SkiP: When to Skip and When to Refine for Efficient Robot Manipulation

Mingtong Dai[1,2,6] **Guanqi Peng**[2,3] **Yongjie Bai**[2,4] **Feng Yan**[5]
Chunjie Chen[1] **Lingbo Liu**[2*] **Liang Lin**[2,4] **Xinyu Wu**[1]
[1]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
[2]Peng Cheng Laboratory [3]Southern University of Science and Technology
[4]Sun Yat-sen University [5]Xi'an Jiaotong University
[6]University of Chinese Academy of Sciences

Abstract

Previous imitation learning policies predict future actions at every control step, whether in smooth motion phases or precise, contact-rich operation phases. This uniform treatment is wasteful: predictable free-space motion and precise contact-rich manipulation demand different temporal resolutions. We partition demonstrations into *key segments* requiring dense refinement and *skip segments* containing temporally redundant motion. We propose a novel *action relabeling* mechanism: at each timestep in a skip segment, we replace the behavior cloning target with the action at the entrance of the next key segment, enabling the policy to leap over redundant steps in a single decision. The resulting **Skip Policy (SkiP)** dynamically leaps over skip segments and intensively refines actions in key segments, within a single unified network requiring no learned skip planner or hierarchical structure. To obtain this partition without manual annotation, we introduce *Motion Spectrum Keying* (MSK), a fast, task-agnostic procedure that analyzes the local frequency structure of trajectory velocities. Extensive experiments across 72 simulated manipulation tasks and three real-robot tasks show that SkiP reduces executed steps by 15–40% while matching or improving success rates across various policy backbones. Project page: `https://pgq18.github.io/SkiP-page/`.

1 Introduction

A robot reaching for a bottle on a shelf traverses half a meter of empty air in a smooth arc, then must carefully close its gripper around the neck. Transit through free space and contact-rich manipulation demand fundamentally different control resolutions, yet behavior cloning treats every timestep identically: one observation in, one action out, repeated hundreds of times per episode. In free-space segments, step-by-step prediction is redundant; intermediate prediction errors can shift subsequent observations away from the demonstration distribution and accumulate over a rollout [23, 24]. In contact segments, the policy must react densely to maintain precision. Standard imitation learning satisfies neither need well.

How should a policy allocate its decisions across these regimes? Prior work answers by enriching the *model*: action chunking predicts short action windows [38, 5, 37]; keyframe methods factor trajectories into sparse anchors and dense connectors [32, 36]; hierarchical policies add planners or options on top of low-level controllers [29]. These approaches improve capacity, but the underlying policy still *executes* at a uniform temporal rate: every timestep receives the same treatment regardless of its information content. Speed-adaptive methods [1] adjust velocity but do not modify what the

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

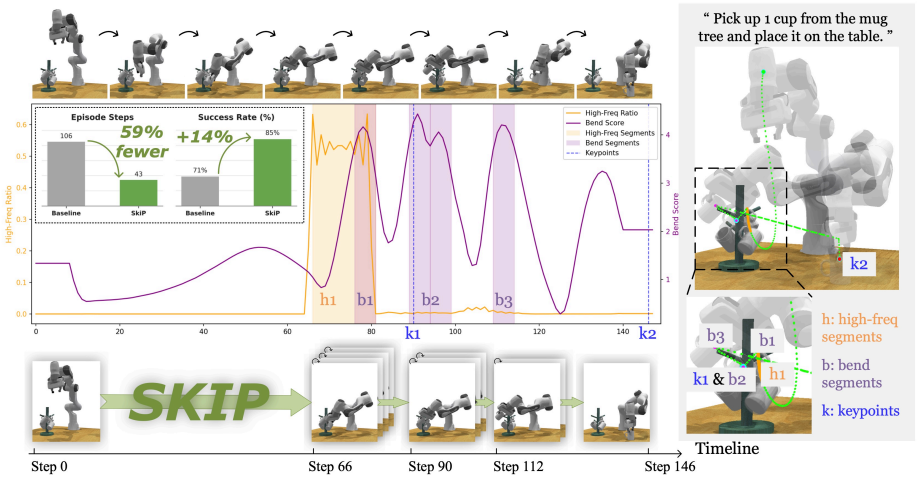


Figure 1: "Pick up 1 cup from the mug tree and place it on the table" analyzed by SkiP. **Top:** high-frequency energy ratio (orange, Eq. (6)) and bend score (purple, Eq. (7)), defined in §3.3; shaded regions are key segments. **Bottom:** SkiP skips from step 0 to step 66, then refines through contacts.

policy learns to predict. In short, prior methods change how the policy generates actions, not *what* it is supervised to predict.

We take a different route (Figures 1 and 2). Rather than adding architectural complexity, we modify the *training target* itself. We partition demonstrations into *key segments* requiring dense refinement and *skip segments* containing temporally redundant motion. At each timestep in a skip segment, we replace the behavior cloning target with the action at the entrance of the next key segment; inside key segments, the target remains the immediate next step. We call this *action relabeling*. The resulting policy, **SkiP** (Skip Policy), learns *when to skip*, advancing past predictable motion in a single decision, and *when to refine*, predicting dense corrections near contacts. It does so within a single network, requiring no architectural changes, no learned skip planner, and no additional inference cost.

Skipping removes unnecessary intermediate predictions along free-space motion, reducing opportunities for sequential prediction errors to accumulate. Our controlled re-querying study (Appendix A) supports the efficiency of this learned behavior: executing the original SkiP predictions achieves comparable success to full-jump re-querying with fewer policy calls.

To partition demonstrations into key and skip segments, we introduce *Motion Spectrum Keying* (MSK), which identifies key segments by analyzing the local frequency structure of trajectory velocities. MSK is fast, task-agnostic, and requires no learned components.

Our contributions are as follows:

1. **Action relabeling.** We show that modifying the behavior cloning target alone is sufficient to produce adaptive skip-and-refine behavior, without architectural changes or learned planners. This demonstrates that temporal resolution in imitation learning can be controlled through the training objective.

2. **Motion Spectrum Keying (MSK).** A frequency-domain procedure that partitions demonstrations into key and skip segments without learning, connecting spectral signal processing to imitation learning supervision design.

3. **Strong empirical performance.** SkiP achieves state-of-the-art results across extensive robot manipulation benchmarks, improving both success rate and execution efficiency while preserving the same policy backbones. We further analyze the learned action magnitudes and show that the policy acquires distinct skip and refine modes (§4.5).

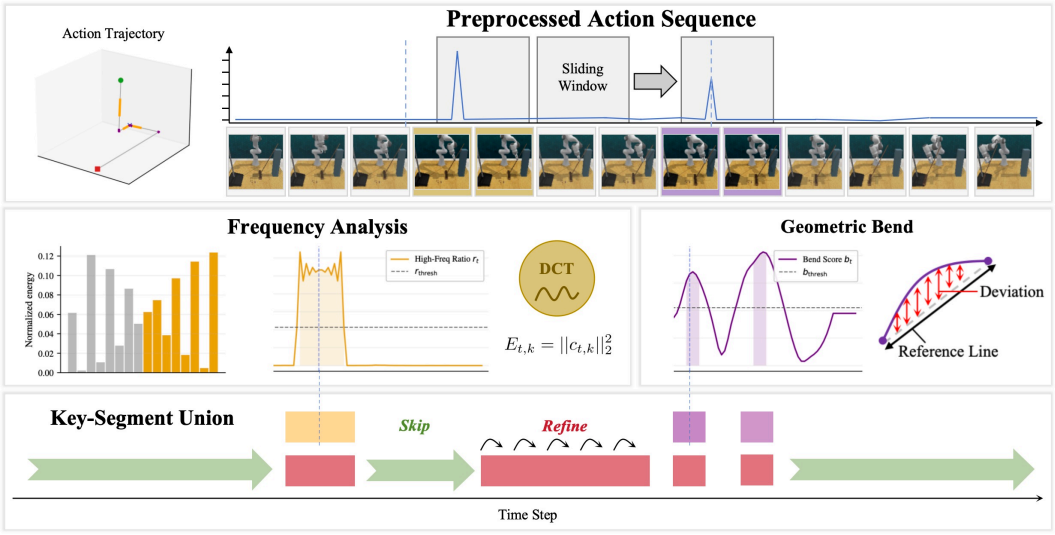


Figure 2: Overview of SkiP. We partition each demonstration into high-information *key segments* and low-information *skip segments* via DCT spectral analysis, then *relabel* training targets: in skip segments, the target is the action at the next key-segment entrance; in key segments, the target is the next-step continuation.

2 Related Work

Dense visuomotor imitation. Transformer-based manipulation policies such as PerAct [27], HiveFormer [9], and Act3D [6] predict 6-DoF actions from structured 3D representations or language-conditioned histories. RVT [8] and TVVE [2] further improve view-based 3D manipulation by predicting keyframe-style actions from compact visual representations. Language-conditioned manipulation systems [26, 17, 33] combine semantic task understanding with spatially precise action prediction. Diffusion Policy [5], 3D Diffuser Actor [14], and Action Chunking with Transformers (ACT) [38] model short action windows via denoising or autoregressive prediction; this reduces replanning frequency but execution remains uniform. Behavior Transformer [25] and other autoregressive sequence models further explore multimodal or variable-length action prediction [37]. At a larger scale, complex manipulation benchmarks [13, 19, 18, 20, 39] and generalist policies [4, 41, 30, 15] show that broad data and vision-language pretraining improve visuomotor control. These works mainly change the action generator, visual representation, or data scale, whereas SkiP changes the supervision target used to allocate temporal resolution.

Keyframe trajectory structuring and temporal abstraction. Temporal abstraction through sparse anchors is a classical strategy for long rollouts; in reinforcement learning it is formalized through the options framework [29]. ChainedDiffuser [32] unifies keypose prediction with a diffusion module that fills in connecting segments. Chain-of-Action (CoA) [36] generates full trajectories via backward reasoning from a predicted keyframe under a trajectory autoregressive formulation. Keyframe-focused imitation [31] and automated annotation pipelines [16] emphasize key moments in demonstrations. These methods exploit sparse structure but still generate and execute dense trajectories between anchors; the modification is to the *model*, not to the *supervision*. Faster-execution methods [1, 10] are closest to SkiP, but adapt execution velocity or the execution pipeline rather than the training target.

