuous values for gripper control, no specific gripper compensation is required.

C.3 Network Architecture

Any-Step Dynamics Model. The ADM models the state transition induced by a variable-length action sequence (Cho et al., 2014), using a recurrent latent state to accumulate the effect of successive actions without autoregressively feeding back predicted states. The initial robot state is encoded into a latent representation h_t, which initializes a Gated Recurrent Unit (GRU). Given a variable-length action sequence, the GRU updates its hidden state sequentially as

$$h_{t+i+1} = g_\theta(h_{t+i}, a_{t+i}), \tag{15}$$

and a transition decoder maps each hidden state to the corresponding future robot-state prediction. Predicted states are not fed back into the recurrent dynamics; the hidden state evolves only through the action sequence.

Scheduler. The scheduler f_ϕ takes the current robot state s_t and the predicted action chunk A_t as input. It separately projects the robot state and action features into a shared representation space, concatenates them, and processes the combined features through an MLP backbone to produce the next-hop logits, where each row corresponds to a position in the action chunk and each column corresponds to a candidate stride.

C.4 ACP HYPERPARAMETERS

Table 6: **Hyperparameters for ADM.**

Hyperparameter	LIBERO	Real World
Hidden dimension	256	256
GRU layers	3	3
Batch size	512	512
Learning rate	1×10^{-4}	1×10^{-4}
Training epochs	100	100

Table 7: **Hyperparameters for Scheduler.**

Hyperparameter	LIBERO	Real World
Chunk horizon	10	10
Strides	$(1, 2, 3, 4, 5)$	$(1, 2, 3, 4, 5)$
Epsilon ϵ	0.01	0.03
Hidden dimension	256	256
MLP layers	3	3
Batch size	512	512
Learning rate	3×10^{-4}	3×10^{-4}
Training epochs	100	100