EXRL: SAMPLE-EFFICIENT ONLINE ADAPTATION OF LONG-CHUNK POLICIES VIA EXECUTION LENGTH CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Predicting long action chunks, often spanning dozens of timesteps, has become a key ingredient in increasingly capable robot policies. In this paper, we find that adapting these long-chunk policies with existing reinforcement learning (RL) methods can require many online samples due to the high dimensionality of the action chunks being optimized, making such adaptation impractical in real-world scenarios where interaction data are costly to collect. Our key insight is to shift adaptation from optimizing actions to optimizing how long to execute each predicted action chunk. This keeps the adaptation space low-dimensional while its expressivity grows with the chunk size. We thus introduce *Execution at the Right Length* (ExRL), a simple approach which augments the base policy with a lightweight adapter that predicts the execution length of each chunk and is trained using standard off-policy RL algorithms. Across simulation benchmarks and real-world robotic tasks, ExRL enables sample-efficient adaptation of a diverse range of long-chunk policies with minimal task-specific hyperparameter tuning. See videos and code at `exrl-anonymous.pyrite-apogee.workers.dev`.

1 INTRODUCTION

Robot policies trained on demonstration datasets have enabled increasingly complex and generalizable behaviors across a wide range of robotic control tasks (Sunday Robotics, 2026; Ai et al., 2026). A key technique shared by many of these policies is action chunking (Zhao et al., 2023). Rather than predicting a single action at each query, action-chunked policies predict a sequence of actions and execute them open-loop, which can help capture non-Markovian structure in demonstrations (Lazzati et al., 2026; Liu et al., 2025) and reduce compounding errors (Lazzati et al., 2026; Zhang et al., 2025). A common choice in recent robot policies is to predict *long action chunks*, often exceeding 20 steps and approaching 50, as in vision-language-action (VLA) models such as $\pi_{0.5}$ (Black et al., 2025) and π_0-FAST (Pertsch et al., 2025), and world action models such as DreamZero (Ye et al., 2026). By producing many actions per model query, long chunks enable high-frequency control (e.g., 50 Hz) for dexterous tasks such as laundry folding (Black et al., 2024), while reducing inference pauses for smoother, faster execution.

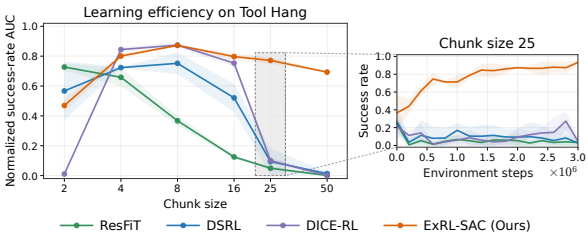


Figure 1: **Prior RL methods are sample-inefficient when adapting diffusion policy with long action chunks.** *Left:* Normalized success-rate AUC over the first 3M environment steps versus chunk size on the `Robomimic Tool Hang` task. *Right:* RL learning curves for chunk size 25.

These long-chunk robot policies, while performant, are rarely optimal when deployed in the real world due to limited demonstration coverage and imperfect generalization (Simchowitz et al., 2025). Reinforcement learning (RL) provides a direct way to adapt these policies from online experience. In other domains such as language modeling, online RL has been shown to be quite effective for adapting models pretrained on offline datasets (Ouyang et al., 2022; Jaech et al., 2024; Guo et al.,

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.

2025). For robotic control, however, online data collection in the real world is often slow, costly, and sometimes unsafe, making sample efficiency a central requirement.

Yet we observe that long action chunks introduce a major bottleneck for sample-efficient adaptation (Figure 1). RL methods that optimize entire action chunks or their latent variables face an increasingly high-dimensional optimization space as the chunk length grows. This substantially increases the amount of online experience required for adaptation.

This motivates us to seek an adaptation interface whose dimensionality does not grow with the chunk length. We observe that action-chunked policies naturally expose such an interface, namely, how long to execute a predicted chunk before querying the policy again. Although execution length is a simple scalar decision, adapting it to the current state and predicted action chunk can substantially alter policy behavior by controlling the trade-off between longer open-loop execution, which better exploits the benefits of action chunking, and shorter execution, which improves responsiveness to unexpected state changes or dynamics mismatch (Liu et al., 2025). This makes execution length a meaningful variable for policy improvement, while being particularly attractive for long chunks: as the chunk size grows, the number of possible execution lengths—and hence the expressivity—increases, whereas the optimization itself remains a low-dimensional discrete problem.