Frequency-domain action representations. FAST [22] uses DCT-based tokenization for autoregressive VLA policies. FreqPolicy variants explore hierarchical frequency modeling [40] and frequency consistency for one-step generation [28]. Wavelet Policy [34] operates in the wavelet domain. These works use frequency representations as a modeling space for dense trajectory generation. SkiP uses spectral energy as a *supervisory signal* for temporal abstraction: the frequency content tells us *where* to allocate dense prediction, not how to represent the action itself. We call this procedure Motion Spectrum Keying (§3.3).

3 Method

The design separates two questions: *what* information a trajectory region carries and *how* the policy should respond to that region. Because the two are decoupled, the relabeling mechanism works with any binary temporal annotation, while the labeling procedure can be swapped or improved independently.

3.1 Problem Setup

We consider imitation learning from offline robot demonstrations. Each demonstration is a trajectory $\tau = \{(o_t, a_t)\}_{t=1}^{T}$, where o_t denotes the observation at time t and $a_t \in \mathbb{R}^d$ denotes a continuous control command. Our goal is to learn a policy $\pi_\theta(a \mid o)$ that solves the task while using decision steps efficiently.

Trajectory information density is highly non-uniform: during contacts and precise alignment, the policy must refine actions densely, but during smooth motions, step-by-step prediction is redundant. SkiP allocates decision steps accordingly, skipping through skip segments and concentrating refinement on key segments.

The framework has two components. *Action relabeling* (§3.2) is the core learning mechanism: given any binary temporal annotation $y_t \in \{0, 1\}$ that marks refine-worthy timesteps, it modifies the behavior cloning targets so that a single policy learns both skipping and refinement. *Motion Spectrum Keying* (MSK, §3.3) is one concrete instantiation of this annotation, based on short-time DCT spectral energy.

3.2 Action Relabel: Skip-and-Refine Learning

Assume we have a set of key segments $\mathcal{S} = \{[s_i, e_i]\}_{i=1}^{M}$ extracted from each demonstration (§3.3; illustrated in Figure 3a). Here s_i and e_i denote the start and end timestep of the i-th key segment. Let $y_t \in \{0, 1\}$ be a per-timestep indicator: $y_t = 1$ if t lies inside a key segment, $y_t = 0$ otherwise.

Observations from both key and skip segments remain training inputs; the relabeled supervision targets teach a single policy two behaviors under the same action interface. When $y_t = 1$, the policy should perform dense refinement by predicting the immediate next-step continuation. When $y_t = 0$ and a subsequent key segment exists, the policy should skip toward the next interaction-heavy phase by predicting the action at the *entrance* of that key segment.

Formally, define the next key-segment start time:

$$t^+(t) = \min\{s_i : s_i > t\}, \tag{1}$$

with $t^+(t)$ undefined if no future key segment exists. We construct a relabeled chunk start index:

$$t^\star(t) = \begin{cases} t+1, & y_t = 1, \\ t^+(t), & y_t = 0 \text{ and } t^+(t) \text{ exists}, \\ t+1, & \text{otherwise}, \end{cases} \tag{2}$$

and use it to form a relabeled target chunk $\tilde{\mathbf{A}}_t \in \mathbb{R}^{H \times d}$ for chunk index $h = 1, \ldots, H$, with a padding mask $m_{t,h} \in \{0, 1\}$:

$$\begin{aligned} k_{t,h} &\triangleq t^\star(t) + h - 1, \\ \tilde{a}_{t,h} &= \begin{cases} a_{k_{t,h}}, & k_{t,h} \le T, \\ \mathbf{0}, & \text{otherwise}, \end{cases} \\ m_{t,h} &= \mathbb{I}[k_{t,h} \le T], \end{aligned} \tag{3}$$

where H is the chunk length. We then minimize a masked imitation loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(o_t, \cdot) \sim \mathcal{D}} \left[\frac{1}{\sum_h m_{t,h}} \sum_{h=1}^{H} m_{t,h}\, \ell(\hat{a}_{t,h},\, \tilde{a}_{t,h}) \right], \tag{4}$$

where ℓ is the imitation loss used by the corresponding backbone. This relabeling unifies skipping and refinement within a single policy. Inside key segments, $t^\star(t) = t + 1$ so the target chunk matches the immediate next-step continuation, training dense feedback-driven refinement. Outside key segments, $t^\star(t) = t^+(t)$ so the target chunk starts at the next key-segment entrance; the policy is trained to jump ahead, implicitly skipping the intermediate skip-segment actions.

3.3 Motion Spectrum Keying (MSK)

We now describe how the binary annotation y_t is produced. We introduce *Motion Spectrum Keying* (MSK), which extracts key segments by analyzing the local frequency structure of trajectory velocities. Prior methods rely on heuristic keyframes such as gripper-state changes or velocity zero-crossings [12], which capture pick-and-place events but miss sustained high-precision motion like sweeping or dragging. MSK measures local motion complexity directly.

DCT decomposition. Given an action sequence $\{a_t\}_{t=1}^{T}$, we compute per-step velocities $v_t = a_t - a_{t-1}$. For each t, we apply a discrete cosine transform along the temporal axis over a centered length-W window of velocities to obtain spectral coefficients

$$c_{t,k} = \sum_{n=0}^{W-1} \alpha_k \, v_{t,n} \cos\left(\frac{\pi(2n+1)k}{2W}\right), \tag{5}$$

where α_k is the standard normalization constant. We compute spectral energy $E_{t,k} = \|c_{t,k}\|_2^2$ and define the high-frequency energy ratio

$$r_t = \frac{\sum_{k\in\mathcal{H}} E_{t,k}}{\sum_k E_{t,k}}, \qquad \mathcal{H} = \{k : k \geq \lceil W/2 \rceil\}, \tag{6}$$

where $\mathcal{H}$ is the upper half of the frequency spectrum. Large r_t indicates rapid local changes associated with contacts or corrections. We threshold $\{r_t\}$ at the per-episode quantile q and group consecutive positives into segments.

Bend-based augmentation. Some task-relevant events appear as brief geometric deviations rather than high-frequency oscillations. We complement the spectral signal with a bend score that compares end-effector translations against a straight-line reference over the same window:

$$b_t = \frac{1}{W} \sum_{n=0}^{W-1} \frac{\|p_{t,n} - \ell_{t,n}\|_2}{\bar{s}_t}, \tag{7}$$

where $\ell_{t,n}$ is the linear interpolant between the first and last in-window translations and $\bar{s}_t$ is the mean step length, making b_t scale-invariant. Timesteps with b_t above its per-episode threshold are added to the key segments.

Final partition. The key segments are the union of frequency-based segments, bend-based segments, and small neighborhoods around heuristic keyframes (gripper-state changes and end-effector velocity zero-crossings, following James and Abbeel [12]); the complement forms the skip segments. We sweep the window size and quantile threshold in Sec. 4.4 and ablate the bend and keyframe-union components in Table 6; full implementation details are in Appendix F.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate SkiP across four settings that vary in policy architecture, observation modality, and embodiment. On RLBench [13], we use 60 manipulation tasks with a 7-DoF Franka Panda, 4 RGB cameras at 128×128, 100 demonstrations per task, and a transformer-based policy adapted from CoA [36]. On RoboMimic [18], we evaluate 4 image-based manipulation tasks using a Diffusion Policy UNet backbone [5]. On RoboTwin [20], we evaluate 8 bimanual tasks with a DP3 [35] backbone in the clean point-cloud setting. For real-robot evaluation, we fine-tune the pretrained $\pi_{0.5}$ [11] on 3 tabletop tasks and include RLBench-18 as a simulation counterpart.

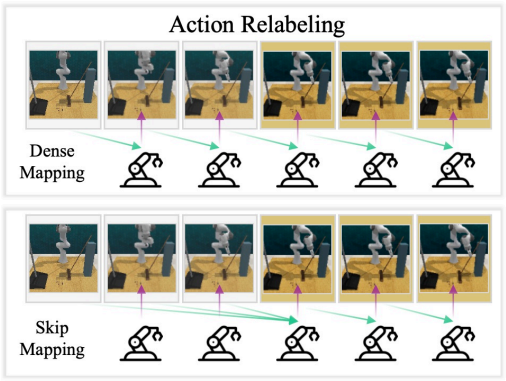
Baselines. On RLBench, we compare against DP [5], ACT [38], Chain-of-Action forward/reverse (CoA-fwd/CoA-rev) [36], and a keyframe-only variant (KF-only). On RoboMimic and RoboTwin, we compare against the respective backbone baselines and CoA variants.

Metrics. We report task *success rate* (SR) and execution efficiency measured by average executed control steps per episode (*Steps*). We additionally report *Steps*$_{succ}$, the average steps over successful episodes only, to isolate efficiency conditional on completion.

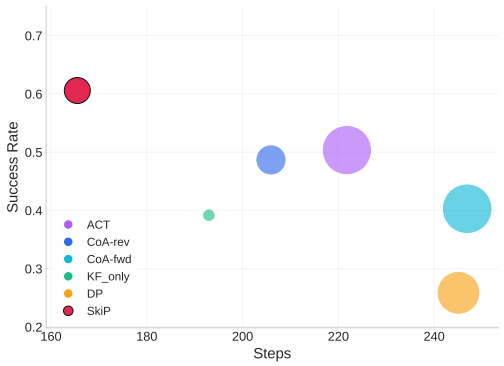
Table 1: RLBench-10 per-task SR↑ and Steps↓ (mean±std over three runs). Blue saturation: darker = better. † SkiP variant that replaces MSK with key segments from Gemini-2.5-Pro [7]. Best in **bold**, second underlined.

Method	Avg		Overall		open-box		open-drawer		pick-up-cup		press-switch	
	SR↑	Steps↓	Steps_s↓	rk↓	SR	Steps	SR	Steps	SR	Steps	SR	Steps
DP	0.43 ± .02	160.0 ± 3	119.1 ± 2	5.4	0.34 ± .02	201.5 ± 4	0.26 ± .06	147.0 ± 6	0.35 ± .10	214.7 ± 25	0.45 ± .03	146.9 ± 10
KF-only	0.49 ± .01	113.1 ± 3	**10.9 ± 3**	4.3	0.17 ± .03	198.0 ± 7	0.55 ± .07	82.1 ± 13	0.69 ± .05	**94.4 ± 15**	0.34 ± .06	187.0 ± 9
CoA-fwd	0.68 ± .01	161.5 ± 1	133.3 ± 1	3.5	0.64 ± .00	185.8 ± 0	0.81 ± .02	119.4 ± 1	0.56 ± .03	194.3 ± 9	0.28 ± .00	243.8 ± 0
ACT	0.71 ± .01	139.4 ± 1	106.1 ± 1	3.1	0.53 ± .03	181.2 ± 2	**1.00 ± .00**	90.8 ± 0	0.55 ± .04	174.2 ± 10	**0.58 ± .03**	177.1 ± 4
CoA-rev	0.70 ± .05	127.5 ± 5	86.2 ± 3	3.1	0.72 ± .35	158.4 ± 31	0.79 ± .03	101.5 ± 6	0.68 ± .03	138.3 ± 7	0.46 ± .02	191.3 ± 5
SkiP†	0.30 ± .01	183.1 ± 2	97.6 ± 7	6.3	0.24 ± .07	208.5 ± 9	0.23 ± .02	165.7 ± 1	0.13 ± .05	272.1 ± 9	0.35 ± .05	200.2 ± 8
SkiP	**0.85 ± .01**	**72.9 ± 2**	43.4 ± 1	**1.6**	**0.91 ± .02**	**87.1 ± 7**	1.00 ± .00	**39.8 ± 2**	**0.74 ± .06**	96.2 ± 17	0.54 ± .03	**144.4 ± 7**

Method	push-button		reach-target		stack-wine		sweep-dustpan		take-lid-off		turn-tap	
	SR	Steps	SR	Steps	SR	Steps	SR	Steps	SR	Steps	SR	Steps
DP	0.66 ± .04	130.8 ± 4	0.60 ± .03	57.6 ± 4	0.40 ± .04	216.9 ± 5	0.08 ± .03	165.4 ± 2	0.73 ± .06	127.8 ± 8	0.44 ± .03	191.2 ± 10
KF-only	0.64 ± .03	95.2 ± 9	**0.72 ± .00**	**37.2 ± 0**	0.48 ± .06	97.4 ± 10	0.04 ± .00	163.4 ± 0	0.80 ± .00	48.5 ± 1	0.50 ± .03	127.3 ± 8
CoA-fwd	0.97 ± .04	126.6 ± 3	0.68 ± .00	68.7 ± 0	0.52 ± .00	213.5 ± 1	0.96 ± .00	120.2 ± 0	0.73 ± .02	159.5 ± 2	0.64 ± .03	183.6 ± 8
ACT	0.58 ± .07	162.5 ± 5	0.72 ± .00	46.6 ± 4	0.90 ± .02	163.6 ± 2	0.88 ± .00	111.8 ± 0	0.80 ± .03	113.1 ± 4	0.57 ± .05	172.8 ± 7
CoA-rev	0.88 ± .05	100.8 ± 11	0.69 ± .02	70.5 ± 3	0.69 ± .03	146.6 ± 5	0.73 ± .40	105.7 ± 35	0.83 ± .02	103.2 ± 4	0.54 ± .03	158.4 ± 6
SkiP†	0.65 ± .04	118.8 ± 8	0.56 ± .00	74.5 ± 0	0.03 ± .02	244.6 ± 4	0.15 ± .02	159.8 ± 2	0.33 ± .02	188.3 ± 6	0.37 ± .02	198.3 ± 6
SkiP	**0.98 ± .02**	**26.2 ± 4**	0.68 ± .00	45.3 ± 0	**1.00 ± .00**	**77.3 ± 0**	**1.00 ± .00**	**59.4 ± 0**	**0.97 ± .02**	**33.4 ± 3**	**0.68 ± .03**	**120.0 ± 8**



(a) Action relabeling. Top: standard dense mapping (every timestep predicts the next). Bottom: skip mapping (skip-segment timesteps predict the next key-segment entrance).



(b) SR vs. Steps on RLBench-60. Bubble area encodes $\text{Steps}_{\text{succ}}$ (smaller = more efficient).

Figure 3: (a) Illustration of SkiP's relabeling scheme: in skip segments, the training target jumps to the next key-segment entrance. (b) SkiP sits in the upper-left: highest SR, fewest steps.

4.2 Simulation Benchmarks

RLBench. We evaluate on the RLBench-10 task suite used by CoA (Table 1) and the broader RLBench-60 suite (Figure 3(b)), with the remaining 50 tasks shown in Figure 4. On RLBench-10, SkiP achieves the highest average SR (0.850) while requiring the fewest executed Steps (72.9) among all methods, improving over CoA-rev by +0.149 SR and 54.6 fewer steps. KF-only attains very low $\text{Steps}_{\text{succ}}$ (10.9) but unreliable SR (0.493).

Figure 3(b) plots the SR–Steps trade-off across all 60 RLBench tasks: SkiP sits in the upper-left corner with both the highest SR and the fewest steps, and the smallest $\text{Steps}_{\text{succ}}$ bubble among methods at comparable SR.

Figure 4 compares SkiP with CoA-rev on the 50 tasks outside RLBench-10. SkiP improves success on a broad range of tasks, especially long-horizon tasks where concentrating control steps around key segments matters most.

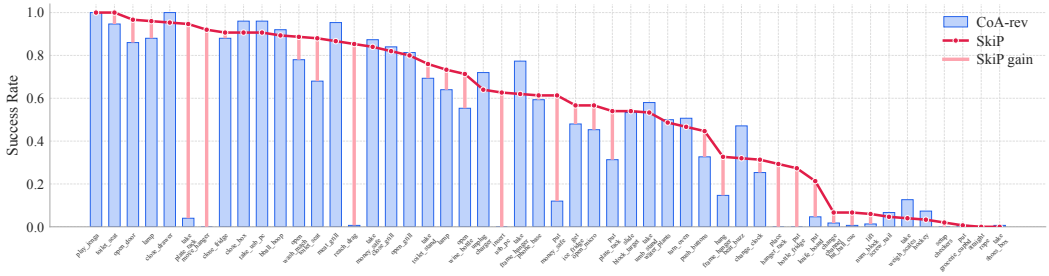


Figure 4: Per-task success rates on the 50 RLBench tasks outside RLBench-10. Blue bars show CoA-rev; rose markers show SkiP, connected as a visual guide across discrete tasks. Pink vertical segments highlight SkiP gains over CoA-rev.

Table 2: RoboMimic results with Diffusion Policy UNet backbone in the image observation setting. SR is reported as a fraction. SAIL entries are mean±standard deviation across evaluation runs.

Task	SAIL	CoA-rev	CoA-fwd	SkiP†	SkiP
`lift`	0.820 ± 0.049	0.960 ± 0.016	$\mathbf{1.000 \pm 0.000}$	0.013 ± 0.019	$\mathbf{1.000 \pm 0.000}$
`can`	0.407 ± 0.038	$\mathbf{0.880 \pm 0.016}$	0.873 ± 0.034	0.233 ± 0.009	0.827 ± 0.019
`square`	0.600 ± 0.016	0.420 ± 0.016	0.327 ± 0.025	0.247 ± 0.019	$\mathbf{0.673 \pm 0.062}$
`transport`	0.387 ± 0.062	$\mathbf{0.633 \pm 0.057}$	0.427 ± 0.047	0.000 ± 0.000	0.587 ± 0.041
Avg SR↑	0.553 ± 0.016	0.723 ± 0.013	0.657 ± 0.013	0.123 ± 0.002	$\mathbf{0.772 \pm 0.012}$
Steps$_{\text{succ}}$↓	200.4 ± 4.9	211.8 ± 3.2	211.9 ± 4.2	$\mathbf{88.6 \pm 3.9}$	144.1 ± 2.2

RoboMimic. Table 2 reports results on RoboMimic using a Diffusion Policy UNet backbone in the image observation setting. SkiP achieves the highest average SR, surpasses CoA-rev on the hardest task `square`, and reduces Steps$_{\text{succ}}$ by 32% relative to CoA-rev. We also reimplement SAIL [1] with the same backbone, training data, and evaluation protocol; SkiP achieves 77.2% average SR versus 55.3% for SAIL. Appendix C provides the simulated execution-time comparison.

RoboTwin. Table 3 reports results on 8 bimanual tasks using a DP3 backbone with clean point cloud demonstrations. SkiP achieves the highest average SR, with the VLM variant SkiP† close behind, and substantially reduces Steps$_{\text{succ}}$ relative to CoA-rev and DP3.

Summary. Across three simulation benchmarks, the same training-target modification transfers across DP, DP3, and autoregressive backbones without per-backbone tuning. We also compare MSK with VLM-derived key segments from Gemini-2.5-Pro (SkiP†). MSK provides higher success rates than VLM-derived labels on RLBench and RoboMimic, while VLM-derived labels are competitive on RoboTwin.

4.3 VLA Fine-tuning

We fine-tune $\pi_{0.5}$ [11], a successor to the π_0 vision-language-action flow model [3] trained on the Open X-Embodiment dataset [21], using both standard behavior cloning (**Base**) and SkiP's relabeling objective to check whether the approach works with pretrained foundation models. Since $\pi_{0.5}$ operates in joint space, we discover key segments from end-effector pose trajectories and transfer the resulting partition to joint-space action targets.