We instantiate this idea as *Execution at the Right Length* (ExRL), a sample-efficient online RL method for adapting long-chunk policies. It adds a lightweight execution-length policy that selects how many actions from each predicted chunk to execute before replanning, and trains it using standard off-policy RL algorithms. Because this adapter can be learned without modifying the base policy itself, ExRL provides a policy-agnostic interface that can be applied to virtually any class of action-chunked policies. Execution-length control is ultimately bounded by the capabilities of the underlying base policy. To go beyond this limitation, we can further optionally distill successful online trajectories back into the base policy during RL using simple imitation loss, allowing the base policy itself to improve alongside the adapter. Finally, ExRL can also leverage general offline data, such as human demonstrations, by reinterpreting arbitrary trajectory segments as off-policy experience under different execution lengths, further improving online sample efficiency.

We empirically demonstrate the effectiveness of ExRL across simulated and real-world manipulation tasks, showing state-of-the-art online adaptation performance across a range of long-chunk policies, largely without task-specific hyperparameter tuning. These include standard action-chunked policies, different classes of pretrained VLA models such as $\pi_{0.5}$ (Black et al., 2025) and π_0-FAST (Pertsch et al., 2025), as well as a world action model, Cosmos Policy (Kim et al., 2026). On challenging real-world tasks, including precise manipulation and long-horizon tasks, ExRL enables sample-efficient adaptation of the $\pi_{0.5}$ policy within a few hundred online trials, improving average task progress from below 50% to above 80%.

2 Related Work

Imitation learning in robotics. Imitation learning, particularly behavior cloning, has been widely used to train data-driven policies for robotic control (Atkeson & Schaal, 1997; Ross et al., 2011; Zhang et al., 2018; Florence et al., 2022; Shafiullah et al., 2022). Prior work has explored a wide range of action parameterizations and policy architectures for imitation learning. Earlier methods often represent actions with Gaussian or GMM policies (Khansari-Zadeh & Billard, 2011; Mandlekar et al., 2021), while more recent approaches use expressive generative modeling techniques, including VAEs (Zhao et al., 2023), diffusion or flow models (Chi et al., 2025; Octo Model Team et al., 2024; Black et al., 2024; Bjorck et al., 2025), and autoregressive next-token prediction models (Zitkovich et al., 2023; Kim et al., 2024; Pertsch et al., 2025). With the growing scale of human demonstration datasets and policy models (Walke et al., 2023; Khazatsky et al., 2024; Brohan et al., 2022; O'Neill et al., 2024), imitation learning has further become a pretraining paradigm for generalist robot policies, absorbing broad prior knowledge into a single model that can be adapted for real-world deployment (Octo Model Team et al., 2024; Kim et al., 2024; Black et al., 2024). Our work provides an efficient method to adapt these diverse policy classes from online experience.

Adaptive execution of action chunks. Action chunking, as used in ACT (Zhao et al., 2023), has become a de facto design choice in robot policies learned with imitation learning. Recognizing that the execution length of action chunks strongly affects policy behavior (Shukor et al., 2025; Zhang

et al., 2025; Liu et al., 2025), several concurrent works have explored adaptive execution strategies. Earlier RL methods such as Dynamic Action Repetition (Lakshminarayanan et al., 2017) and TempoRL (Biedenkapp et al., 2021) adapt how long a primitive action is repeated, rather than how much of a predicted action chunk is executed. AAC (Liang et al., 2026) selects the execution length of vision-language-action models using action entropy, while FFDC (Wang et al., 2026) selects the execution length for world action models using a learned verifier to check whether predicted future observations remain consistent with the real rollout. ExRL instead learns execution-length decisions directly from task rewards through online RL. TAS (Weng et al., 2025) uses online RL to select among actions cached from overlapping chunks at every timestep, rather than explicitly learning a chunk-level execution length. Two other recent works, ACSAC (Chen et al., 2026) and ACH (Shin et al., 2026), learn value-based adaptive chunk-size selection for offline-to-online RL. However, these methods train and evaluate specific chunked policies within offline and offline-to-online RL frameworks, and are not designed as plug-in adapters for arbitrary pretrained action-chunked policies. The closest work to ours is Zhao et al. (2026), a concurrent study that also adapts action-chunked policies through reinforcement learning over execution length. However, it relies on PPO for optimization; our comparison in Appendix B shows that off-policy learning can adapt execution lengths with substantially fewer online samples.