Real-robot results. Table 4 reports results on three tabletop tasks with 15 rollouts each, comparing SkiP against three baselines: standard behavior cloning (Base), keyframe-only prediction (KF-only), and Chain-of-Action (CoA). SkiP gets the highest SR on all three tasks and uses fewer executed steps and less wall-clock time per episode. On `stack-bowls`, SR rises from 33.3% (Base) to 53.3% (SkiP) while wall-clock time drops from 2 m 41 s to 2 m 16 s. KF-only is fast on successful runs but rarely succeeds in the open-world setting, with SR consistently below 14%. That SkiP improves a pretrained VLA suggests relabeling is complementary to large-scale pretraining: the pretrained

Table 3: RoboTwin 2.0 results with DP3 backbone in the clean point-cloud setting.

Task	DP3	CoA-fwd	CoA-rev	SkiP†	SkiP
adjust_bottle	0.987 ± 0.009	0.980 ± 0.008	0.980 ± 0.008	**0.997 ± 0.005**	0.993 ± 0.005
beat_block_hammer	0.627 ± 0.050	0.770 ± 0.029	0.817 ± 0.025	**0.857 ± 0.021**	0.833 ± 0.024
handover_block	0.713 ± 0.025	0.840 ± 0.024	0.780 ± 0.029	0.790 ± 0.014	**0.870 ± 0.008**
move_can_pot	0.527 ± 0.012	0.440 ± 0.037	0.393 ± 0.060	0.487 ± 0.078	**0.540 ± 0.014**
open_microwave	0.297 ± 0.076	0.593 ± 0.012	0.683 ± 0.031	**0.860 ± 0.036**	0.830 ± 0.113
place_container_plate	0.767 ± 0.009	0.857 ± 0.012	**0.883 ± 0.019**	0.863 ± 0.005	0.863 ± 0.012
place_empty_cup	0.637 ± 0.019	0.893 ± 0.012	**0.910 ± 0.028**	0.833 ± 0.045	0.810 ± 0.029
place_shoe	0.360 ± 0.024	0.437 ± 0.069	**0.443 ± 0.082**	0.440 ± 0.008	**0.443 ± 0.075**
Avg SR↑	0.614 ± 0.005	0.726 ± 0.013	0.736 ± 0.019	0.766 ± 0.017	**0.773 ± 0.016**
Steps$_{succ}$↓	268.7 ± 4.8	242.7 ± 6.1	176.2 ± 3.5	**121.7 ± 2.3**	126.6 ± 7.3

Table 4: Real-robot $\pi_{0.5}$ fine-tuning results. Time = wall-clock time (min:sec).

	Base	KF-only	CoA	SkiP
SR (%)↑				
pour-water	40.0	6.7	33.3	**46.7**
stack-bowls	33.3	6.7	26.7	**53.3**
tidy-up-desk	66.7	13.3	40.0	**73.3**
Steps↓				
pour-water	290.4	309.6	281.9	**265.4**
stack-bowls	246.3	271.4	226.8	**204.5**
tidy-up-desk	250.7	286.8	232.4	**207.4**
Time↓				
pour-water	3:44	4:01	3:39	**3:28**
stack-bowls	2:41	2:59	2:29	**2:16**
tidy-up-desk	2:20	2:43	2:12	**2:00**

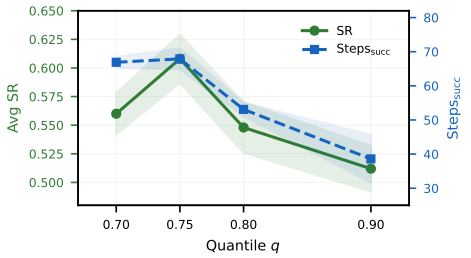


Figure 5: Ablation on quantile threshold q (RLBench-60, 3 eval repeats, shaded bands $= \pm 1$ std). SR peaks at q=0.75 and drops for $q \geq 0.80$; Steps$_{succ}$ decreases monotonically.

weights provide general motor competence, and relabeling teaches the policy where to spend its decision budget.

Simulation counterpart: RLBench-18. We repeat the same protocol on RLBench-18 in simulation. Averaged across 18 tasks (3 seeds each), SkiP improves average SR from 18.59% to 20.96% (+2.37 points) while reducing Steps$_{succ}$ from 108.30 to 66.38 (−39%). Full per-task results are in Appendix H.

4.4 Ablations

Quantile threshold q. The threshold q controls how conservatively key segments are labeled: larger q marks fewer timesteps as refine-worthy. Figure 5 traces SR and Steps$_{succ}$ as q varies from 0.70 to 0.90 on RLBench-60. Among the tested settings, q=0.75 achieves the highest SR, while larger thresholds reduce successful-episode execution steps. Steps$_{succ}$ decreases monotonically with q as fewer timesteps are marked refine-worthy. We select q=0.75 as the default, which achieves the best SR while already providing substantial step reduction.

ST-DCT window size W. We further vary the short-time DCT window W while keeping other hyperparameters fixed (Table 5). W=16 gives the highest SR among the tested settings, exceeding the smaller windows (W=4, 8) by 5–8 points; W=32 produces the lowest Steps$_{succ}$.

Key-segment components. Table 6 ablates keyframe union (u) and bend post-processing (b). Removing both reduces SR from 0.606 to 0.530; the remaining 0.530 with DCT-only labels is still above CoA-rev's 0.490 on RLBench-60.

Label source comparison. To test whether the benefit of action relabeling depends on *how* key segments are identified, we train three alternative segmentation strategies on RLBench-10 with the same architecture and hyperparameters, changing only the label source: *Random Stride* (RS) places key segments at fixed periodic intervals matching the ∼25% key ratio of MSK; *Velocity Only* (VO) labels high-velocity timesteps as key (q=0.75 quantile threshold on raw velocity magnitude); *Low*

Table 5: Ablation on the ST-DCT window size W (RLBench-60).

W	SR↑	Steps↓	$\text{Steps}_{\text{succ}}$ ↓
4	0.529	178.4	50.6
8	0.547	175.6	57.2
16	**0.606**	**165.5**	64.1
32	0.514	176.7	**47.2**

Table 6: Ablation of keyframe union (u) and bend post-processing (b) on RLBench-60.

Variant	u	b	SR↑	Steps↓
w/o u & b			0.530	176.3
w/o union		✓	0.574	168.8
w/o bend	✓		0.574	**163.0**
full	✓	✓	**0.606**	165.5

Table 7: Label source ablation on RLBench-10 (SR↑). Same architecture; only segmentation differs.

Task	RS	VO	LV	MSK (SkiP)
`open-box`	.76 ± .00	.03 ± .02	.76 ± .00	**.91 ± .02**
`open-drawer`	.72 ± .00	.04 ± .00	.96 ± .00	**1.0 ± .00**
`pick-up-cup`	.23 ± .05	.75 ± .04	**.81 ± .08**	.76 ± .06
`press-switch`	.41 ± .05	**.79 ± .04**	.57 ± .05	.53 ± .04
`push-button`	.12 ± .00	.65 ± .02	**1.0 ± .00**	.99 ± .02
`reach-target`	.64 ± .00	**.68 ± .00**	.31 ± .02	**.68 ± .00**
`stack-wine`	.77 ± .02	.85 ± .02	.88 ± .00	**1.0 ± .00**
`sweep-dustpan`	.64 ± .00	.04 ± .00	**1.0 ± .00**	**1.0 ± .00**
`take-lid-off`	.60 ± .00	**.99 ± .02**	.71 ± .02	.97 ± .02
`turn-tap`	.31 ± .04	.64 ± .03	**.73 ± .02**	.67 ± .02
Avg	.520	.545	.773	**.851**

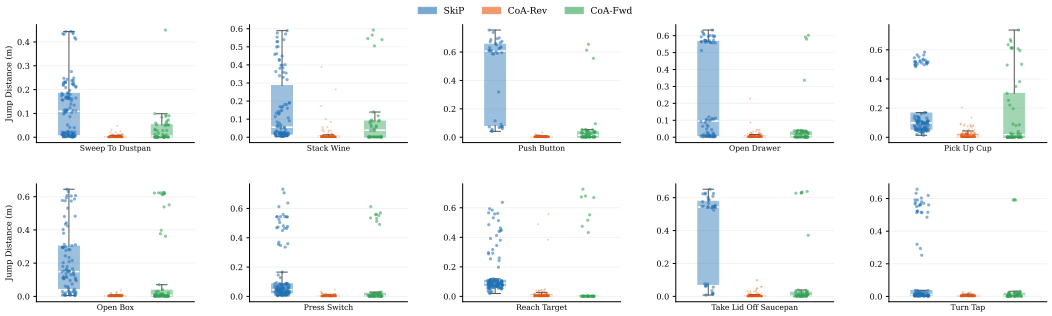


Figure 6: Action displacement distribution per policy call across 10 RLBench tasks. SkiP shows a bimodal pattern: large jumps in skip mode, small adjustments in key mode. CoA-rev concentrates near zero; CoA-fwd shows moderate unimodal displacements.

Velocity Key (LV) inverts this, labeling low-velocity timesteps as key under the intuition that slow motion corresponds to careful manipulation. Table 7 shows that MSK outperforms all alternatives ($+0.078$ over LV, $+0.331$ over RS). MSK's advantage over RS, which matches the key-segment ratio, supports the value of task-relevant segment *placement*. VO achieves SRs of 0.027 on `open-box` and 0.040 on `sweep-dustpan`. LV is the strongest alternative; MSK additionally uses trajectory curvature and frequency-domain cues.

4.5 Analysis: Does SkiP Learn to Skip and Refine?

We measure the *jump distance* $\|a_1 - p_{\text{ee}}\|_2$ per policy call during evaluation, splitting SkiP calls into *key* and *skip* categories by a per-task displacement threshold. Figure 6 shows a clear bimodal pattern across 10 RLBench tasks: skip-mode calls produce jumps of 0.1–0.7 m, while key-mode calls cluster near zero. CoA-rev concentrates near zero; CoA-fwd shows moderate unimodal displacements. This supports the interpretation that relabeling teaches distinct output modes rather than a single averaged behavior.

5 Conclusion

We introduced SkiP, which learns when to skip and when to refine by relabeling behavior cloning targets. Across RLBench, RoboMimic, RoboTwin, and real-robot $\pi_{0.5}$ fine-tuning, SkiP cuts executed steps by 15–40% while matching or improving success rates across various policy architectures. The broader message: temporal resolution in imitation learning can be controlled at the supervision level; changing *what* the policy predicts is sufficient to produce adaptive skip-and-refine behavior.

Acknowledgments

This work was supported in part by the Guangdong S&T Programme under Grant No. 2024B0101010003, the Shenzhen Medical Research Fund under Grant D2404006, and the Major Key Project of PCL under Grant PCL2024A04.

References

[1] Nadun Ranawaka Arachchige, Zhenyang Chen, Wonsuhk Jung, Woo Chul Shin, Rohan Bansal, Pierre Barroso, Yu Hang He, Yingyang Celine Lin, Benjamin Joffe, Shreyas Kousik, and Danfei Xu. Sail: Faster-than-demonstration execution of imitation learning policies, 2025. URL https://arxiv.org/abs/2506.11948.

[2] Yongjie Bai, Zhouxia Wang, Yang Liu, Kaijun Luo, Yifan Wen, Mingtong Dai, Weixing Chen, Ziliang Chen, Lingbo Liu, Guanbin Li, and Liang Lin. Learning to see and act: Task-aware virtual view exploration for robotic manipulation, 2025. URL https://arxiv.org/abs/2508.05186.

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

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S Ryoo, Grecia Salazar, Pannag R Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan H Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.025.

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

[6] Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. Act3d: 3d feature field transformers for multi-task robotic manipulation. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 3949–3965. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/gervet23a.html.

[7] Google DeepMind. Gemini 2.5: Our most intelligent AI model, 2025. URL https://deepmind.google/technologies/gemini/. Technical report.

[8] Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. Rvt: Robotic view transformer for 3d object manipulation. In Jie Tan, Marc Toussaint, and Kourosh Darvish,

editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 694–710. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/goyal23a.html.

[9] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 175–187. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/guhur23a.html.

[10] Yiyang Huang, Yuhui Hao, Bo Yu, Feng Yan, Yuxin Yang, Feng Min, Yinhe Han, Lin Ma, Shaoshan Liu, Qiang Liu, and Yiming Gan. DaDu-Corki: Algorithm-architecture co-design for embodied AI-powered robotic manipulation, 2025. URL https://arxiv.org/abs/2407.04292.

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

[12] Stephen James and Pieter Abbeel. Coarse-to-fine q-attention with learned path ranking, 2022. URL https://arxiv.org/abs/2204.01571.

[13] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2): 3019–3026, apr 2020. doi: 10.1109/LRA.2020.2974707. URL https://doi.org/10.1109/LRA.2020.2974707.

[14] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3D diffuser actor: Policy diffusion with 3D scene representations, 2024. URL https://arxiv.org/abs/2402.10885.

[15] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model, 2024. URL https://arxiv.org/abs/2406.09246.

[16] Longxin Kou, Fei Ni, Yan Zheng, Jinyi Liu, Yifu Yuan, Zibin Dong, and Jianye Hao. KISA: A unified keyframe identifier and skill annotator for long-horizon robotics demonstrations. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 25441–25474. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/kou24b.html.

[17] Fanfan Liu, Feng Yan, Liming Zheng, Chengjian Feng, Yiyang Huang, and Lin Ma. RoboUniView: Visual-language model with unified view representation for robotic manipulation, 2024. URL https://arxiv.org/abs/2406.18977.

[18] 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 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1678–1690. PMLR, 2022.

[19] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. doi: 10.1109/LRA.2022.3180108. URL https://doi.org/10.1109/LRA.2022.3180108.

[20] Yao Mu, Tianxing Chen, Shijia Peng, Zanxin Chen, Zeyu Gao, Yude Zou, Lunkai Lin, Zhiqiang Xie, and Ping Luo. Robotwin: Dual-arm robot benchmark with generative digital twins, 2024. URL https://arxiv.org/abs/2409.02920.

[21] Open X-Embodiment Collaboration et al. Open x-embodiment: Robotic learning datasets and RT-X models, 2023. URL https://arxiv.org/abs/2310.08864.

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

[23] Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/ross10a.html.

[24] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/ross11a.html.

[25] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone, 2022. URL https://arxiv.org/abs/2206.11251.

[26] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. CLIPort: What and where pathways for robotic manipulation, 2021. URL https://arxiv.org/abs/2109.12098.

[27] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 785–799. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/shridhar23a.html.

[28] Yifei Su, Ning Liu, Dong Chen, Zhen Zhao, Kun Wu, Meng Li, Zhiyuan Xu, Zhengping Che, and Jian Tang. Freqpolicy: Efficient flow-based visuomotor policy via frequency consistency, 2025. URL https://arxiv.org/abs/2506.08822.

[29] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2): 181–211, 1999.

[30] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy, 2024. URL https://arxiv.org/abs/2405.12213.

[31] Chuan Wen, Jierui Lin, Jianing Qian, Yang Gao, and Dinesh Jayaraman. Keyframe-focused visual imitation learning, 2021. URL https://arxiv.org/abs/2106.06452.

[32] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2323–2339. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/xian23a.html.

[33] Feng Yan, Fanfan Liu, Liming Zheng, Yufeng Zhong, Yiyang Huang, Zechao Guan, Chengjian Feng, and Lin Ma. RoboTron-Mani: All-in-one multimodal large model for robotic manipulation, 2025. URL https://arxiv.org/abs/2412.07215.

[34] Changchuan Yang, Yuhang Dong, Guanzhong Tian, Haizhou Ge, and Hongrui Zhu. Wavelet policy: Imitation policy learning in the scale domain with wavelet transforms, 2025. URL https://arxiv.org/abs/2504.04991.

[35] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations, 2024. URL https://arxiv.org/abs/2403.03954.

[36] Wenbo Zhang, Tianrun Hu, Yanyuan Qiao, Hanbo Zhang, Yuchu Qin, Yang Li, Jiajun Liu, Tao Kong, Lingqiao Liu, and Xiao Ma. Chain-of-action: Trajectory autoregressive modeling for robotic manipulation, 2025. URL https://arxiv.org/abs/2506.09990.

[37] Xinyu Zhang, Yuhan Liu, Haonan Chang, Liam Schramm, and Abdeslam Boularias. Autoregressive action sequence learning for robotic manipulation, 2025. URL https://arxiv.org/abs/2410.03132.

[38] 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*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016.

[39] Liming Zheng, Feng Yan, Fanfan Liu, Chengjian Feng, Zhuoliang Kang, and Lin Ma. RoboCAS: A benchmark for robotic manipulation in complex object arrangement scenarios, 2024. URL https://arxiv.org/abs/2407.06951.

[40] Yiming Zhong, Yumeng Liu, Chuyang Xiao, Zemin Yang, Youzhuo Wang, Yufei Zhu, Ye Shi, Yujing Sun, Xinge Zhu, and Yuexin Ma. Freqpolicy: Frequency autoregressive visuomotor policy with continuous tokens, 2025. URL https://arxiv.org/abs/2506.01583.

[41] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A Controlled Re-querying Study

This study examines the value of executing the skip behavior learned by SkiP. We keep the trained checkpoint fixed and vary the density of intermediate policy queries within predicted skip intervals. All settings use RLBench-10 and the same evaluation protocol as the main RLBench-10 comparison.

Execution settings. Original SkiP executes its full predicted action chunk. Full-jump re-query executes the first long-range skip target and queries the policy again immediately after reaching it. The H/2, H/4, and 1-step settings split each long-range skip target into 2, 4, and H interpolated waypoints, respectively, and re-query after each waypoint is reached. The intervention applies to detected long-range skip predictions; refine chunks retain their original execution procedure.

Table 8: Controlled re-querying on RLBench-10 with a fixed SkiP checkpoint. Bold denotes the highest success rate and the fewest policy calls.

Execution mode	Average SR (%) ↑	Average policy calls ↓
Original SkiP	85.0	**3.57**
Full-jump re-query	**85.6**	7.69
H/2 re-query	81.2	12.47
H/4 re-query	82.0	17.70
1-step re-query	77.2	56.45

Decision efficiency. Original SkiP attains 85.0% success with 3.57 policy calls per episode. Full-jump re-query yields a similar observed success rate of 85.6% with 7.69 calls. Subdividing the skip into intermediate waypoints increases calls to 12.47–56.45 without improving success over Original SkiP. Executing the learned skip avoids repeated queries along the skipped motion.

B Performance under Matched Decision Budgets

To examine how effectively SkiP allocates policy decisions, we compare it with CoA-fwd on RLBench-10 under the same policy-call budget. CoA-fwd retains dense action supervision without key/skip relabeling. Table 9 reports the full budget sweep.

Table 9: Success rate under matched decision budgets on RLBench-10. Bold denotes the higher success rate at each budget.

Decision budget	SkiP SR (%) ↑	CoA-fwd SR (%) ↑
2	51.6	**66.8**
3	66.4	**68.4**
4	**73.2**	68.4
5	**82.0**	68.4
6	**82.4**	68.4
8	**83.2**	68.4
10	**83.2**	68.4
12	**85.0**	68.4

Adaptive allocation of decisions. With budgets of 4–12 calls, SkiP achieves higher success at every tested budget, reaching 85.0% at 12 calls compared with 68.4% for CoA-fwd. The reported median chunk lengths are 31 for SkiP and 138 for CoA; the longer dense chunks cover more of a trajectory when the budget permits just two or three calls. Once the budget supports decisions across multiple stages, SkiP's learned skip-and-refine allocation provides a stronger success–decision-budget trade-off. The comparison shows that call count alone does not explain SkiP's performance advantage.

C SAIL Execution-Time Comparison

The main RoboMimic comparison (Table 2) includes success rates for our reimplementation of SAIL [1], using the same Diffusion Policy backbone, training data, and evaluation protocol as SkiP. Here we complement those results with simulated execution time over successful episodes, T_{succ}.