Reinforcement learning for robot policy adaptation. On-policy methods such as PPO and GRPO have been used to fine-tune pretrained robot policies (Li et al., 2025a; Lu et al., 2025; Chen et al., 2025; Ren et al., 2025), but often require many online rollouts, limiting their practicality on real robots. Off-policy RL algorithms such as SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) can use data more efficiently, but their standard implementations typically use deterministic or Gaussian actors, making direct adaptation of more complex generative policies nontrivial. Policy-specific approaches address this challenge in different ways: DSRL (Wagenmaker et al., 2025) and ZPRL (Yu et al., 2026) optimize diffusion/flow policies in latent spaces. OGPO (Patil et al., 2026) uses off-policy critics to guide optimization of the diffusion or flow generation process, while RL-100 (Lei et al., 2025) applies PPO-style updates across diffusion denoising steps. These methods can improve sample efficiency, but remain tied to particular policy classes. Residual RL instead learns additive corrections to the base actions (Yuan et al., 2024; Ankile et al., 2025b;a; Dong et al., 2025a; Sun & Song, 2026). While policy-agnostic and lightweight, residual RL still requires continuous action optimization; correcting an entire chunk makes this space grow with chunk length. RL Token (Xu et al., 2026) learns a lightweight actor conditioned on VLA representations and reference chunks, with regularization toward the base actions. PA-RL (Mark et al., 2024) achieves policy-agnostic adaptation by casting RL as action optimization followed by supervised policy learning. RECAP (Amin et al., 2025) builds an offline-to-online pipeline for training advantage-conditioned policies, which can incorporate batch online RL (Dong et al., 2025b) for efficient adaptation. In contrast to these action-modifying approaches, ExRL adapts the execution length of each predicted chunk, reducing online adaptation to a compact discrete decision space and enabling sample-efficient, policy-agnostic adaptation.

3 Preliminaries

Markov decision processes. Markov decision process (MDP) is denoted by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(\cdot \mid s, a)$ is the transition dynamics, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, ρ_0 is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a mapping from states to distributions over actions. The Q-function, or critic, of a policy π is defined as $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{t \geq 0} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a]$, which is the expected discounted return of π from taking action a in state s. The goal of reinforcement learning is to find a policy that maximizes the expected discounted return.

Q-learning. A broad class of RL algorithms learns a parameterized Q-function $Q_\phi(s_t, a_t)$ by bootstrapping from Bellman targets. Given a transition (s_t, a_t, r_t, s_{t+1}), a generic one-step temporal-difference (TD) target takes the form

$$y_t = r_t + \gamma V(s_{t+1}), \tag{1}$$

where V denotes the bootstrap value of the next decision state induced by the particular algorithm. The critic is then trained by minimizing the squared TD error

$$\min_\phi \ \mathbb{E}\left[(Q_\phi(s_t, a_t) - y_t)^2\right]. \tag{2}$$

Action chunking. An action-chunked policy π is a mapping $\mathcal{S} \to \Delta(\mathcal{A}^H)$ from states to distributions over H-step action sequences. At a query state s_t, a standard action-chunked policy samples an action chunk $a_{t:t+H} = [a_t, a_{t+1}, ..., a_{t+H-1}] \sim \pi(\cdot \mid s_t)$, and then executes $a_{t:t+k}$, where $1 \leq k \leq H$ is a fixed execution length, after which the policy is queried again at state s_{t+k}.

Problem setting. In this paper, we assume access to a pretrained action-chunked policy π_{base}, and our goal is to adapt it to maximize the expected return specified by some reward r using online interactions with an environment $\mathcal{M}$. Note that we make no assumptions on the parameterization or architecture of π_{base}.

4 Execution at the Right Length

We propose ExRL, which adapts action-chunked policies by dynamically controlling how long each predicted action chunk is executed before replanning, rather than modifying the action chunk itself. In this section, we formalize this idea as reinforcement learning over execution-length decisions.

Given a pretrained action-chunked policy π_{base} with chunk size H, we learn an additional execution-length policy $\pi_{\text{ex}} : \mathcal{S} \times \mathcal{A}^H \to \Delta(\mathcal{K})$, where $\mathcal{K} = \{1, \dots, H\}$. At each query, π_{base} first predicts an action chunk, and π_{ex} maps the current state and this predicted chunk to a distribution over feasible execution lengths. In this way, the raw primitive actions are still produced by π_{base}, while the learned execution-length policy acts in the much smaller discrete space $\mathcal{K}$, avoiding direct optimization in the continuous, high-dimensional space of action chunks. We can also optionally allow the policy to choose $k = 0$, corresponding to executing none of the actions in the current chunk, which is essentially an immediate replanning decision.