SAIL uses controller-command action targets, matching the target representation used in the main comparison. For fixed-frequency policies, simulated execution time is computed from executed control steps at 20 Hz; for SAIL, it is accumulated from the simulator clock under variable control frequencies. Inference time is excluded. Values are mean$\pm$standard deviation across evaluation runs.

Table 10: Successful-episode execution time on RoboMimic, T_{succ} in seconds. Bold denotes the shorter mean time. Success rates appear in Table 2.

Task	SAIL $T_{\mathrm{succ}} \downarrow$	SkiP $T_{\mathrm{succ}} \downarrow$
Lift	$\mathbf{2.571 \pm 0.562}$	2.885 ± 0.153
Can	$\mathbf{3.737 \pm 0.290}$	3.995 ± 0.078
Square	7.018 ± 0.428	$\mathbf{6.860 \pm 0.497}$
Transport	20.426 ± 0.046	$\mathbf{15.076 \pm 0.074}$
Average	8.438 ± 0.160	$\mathbf{7.204 \pm 0.111}$

Execution efficiency. SkiP's average successful-episode execution time is 7.204 s compared with 8.438 s for SAIL. In particular, Transport combines higher success with a shorter execution time, from 20.426 s to 15.076 s. Together with the success rates in Table 2, these results demonstrate the practical value of learning skip-and-refine behavior through supervision: the same policy backbone achieves both higher average success and lower average successful-episode execution time.

D Additional Real-Robot Evaluation

We extend the real-robot evaluation to five additional manipulation tasks: Push Button, Cube Setup, Pick Grape, Basin Loading, and Pick Flower. Table 11 compares the base policy with SkiP using success rate and execution steps over successful episodes, $\mathrm{Steps}_{\mathrm{succ}}$.

Table 11: Five additional real-robot tasks. Bold denotes the higher success rate or fewer successful-episode execution steps; ties are both highlighted.

	SR (%) $\uparrow$		$\mathrm{Steps}_{\mathrm{succ}} \downarrow$	
Task	Base	SkiP	Base	SkiP
Push Button	**88.0**	**88.0**	32.23	**17.86**
Cube Setup	56.0	**76.0**	105.64	**68.42**
Pick Grape	72.0	**84.0**	51.89	**40.19**
Basin Loading	60.0	**72.0**	132.33	**64.89**
Pick Flower	52.0	**68.0**	51.00	**35.88**

Efficient task completion. Across these five tasks, SkiP matches or improves the reported success rate while reducing successful-episode execution steps in every case. Cube Setup improves from 56% to 76% success with steps reduced from 105.64 to 68.42; Basin Loading improves from 60% to 72% with steps reduced from 132.33 to 64.89.

E SkiP Relabeling Pseudocode

Algorithm 1 summarizes the relabeling logic for constructing SkiP training targets as H-step chunks.

Algorithm 1 SkiP training sample construction

Require: Demonstration actions $\{a_t\}_{t=1}^{T}$, `region`[1..T], `next_high`[1..T] (set to $\perp$ if no future key segment), chunk length H
Ensure: Index i, target chunk $\tilde{\mathbf{A}} \in \mathbb{R}^{H\times d}$, mask $m \in \{0,1\}^H$

if `region`$[i]=0$ **and** `next_high`$[i]\neq\perp$ **then**
  $t_{\text{tgt}} \leftarrow$ `next_high`$[i]$ {**skip sample**}
else
  $t_{\text{tgt}} \leftarrow i+1$ {**refine, or skip-tail with no future key segment**}
end if
for $h=1$ **to** H **do**
  if $t_{\text{tgt}}+h-1 \le T$ **then**
    $\tilde{\mathbf{A}}[h] \leftarrow a_{t_{\text{tgt}}+h-1}$; $m[h] \leftarrow 1$
  else
    $\tilde{\mathbf{A}}[h] \leftarrow \mathbf{0}$; $m[h] \leftarrow 0$
  end if
end for

F Reproducibility Details

F.1 Evaluation Protocol

For SkiP on RLBench, each policy query outputs an H-step action chunk, which is executed before the next query; temporal ensembling is disabled. Steps counts the number of executed `env.step` calls until termination. All settings use absolute target actions: RLBench and RoboMimic use absolute end-effector pose targets, while RoboTwin and the $\pi_{0.5}$ setup use absolute joint position targets. For RoboMimic, `tool_hang` is excluded from the main average because all methods obtain 0.00 SR on this task.

F.2 Policy Architecture

SkiP uses the same transformer-based policy architecture as CoA on RLBench (ResNet-18 image encoder, 4-layer encoder / 6-layer decoder transformer, d_{model}=512, 78.4M parameters total). On RoboMimic, we use Diffusion Policy's UNet architecture. On RoboTwin, we use DP3's 3D diffusion architecture. On real-robot and RLBench-18, we fine-tune from the $\pi_{0.5}$ checkpoint.

F.3 Motion Spectrum Keying Details

MSK has two tunable hyperparameters: the DCT window size W=16 (centered, even split $[t-8, t+7]$) and the quantile threshold q=0.75 (per-episode percentile on the high-frequency energy ratio); both are swept in §4.4. The upper half of the DCT spectrum ($k \ge \lceil W/2 \rceil$) defines the high-frequency band, and segments shorter than 3 steps are discarded as noise. The remaining implementation constants are fixed across all experiments without tuning: bend-based augmentation uses a deviation cutoff of 0.30 with $\pm$2-step expansion; heuristic keyframes (gripper-state changes and velocity zero-crossings, following James and Abbeel [12]) are included with a 5-step neighborhood; the first 20% of each episode is excluded to avoid labeling the initial reset motion.

G Efficiency Decomposition

Table 12 decomposes executed steps and policy forward calls on RLBench-10. SkiP achieves the lowest executed steps in this comparison, with 3.57 policy calls per episode. CoA-fwd has the fewest calls (2.08) due to very long chunks, but many more executed steps (160.5). CoA-rev's adaptive

Table 12: Efficiency decomposition on RLBench-10.

Method	Steps↓	# forward calls/ep↓
DP	160.0	8.34
ACT	138.8	7.49
KF-only	<u>113.4</u>	111.94
CoA-fwd	160.5	**2.08**
CoA-rev	125.4	41.36
SkiP	**72.5**	<u>3.57</u>

early-stop mechanism often collapses effective chunk length to 1 step after the first interval, inflating calls to 41.36 per episode.

H RLBench-18 Fine-tuning from $\pi_{0.5}$

Table 13 reports per-task success rate and $\text{Steps}_{\text{succ}}$ when fine-tuning $\pi_{0.5}$ on the RLBench-18 suite. SkiP improves SR from 18.59% to 20.96% on average while reducing $\text{Steps}_{\text{succ}}$ from 108.30 to 66.38 (-39%). Gains are observed on tasks such as `close_jar`, `meat_off_grill`, and `reach_and_drag`.

I Real-Robot Rollout Examples

Figure 7 shows the real-robot setup and rollout examples for the three tabletop tasks. These examples match the tasks used in Table 4.

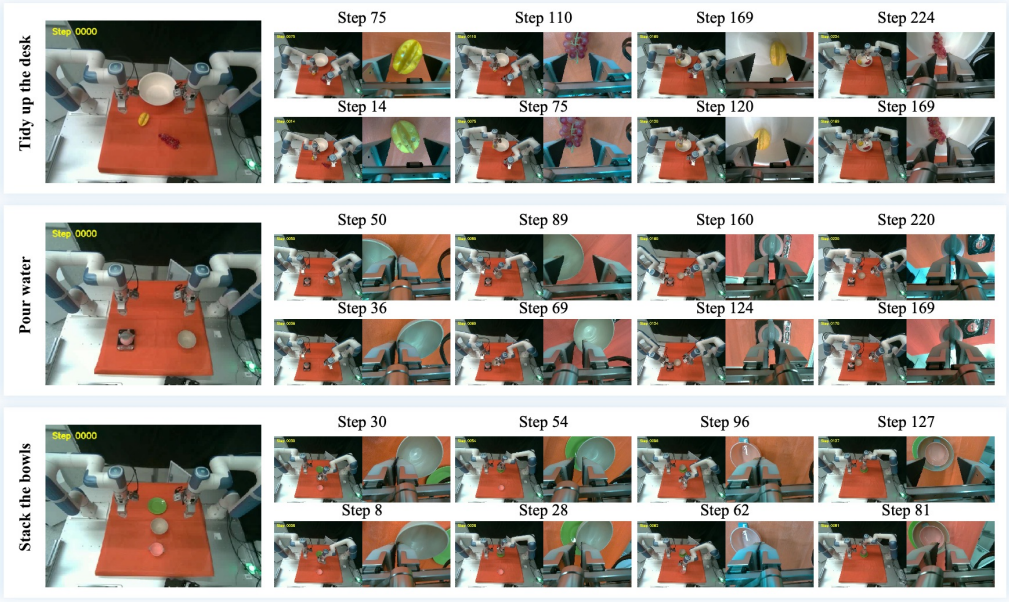


Figure 7: Real-robot rollout examples for `pour-water`, `stack-bowls`, and `tidy-up-desk`.

J RLBench Per-Task Results

We report extended per-task results on the RLBench-60 suite. The main paper's Table 1 already covers RLBench-10 SR and Steps; here we provide $\text{Steps}_{\text{succ}}$ per task and the full RLBench-50 numbers.

Table 13: RLBench-18 fine-tuning from $\pi_{0.5}$ (3 seeds). Bold indicates better performance.

Task	Method	SR↑	$\text{Steps}_{\text{succ}}$ ↓
close_jar	Base	18.67±6.11	189.77±21.21
	SkiP	**42.67±2.31**	**140.49±11.32**
insert_onto_square_peg	Base	0.00±0.00	—
	SkiP	0.00±0.00	—
light_bulb_in	Base	0.00±0.00	—
	SkiP	0.00±0.00	—
meat_off_grill	Base	36.00±6.93	150.61±15.41
	SkiP	**72.00±4.00**	**94.79±2.97**
open_drawer	Base	**48.00±6.93**	101.32±3.53
	SkiP	32.00±8.00	**47.26±10.67**
place_cups	Base	0.00±0.00	—
	SkiP	0.00±0.00	—
place_shape_in_shape_sorter	Base	0.00±0.00	—
	SkiP	0.00±0.00	—
place_wine_at_rack_location	Base	**48.00±13.86**	209.21±1.78
	SkiP	25.33±9.24	**99.94±9.82**
push_buttons	Base	**24.00±4.00**	76.36±1.94
	SkiP	9.33±8.33	**29.50±25.75**
put_groceries_in_cupboard	Base	0.00±0.00	—
	SkiP	0.00±0.00	—
put_item_in_drawer	Base	**25.33±4.62**	457.98±8.88
	SkiP	5.33±6.11	**256.33±245.79**
put_money_in_safe	Base	20.00±10.58	238.81±3.11
	SkiP	**36.00±6.93**	**195.97±18.91**
reach_and_drag	Base	21.33±8.33	182.56±26.19
	SkiP	**49.33±6.11**	**127.49±5.22**
slide_block_to_color_target	Base	18.67±4.62	84.64±0.63
	SkiP	**37.33±2.31**	**47.26±12.71**
stack_blocks	Base	0.00±0.00	—
	SkiP	0.00±0.00	—
stack_cups	Base	0.00±0.00	—
	SkiP	0.00±0.00	—
sweep_to_dustpan_of_size	Base	**48.00±4.00**	122.66±7.08
	SkiP	45.33±2.31	**61.84±4.95**
turn_tap	Base	**26.67±8.33**	135.56±5.37
	SkiP	22.67±2.31	**94.06±11.08**
Overall (avg)	Base	18.59±1.05	108.30±1.63
	SkiP	**20.96±1.51**	**66.38±11.61**

J.1 RLBench-10 Per-Task Steps$_{\text{succ}}$

Table 14 reports per-task Steps$_{\text{succ}}$ (averaged over successful episodes only). KF-only is extremely efficient on pick-and-place-like tasks where it succeeds (e.g., `reach_target`, `push_button`, `take_lid_off`) but fails outright on contact-heavy tasks like `open_box` and `sweep_to_dustpan`, motivating reporting Steps alongside Steps$_{\text{succ}}$.

Table 14: RLBench-10 per-task Steps$_{\text{succ}}$ (average executed steps over successful episodes). Dashes indicate 0 successful episodes for that method on that task.

Task	DP	KF-only	CoA-fwd	ACT	CoA-rev	SkiP
`open_box`	165.2	<u>71.1</u>	163.7	143.1	132.7	**73.8**
`open_drawer`	96.1	**2.0**	105.6	90.8	87.9	<u>39.7</u>
`pick_up_cup`	97.6	**2.1**	117.8	78.4	76.1	<u>24.6</u>
`press_switch`	134.6	**14.0**	176.6	121.7	110.1	<u>40.3</u>
`push_button`	91.3	**8.4**	124.6	99.3	80.4	<u>21.7</u>
`reach_target`	43.7	**1.1**	39.9	38.5	45.4	<u>5.4</u>
`stack_wine`	186.3	**2.1**	201.4	160.9	111.4	<u>77.3</u>
`sweep_to_dustpan`	118.5	**4.0**	118.1	103.9	82.1	<u>59.5</u>
`take_lid_off`	108.2	**3.5**	133.4	89.8	85.1	<u>27.2</u>
`turn_tap`	150.4	**2.4**	145.3	129.5	87.1	<u>61.6</u>
Avg	119.2	**11.1**	132.6	105.6	89.8	<u>43.1</u>

J.2 Per-Task Chunk Length H

Table 15 reports the per-task action-chunk length H used by CoA-fwd/rev (max sub-trajectory length between consecutive keyframes) versus SkiP (max discovered key-segment length). Across RLBench-60, CoA's median H is 138 versus SkiP's 31, a $4.5\times$ gap. This gap explains why CoA-fwd attains low policy-call counts under open-loop execution while SkiP retains shorter chunks to prioritize fine-grained control near key segments.

Table 15: Per-task chunk length H on RLBench-60. CoA's H is the per-task training action-sequence length; SkiP's H is the longest discovered key-segment length under default MSK settings (W=16, q=0.75).

Task	H (CoA)	H (SkiP)	Task	H (CoA)	H (SkiP)
`basketball_in_hoop`	190	25	`open_microwave`	126	43
`beat_the_buzz`	150	27	`open_washing_machine`	149	33
`change_channel`	134	31	`open_wine_bottle`	113	32
`change_clock`	211	35	`phone_on_base`	149	41
`close_box`	211	23	`pick_up_cup`	118	30
`close_drawer`	97	15	`place_hanger_on_rack`	167	34
`close_fridge`	185	38	`play_jenga`	86	12
`close_grill`	108	23	`press_switch`	145	28
`get_ice_from_fridge`	242	44	`take_shoes_out_of_box`	192	18
`hang_frame_on_hanger`	245	28	`toilet_seat_down`	117	27
`hit_ball_with_queue`	210	33	`turn_tap`	119	28
`hockey`	134	26	`water_plants`	156	35
`insert_usb_in_computer`	97	16	*(remaining tasks omitted for space)*		
`lamp_off`	101	31			
`lamp_on`	102	31	**Median (60 tasks)**	**138**	**31**

J.3 RLBench-50 Per-Task Results

Table 17 reports per-task SR and Steps for all 50 remaining tasks. SkiP frequently reduces Steps on tasks with long free-space motion while maintaining competitive SR. Notable examples: on `close_fridge`, SkiP reduces Steps from 105.2 (CoA-rev) to 51.1 while improving SR; on `lamp_off`, SkiP reaches 0.960 SR with 26.7 Steps vs. CoA-rev's 0.880 SR with 78.8 Steps.

Table 17: RLBench-50 per-task SR↑ and Steps↓. Blue saturation: darker = better. Best in **bold**, second <u>underlined</u>.

Method	basketball_in_hoop		beat_the_buzz		change_channel		change_clock		close_box	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.827	159.1	0.253	161.7	0.027	343.8	0.247	256.9	**0.987**	184.4
CoA-r	**0.920**	**89.4**	**0.467**	145.2	0.027	**329.6**	0.267	233.3	0.960	120.4
SkiP	0.867	112.4	0.320	**105.0**	**0.093**	330.9	**0.307**	**219.3**	0.933	**92.2**

Method	close_drawer		close_fridge		close_grill		get_ice_from_fridge		hang_frame_on_hanger	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.920	80.5	0.873	121.0	0.640	122.6	0.033	378.3	0.300	329.9
CoA-r	**1.000**	66.6	0.880	105.2	**0.840**	69.3	0.480	269.9	0.173	329.8
SkiP	0.947	**25.2**	**0.893**	**51.1**	0.800	**67.9**	**0.613**	**263.1**	**0.373**	**281.6**

Method	hit_ball_with_queue		hockey		insert_usb_in_computer		lamp_off		lamp_on	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.000	280.0	0.027	356.4	0.600	167.7	0.773	106.8	0.667	120.0
CoA-r	0.013	276.9	**0.067**	**352.0**	0.000	210.0	0.880	78.8	0.640	108.9
SkiP	**0.093**	**250.1**	0.027	354.3	**0.613**	**105.9**	**0.960**	**26.7**	**0.747**	**68.9**

Method	lift_numbered_block		meat_off_grill		move_hanger		open_door		open_grill	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.007	288.7	0.807	156.3	0.800	179.8	0.953	158.8	0.667	200.2
CoA-r	0.027	284.3	**0.947**	**77.4**	0.000	220.0	0.880	142.2	**0.827**	141.1
SkiP	**0.040**	**279.7**	0.853	102.5	**0.920**	**95.0**	**0.973**	**88.5**	0.800	**123.9**

Method	open_microwave		open_washing_machine		open_wine_bottle		phone_on_base		place_hanger_on_rack	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.393	192.3	0.580	208.2	0.567	146.1	0.293	288.3	0.060	320.3
CoA-r	0.467	170.4	0.760	142.8	0.547	135.4	0.547	206.1	0.000	330.0
SkiP	**0.573**	**144.2**	**0.893**	**69.8**	**0.787**	**116.7**	**0.587**	**199.3**	**0.307**	**261.9**

Method	play_jenga		push_buttons		put_bottle_in_fridge		put_groceries_in_cupboard		put_knife_on_chop_board	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.960	99.0	**0.413**	181.3	0.047	534.7	0.007	418.0	0.087	455.8
CoA-r	**1.000**	71.6	0.333	**123.3**	0.000	540.0	**0.013**	**406.9**	0.093	460.4
SkiP	**1.000**	**29.8**	**0.413**	169.6	**0.267**	**435.8**	**0.013**	416.0	**0.267**	**370.3**

Method	put_money_in_safe		put_plate_in_dish_rack		reach_and_drag		screw_nail		setup_checkers	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	**0.673**	215.6	0.153	461.0	0.740	198.3	0.000	450.0	0.000	640.0
CoA-r	0.080	253.9	0.187	400.9	0.013	329.4	**0.093**	**419.7**	0.000	640.0
SkiP	0.587	**163.6**	**0.480**	**289.4**	**0.827**	**128.7**	0.053	423.3	**0.040**	**595.4**

Method	slide_block_to_target		straighten_rope		take_frame_off_hanger		take_money_out_safe		take_off_weighing_scales	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.307	147.4	0.000	—	0.547	211.9	0.840	189.8	0.027	365.0
CoA-r	0.547	120.1	0.000	—	**0.707**	**164.5**	**0.867**	134.0	**0.093**	**343.8**
SkiP	**0.560**	**96.8**	0.000	—	0.600	176.5	0.840	**99.2**	0.040	361.4

Method	take_plate_off_dish_rack		take_shoes_out_of_box		take_toilet_roll_off_stand		take_umbrella_out		take_usb_out_of_computer	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.733	206.4	**0.273**	**582.9**	0.533	218.5	0.360	152.8	0.880	60.2
CoA-r	0.040	328.7	0.013	653.7	0.667	187.1	**0.560**	125.6	**0.960**	55.9
SkiP	**0.933**	**89.2**	0.000	660.0	**0.800**	**111.2**	0.507	**119.6**	0.933	**29.5**

Method	toilet_seat_down		toilet_seat_up		turn_oven_on		unplug_charger		water_plants	
	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓	SR↑	Steps↓
ACT	0.893	102.9	0.840	204.8	**0.647**	192.0	0.440	143.1	0.387	168.4
CoA-r	0.933	61.8	0.680	177.0	0.533	177.8	**0.720**	109.6	0.480	142.3
SkiP	**1.000**	**40.8**	**0.880**	**99.5**	0.453	**175.4**	0.640	**107.1**	**0.533**	**120.5**

K Limitations and Discussion

Failure modes. Three task settings place distinct demands on temporal resolution: (1) *hard constraint tasks* (`beat_the_buzz`), where contact during a skip jump terminates the episode; (2) *repetitive manipulation* (`take_shoes_out_of_box`), where repeated pick-and-place cycles produce many short segments; and (3) *precision continuous manipulation* (`turn_oven_on`), where smooth rotational motions require step-level precision. In these settings, segment selection must account for geometric constraints, repeated subgoals, and sustained precision alongside motion frequency. Detailed per-task observations are in Appendix N.