Duration-aware Q-learning. The execution-length decision naturally leads to a temporally extended Q-function. At a query timestep t, let $a_{t:t+H}$ denote the action chunk sampled from $\pi_{\text{base}}(\cdot \mid s_t)$. In ExRL, we learn a critic $Q_\phi(s_t, a_{t:t+H}, k)$, which estimates the return obtained by executing the first k actions of the predicted chunk and then continuing with the current execution-length strategy. This is closely related to Q-chunking (Li et al., 2025b), which learns a critic over an action sequence rather than a single primitive action. The difference is that in Q-chunking, the critic evaluates the action chunks, while the critic in ExRL evaluates the selected prefix lengths of action chunks. Thus, the accumulated reward and the bootstrap discount in ExRL both depend on the chosen "meta-action" k_t, leading to the duration-aware TD target below:

$$y_t = \sum_{i=0}^{k_t-1} \gamma^i r(s_{t+i}, a_{t+i}) + \gamma^{k_t} V(s', a'), \tag{3}$$

where $s' = s_{t+k_t}$ is the next query state, $a' = a_{t+k_t:t+k_t+H} \sim \pi_{\text{base}}(\cdot \mid s')$ is the next chunk sampled from the frozen base policy, and V is the next-state value. This duration-aware backup mirrors the Bellman backup used in semi-Markov decision processes (SMDPs) (Sutton et al., 1999), where temporally extended decisions may last for a variable number of primitive timesteps. In Appendix E, we formalize this connection by showing how execution-length control induces an SMDP under the frozen base policy π_{base}.

Practical instantiations. ExRL is agnostic to the choice of RL algorithm, as long as the algorithm can optimize over the discrete execution-length space with the duration-aware backup above. In practice, we instantiate ExRL with two off-policy variants for sample-efficient online adaptation. The first variant, **ExRL-Q**, utilizes the discrete nature of the action space and uses value-based RL without learning an explicit actor (Watkins & Dayan, 1992; Mnih et al., 2013). In this case, the next-state value V is computed by a Bellman optimality backup:

$$V(s', a') = \max_{k' \in \mathcal{K}} Q_{\bar{\phi}}(s', a', k'), \tag{4}$$

where $Q_{\bar{\phi}}$ is a slowly-updated target, and the critic is trained by minimizing the squared TD error:

$$\mathcal{L}_Q(\phi) = \mathbb{E}\left[\left(Q_\phi(s_t, a_{t:t+H}, k_t) - y_t\right)^2\right]. \tag{5}$$

During online rollout, the execution length is selected greedily as

$$k_t = \arg\max_{k \in \mathcal{K}} Q_\phi(s_t, a_{t:t+H}, k). \tag{6}$$

The second variant, **ExRL-SAC**, adds an explicit actor $\pi_\psi(k_t \mid s_t, a_{t:t+H})$ that outputs a categorical distribution over execution lengths. Since $\mathcal{K}$ is discrete, we use discrete SAC (Christodoulou, 2019) as an actor-critic instantiation. Compared with continuous-action SAC, the discrete action space allows us to compute the policy value exactly by summing over all $k \in \mathcal{K}$. In this case, the hard maximum in ExRL-Q is replaced by the soft value under the actor. The soft value is

$$V(s', a') = \sum_{k' \in \mathcal{K}} \pi_\psi(k' \mid s', a') \left[Q_{\bar{\phi}}(s', a', k') - \alpha \log \pi_\psi(k' \mid s', a') \right]. \tag{7}$$

The entropy term $-\alpha \log \pi_\psi$ encourages the policy to maintain uncertainty over execution lengths, which improves exploration and prevents early collapse to a single fixed k. Squared TD error is used to update the critic, and the actor is trained to choose execution length with high soft value:

$$\mathcal{L}_\pi(\psi) = -\mathbb{E}_{k_t \sim \pi_\psi(\cdot \mid s_t, a_{t:t+H})} \left[Q_\phi(s_t, a_{t:t+H}, k_t) - \alpha \log \pi_\psi(k_t \mid s_t, a_{t:t+H}) \right]. \tag{8}$$