Label source comparison. All main results use MSK for segmentation. We compare against alternative label sources in §4.4 (Table 7); MSK exceeds the best alternative (LV) by 0.078 SR.

L Action Displacement Analysis

For each policy call during evaluation, we measure the *jump distance*: $\|a_1 - p_{\text{ee}}\|_2$, the Euclidean distance between the first predicted target position and the current end-effector position. If SkiP learns distinct skip and refine modes, we expect a bimodal distribution: large jumps when the policy skips past free space, and small adjustments when it refines near contacts.

Figure 6 (main paper) shows this distribution across 10 RLBench tasks. For SkiP, we split policy calls into *key* and *skip* categories using a per-task displacement threshold derived from training-demo statistics. SkiP exhibits a clear bimodal pattern: skip-mode calls produce jumps of 0.1–0.7 m, while key-mode calls cluster near zero. In contrast, CoA-rev concentrates near zero across all tasks, and CoA-fwd (open-loop long chunks) shows moderate but unimodal displacements. This supports the interpretation that relabeling teaches the policy to produce qualitatively different outputs depending on the trajectory phase, rather than a single averaged behavior.

M Supplementary Ablation Details

Key-segment component contribution. The 0.076-point SR gap between the full MSK variant (0.606) and DCT-only (0.530) on RLBench-60 quantifies how much of SkiP's performance is attributable to the specific labeling stack (heuristic keyframe union and bend-based augmentation). Each component contributes independently, and the full variant achieves the best SR. Notably, even the DCT-only variant (0.530) is comfortably above CoA-rev's 0.490 on the same suite, confirming that the relabeling mechanism itself, not the labeling refinements, accounts for the majority of the gain.

Cross-representation transfer. MSK transfers across action representations: on RLBench-60, joint-space DCT achieves 0.568 SR versus 0.574 for end-effector DCT (both without bend, since bend requires end-effector coordinates). The negligible gap (0.006) indicates that frequency content carries similar task-relevant information regardless of the action space, and practitioners can apply MSK to whichever representation is available.

Label source analysis. Velocity Only (VO) achieves SRs of 0.027 on `open-box` and 0.040 on `sweep-dustpan`, providing a comparison with segmentation based on velocity magnitude alone. Low Velocity Key (LV, 0.773) is the strongest alternative; MSK additionally captures trajectory curvature and frequency-domain patterns. Random Stride (RS, 0.520) matches the $\sim 25\%$ key-segment ratio. MSK's advantage supports selecting task-relevant segment *locations* rather than matching segment *count* alone.

Absolute-target requirement (expanded). SkiP assumes the controller can reach a distant waypoint in one step, which holds when actions are absolute poses or joint angles. All four benchmarks satisfy this: motion planning on RLBench, absolute end-effector pose targets on RoboMimic, absolute joint targets on RoboTwin and $\pi_{0.5}$. Under delta-action or velocity control, skip targets would need to be reformulated as accumulated deltas over the skipped span, and large gaps may introduce tracking errors that offset the benefit of skipping. Extending SkiP to delta-action settings is a natural direction for future work.

N Failure Case Study Details

We examine the three task settings identified in §K through quantitative segment statistics and task-level observations.

Setup. All evaluations use the main SkiP configuration (W=16, q=0.75, $L_{\min}$=3). Segment statistics are computed over 20 training demonstrations per task. Each task is evaluated with 3 repeats $\times$ 25 episodes.

Table 18: DCT segment statistics for three failure tasks vs. two representative success tasks.

Task	Ep. Length	Segments	Avg Seg Len	Key Ratio
`beat_the_buzz`	138.1	4.9	7.5	26.7%
`take_shoes_out_of_box`	465.2	20.0	6.2	26.6%
`turn_oven_on`	132.4	4.3	10.1	31.0%
`open_door` (success)	∼90	3–4	∼25	∼30%
`lamp_off` (success)	∼70	2–3	∼30	∼28%

(1) Hard constraint violation: `beat_the_buzz`**.** The task requires sliding a wand along a curved wire from left to right; touching the wire triggers immediate episode termination. The trajectory is geometrically smooth (low DCT spectral energy), so MSK labels most of the wire-following phase as a skip segment. However, the wand must maintain sub-centimeter clearance from the wire at every step. When SkiP issues a skip action that jumps several steps ahead, the resulting straight-line interpolation can clip the wire, terminating the episode. CoA-rev, which replans at every step, maintains the fine-grained corrections needed to avoid contact (SR 0.467 vs. SkiP 0.320).

(2) Over-fragmented segmentation: `take_shoes_out_of_box`**.** This task requires extracting two shoes from a box via 4+ pick-and-place cycles, producing the longest episodes in RLBench-60 (465 steps). The repetitive grasp-lift-move-release pattern creates alternating frequency signatures, yielding 20 segments averaging only 6.2 steps each. Under this segmentation, all 75 evaluation episodes time out at the 660-step budget. ACT, which uses uniform action chunking, achieves 27.3% SR on this task.

(3) Precision continuous manipulation: `turn_oven_on`**.** Rotating a knob through 86° (1.5 rad) requires the end-effector to maintain contact and apply consistent rotational force. The segmentation is well-behaved (4.3 segments, 10.1 avg length), so failure is not due to fragmentation. Rather, the rotation phase is smooth and repetitive (low frequency), causing MSK to label it as skippable. Skipping within this phase causes loss of contact or angular overshoot. When SkiP fails, episodes consistently time out at 280 steps, indicating the policy gets close to but cannot precisely reach the 1.5 rad threshold.

Common thread and future directions. These cases involve geometric constraints, repeated subgoals, or sustained precision requirements. They motivate incorporating task-specific structure into temporal allocation.

Potential mitigations include: (1) *constraint-aware skipping* that incorporates task-level safety boundaries into the skip decision; (2) *adaptive segment merging* that detects and consolidates over-fragmented patterns in repetitive tasks; and (3) *learned skip confidence* that attenuates skip magnitude during high-precision manipulation phases.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

Justification: The abstract and §1 state the action-relabeling contribution and its success–efficiency benefits. Sections 4.2–4.5 and the supplementary experiments provide the corresponding comparisons and analyses.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix K discusses the method's limitations, with detailed task-level analysis in Appendix N.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper presents an empirical learning method and does not claim formal theoretical results. The learning objective and algorithm are specified in §3.2 and Appendix E.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sections 3.2–3.3 specify the learning objective and labeling procedure. Section 4.1 and Appendix F describe the benchmarks, architectures, and evaluation protocol. Appendix E provides the relabeling algorithm, and the project page links to the code.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The project page, `https://pgq18.github.io/SkiP-page/`, links to the public code repository with environment setup, RLBench data preparation, training and evaluation commands, and RLBench-10 checkpoints. The simulation benchmarks RLBench [13], RoboMimic [18], and RoboTwin [20] are publicly available; the paper describes the corresponding experimental settings.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 describes the benchmarks, baselines, and metrics. Appendix F gives policy and evaluation details, and Appendix F.3 specifies the labeling parameters. Section 4.4 reports hyperparameter and component ablations.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The simulation result tables report means and standard deviations, including Table 1 and Appendix H. The SAIL comparison in Table 2 and Appendix C reports mean$\pm$standard deviation across evaluation runs. Additional real-robot results appear in Appendix D.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The experiments use an NVIDIA RTX 5880 Ada GPU with 48 GB memory. The simulation training configuration uses 20,000 gradient steps and batch size 128 per task; a method–task run takes approximately 1–2 hours for training and 20–30 minutes for evaluation.

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The research uses public manipulation benchmarks and a tabletop robot setup. It does not involve human subjects or newly scraped data. The authors have reviewed the NeurIPS Code of Ethics.

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: The paper focuses on a general method for efficient robot manipulation and its experimental evaluation. It does not include a separate discussion of positive and negative societal impacts.

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The introduced asset is a robot manipulation training and evaluation framework; it does not introduce a high-risk generative model or scraped dataset requiring release safeguards.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the benchmarks and policy backbones used in the experiments, including RLBench [13], RoboMimic [18], RoboTwin [20], Diffusion Policy [5], ACT [38], Chain-of-Action [36], DP3 [35], and $\pi_{0.5}$ [11]. The code builds on these assets under their respective licenses.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The project page links to the SkiP repository, released under Apache-2.0. Its README documents environment setup, RLBench data preparation, training and evaluation commands, and available RLBench-10 checkpoints. The paper provides the method specification, pseudocode, and experimental settings.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The research does not involve human subjects.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Section 4.2 and Table 1 describe the SkiP† comparison using Gemini-2.5-Pro for segment labeling. Section 4.3 describes the pretrained $\pi_{0.5}$ vision-language-action backbone. The core action-relabeling method does not require an LLM.