Both ExRL-Q and ExRL-SAC further use conservative Q-learning (CQL) (Kumar et al., 2020) to reduce overestimation of execution lengths with limited support in the replay data. Specifically, we augment the TD objective with

$$\mathcal{L}_{\text{critic}}(\phi) = \mathcal{L}_Q(\phi) + \lambda_{\text{CQL}} \mathbb{E}_{(s,a,k) \sim \mathcal{D}} \left[\log \left(\frac{1}{|\mathcal{K}|} \sum_{k' \in \mathcal{K}} \exp Q_\phi(s, a, k') \right) - Q_\phi(s, a, k) \right], \tag{9}$$

where $\mathcal{D}$ is the replay buffer and λ_{CQL} controls the regularization strength. The algorithms of ExRL-Q and ExRL-SAC are summarized in Algorithm 1 and 2, respectively.

ExRL + online behavior cloning (BC). Execution-length adaptation can improve closed-loop behavior without changing the actions proposed within each chunk, but its capability is ultimately bounded by the underlying base policy. Importantly, by replanning at adaptive times, ExRL can stitch together prefixes from different predicted chunks and produce successful trajectories that would not arise from simply executing the original long chunks open-loop. We can therefore distill these improved behaviors back into the base policy during RL. Specifically, from each successful ExRL rollout, we construct H-step action-chunk targets from the primitive actions actually executed, even when a target spans multiple replanning decisions, and update π_{base} using the same behavior cloning objective as in pretraining. In this way, behaviors discovered through execution-length adaptation become direct supervision for the base policy, improving its chunk predictions and in turn providing a stronger basis for further adaptation.

Leveraging offline data. ExRL can also reuse general offline trajectories, such as human demonstrations, even if they were collected without action chunking or execution-length decisions. The key observation is that any sequence of recorded primitive actions can be retrospectively treated as a virtual action chunk. Its prefixes of different lengths then provide real experience of what happens when the same action sequence is executed for different numbers of steps before replanning. Importantly, ExRL does not attempt to imitate how the offline behavior chose these actions or when it replanned. The offline trajectory only provides the rewards and resulting state for each prefix; from that state onward, the value target bootstraps with the current base policy and execution-length strategy. This allows each offline trajectory segment to be relabeled into off-policy supervision for multiple execution lengths, improving sample efficiency without additional environment interaction. Concretely, before online training, we enumerate trajectory segments of all lengths $k \leq H$ and add the corresponding execution-length transitions to the replay buffer. When online behavior cloning is enabled, demonstration data can also be mixed with successful online trajectories to jointly update the base policy.

Remark. One may ask why we do not simply use a short fixed execution length for previous RL methods to avoid high-dimensional optimization, or first select the optimal fixed execution length and then optimize on top of it. Always using a short execution length for RL, however, can be problematic for several reasons: it may yield very poor initial performance on some tasks, making RL optimization difficult; it requires more frequent policy queries, increasing training cost and inference overhead while potentially introducing inference–execution synchronization issues; and, on long-horizon tasks, it substantially increases the effective decision horizon, further complicating RL. Alternatively, preselecting an optimal fixed execution length instead requires extensive real-world evaluation, potentially hundreds of rollouts, which is already comparable to the interaction budget of a typical real-world online RL experiment.

5 EXPERIMENTS

The goal of our experiments is to answer the following core questions:

- **(Q1)** Can ExRL enable sample-efficient adaptation across action-chunked policies, including generalist robot policies in both simulation and real-world tasks?
- **(Q2)** Can updating the base policy with successful online trajectories and demonstrations further improve adaptation?
- **(Q3)** What execution-length patterns does ExRL learn?

5.1 BASE POLICIES

To show the generality of our method, we conduct experiments on diverse classes of action-chunked policies. We consider both representative action-chunked policies trained with imitation learning and several pretrained generalist robot policies of different classes.

Representative action-chunked policies: (1) **Diffusion Policy (DP)** (Chi et al., 2025): an action-chunked policy that models multimodal continuous action chunks using denoising diffusion (Ho et al., 2020); and (2) **ACT** (Zhao et al., 2023): a transformer-based policy that predicts action chunks using an encoder–decoder architecture.

Generalist robot policies. (1) π_0 (Black et al., 2024) and $\pi_{0.5}$ (Black et al., 2025): state-of-the-art VLA models that use flow matching (Lipman et al., 2022) to model action chunks; (2) π_0-**FAST** (Pertsch et al., 2025): an autoregressive VLA model that generates action chunks through efficient action tokenization; and (3) **Cosmos Policy** (Kim et al., 2026): a world action model built on a large pretrained video generation model.

5.2 BASELINES AND SETUP

We compare ExRL against execution-length baselines and several online adaptation baselines, with a focus on policy-agnostic methods for action-chunked policies. Our baselines include:

Execution-length baselines. (1) **Random-k**: randomly samples an execution length from the feasible set at each policy query; (2) **Optimal fixed-k**: sweeps fixed execution lengths and reports the best-performing one.
AAC (Liang et al., 2026): A training-free adaptive execution method that selects the execution length using action entropy.
Online BC: An ablation that directly distills successful online rollouts back into the base policy using behavior cloning without RL.
Policy Decorator (Yuan et al., 2024): An off-policy residual RL method that freezes the base policy and learns additive corrections to its predicted actions, with bounded residuals and controlled exploration for stable online learning.
DSRL (Wagenmaker et al., 2025): A latent steering method for diffusion- and flow-based policies that keeps the base policy frozen and applies RL in its initial latent-noise space rather than directly modifying the generated actions.
PA-RL (Mark et al., 2024): A policy-agnostic RL finetuning method that uses a learned critic to improve action candidates through re-ranking and local optimization, and then distills the improved actions back into the base policy.
OGPO (Patil et al., 2026): An off-policy full-finetuning method for diffusion- and flow-based policies. It learns off-policy critics for data reuse and uses their values to optimize the full diffusion or flow generation process through a modified PPO objective.
ResFiT (Ankile et al., 2025a): A demo-augmented residual RL method that keeps the base policy frozen, constrains residual magnitude for stable exploration, and jointly trains on demonstration and online data; we adapt its original per-step residual formulation to operate over action chunks.
DICE-RL (Sun & Song, 2026): Also a demo-augmented residual RL method that combines value-guided action selection with selective behavior regularization to reinforce high-value behaviors while remaining close to the pretrained policy.

Simulation benchmarks and setup. We consider three simulation benchmarks: `Robomimic` (Mandlekar et al., 2021), spanning five tabletop manipulation tasks from simple tasks to challeng-

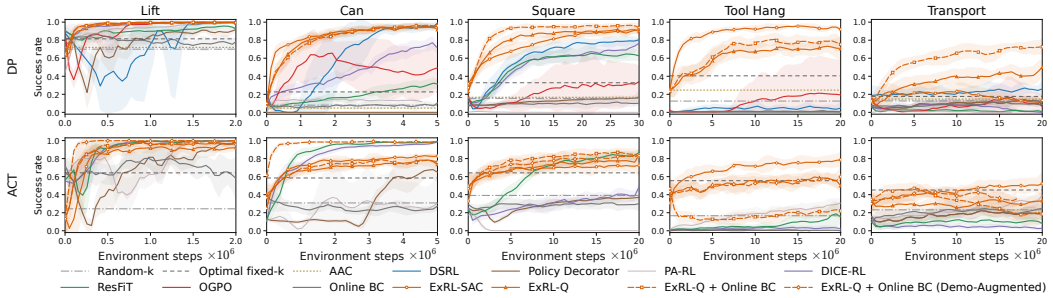


Figure 2: ExRL enables sample-efficient online adaptation of DP and ACT of chunk size 50 on `Robomimic`.

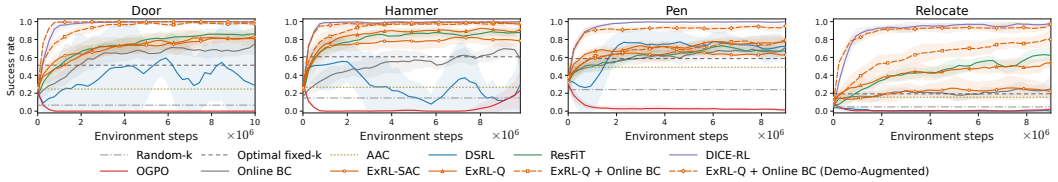


Figure 3: Online adaptation of DP on `Adroit` with chunk size $H = 32$.

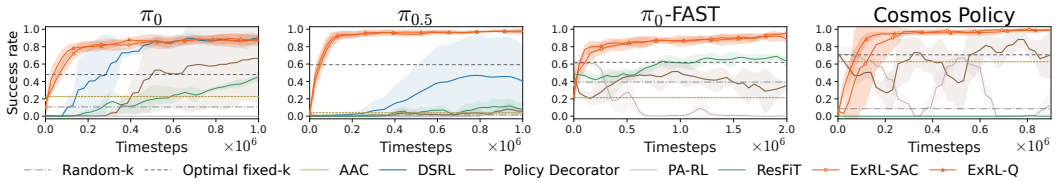


Figure 4: ExRL enables sample-efficient online adaptation of different classes of generalist robot policies on `Libero`, including the flow-based VLA models π_0 and $\pi_{0.5}$, the autoregressive VLA model π_0-FAST, and the world action model Cosmos Policy. All base policies are of chunk size 50.

ing long-horizon tasks such as `Tool Hang` and `Transport`; `Adroit` (Rajeswaran et al., 2017), spanning four high-frequency dexterous hand manipulation tasks ($H = 32$); and a `LIBERO` (Liu et al., 2023) task following the DSRL setup, except that we use $H = 50$ instead of $H = 10$. On `Robomimic`, the main experiments use $H = 50$ for DP and ACT; Appendix Figure 12 reports additional $H = 32$ results for both policies. Following prior work (Wagenmaker et al., 2025; Patil et al., 2026), we use base-policy checkpoints that leave meaningful headroom for RL improvement. Our goal on these benchmarks is not to maximize base-policy performance, but to use them as a controlled testbed for evaluating long-chunk policy adaptation. For each variant of ExRL, we keep most hyperparameters fixed across tasks and make only a few task-level adjustments when necessary, such as the discount factor γ and network width. Appendix A provides task details, policy architectures, baseline adaptations, and hyperparameter settings. For each run in the plot, we report the average success rate and 95% confidence intervals over three seeds.

5.3 Sample-Efficient Adaptation in Simulation

Figures 2, 3, and 4 show that ExRL substantially improves frozen base policies, often reaching success rates close to 100% by learning only the execution length. The gains over the optimal fixed-k baseline are particularly large on DP `Can`, `Square`, and `Tool Hang`, and across all four `LIBERO` policies. As these improvements require no changes to the actions predicted by the base policy, they validate the importance and expressiveness of execution-length control as an adaptation interface.

Compared with other RL methods, ExRL often learns substantially faster, particularly early in online adaptation. On `LIBERO`, for example, it improves $\pi_{0.5}$ from nearly 0% to over 90% success within 200,000 environment steps. The advantage is also pronounced on the challenging `Tool Hang` task, where ExRL-SAC achieves high success with both DP and ACT; with DP, it exceeds 90% success while all other baselines remain at around 20% or below.

Online BC further improves both sample efficiency and final performance, particularly on DP `Transport` and several Adroit tasks. In these settings, ExRL-Q + Online BC substantially outperforms both frozen-base ExRL-Q and Online BC alone, supporting the value of distilling the improved behaviors discovered by ExRL back into the base policy.

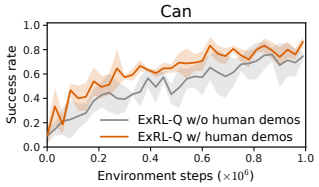


Figure 5: Offline human demonstrations improve early online adaptation of ExRL-Q on `Can` ($H = 50$).

With demonstration augmentation, ExRL-Q + Online BC rapidly reaches near-perfect success on ACT `Can` and all four Adroit tasks, competitive with the strongest demonstration-based baselines. Figure 5 further isolates the benefit of demonstrations for execution-length learning: adding human experience to RL replay accelerates early adaptation on `Can` while keeping the base policy frozen.

The `LIBERO` results also highlight the generality of ExRL. The same execution-length formulation enables efficient adaptation of flow-based VLAs, autoregressive VLAs, and a world action model, despite their substantially different architectures and action-generation processes.

5.4 Sample-Efficient Real-World Adaptation

We further adapt $\pi_{0.5}$ with chunk size $H = 50$ to two challenging real-world tasks (Figure 6). *Power insertion* requires precise contact-rich manipulation to insert a charger plug into a power strip. *Parts organization* requires placing four small parts into their corresponding compartments and closing the box, testing long-horizon coordination. We report success rate for power insertion and the fraction of completed stages for parts organization. Further setup details are provided in Appendix A.7.

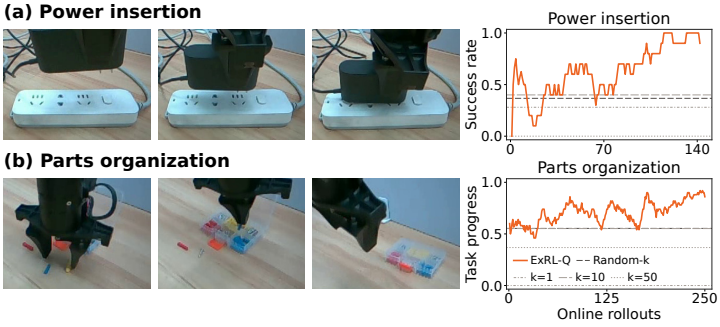


Figure 6: **Real-world adaptation of $\pi_{0.5}$.** Three representative keyframes illustrate each task. The curves show rolling averages over ten online rollouts, including warmup, with random- and fixed-k baselines.

Method	Power	Parts
Random k	$11/30$	$83/150$
$k = 1$	$7/25$	$0/125$
$k = 10$	$10/25$	$69/125$
$k = 50$	$0/25$	$46/125$
ExRL-Q	$\mathbf{22/25}$	$\mathbf{104/125}$

Table 1: Final evaluation. Power: successful trials; Parts: completed stages.

ExRL-Q improves both tasks within a few hundred real-world rollouts while keeping the base policy frozen. Final evaluations (Table 1) yield 88% success on power insertion and 83.2% task progress on parts organization, substantially outperforming random- and fixed-length execution.

5.5 Ablation Studies

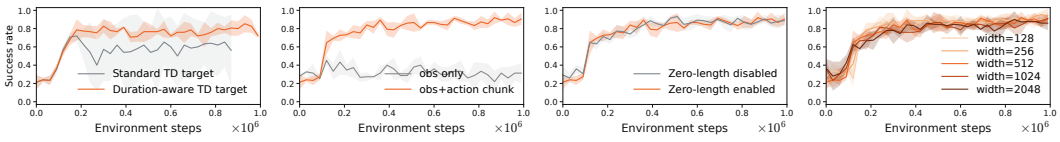


Figure 7: **Ablation studies on `Can` with $H = 32$.** Left to right: duration-aware TD target, conditioning on the predicted action chunk, zero-length execution, and network width.

The leftmost panel of Figure 7 shows that the duration-aware TD target achieves higher success than a standard TD target that treats each execution-length decision as a single step, supporting the need to account for duration in reward accumulation and bootstrapping. The second panel shows the importance of conditioning on the predicted chunk: using observations alone yields around 30% success, whereas including the chunk raises success to nearly 90%. The execution-length decision

therefore benefits substantially from knowing what the base policy proposes, beyond the current state alone. In the third panel, allowing $k = 0$ produces nearly identical learning curves to using only positive lengths, indicating that short positive executions already provide sufficient opportunities to replan on this task. Finally, the rightmost panel shows similar learning curves across network widths from 128 to 2048, suggesting that ExRL does not require precise tuning of network capacity.

5.6 What Execution Length Patterns Does ExRL Learn?

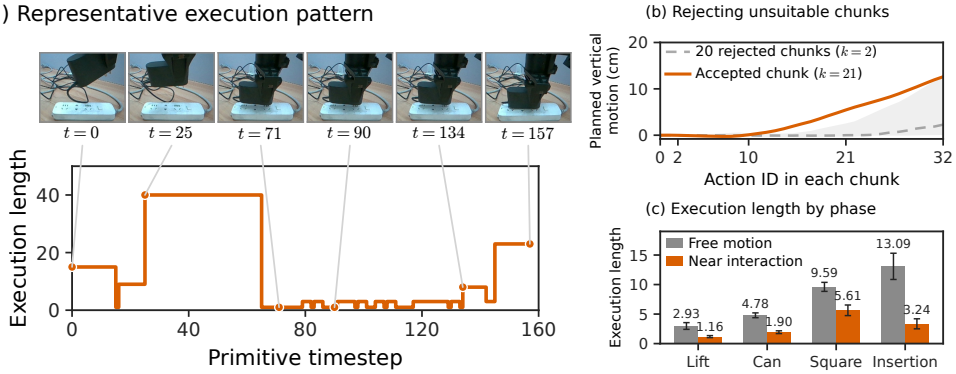


Figure 8: **Execution patterns and further analysis of ExRL's decisions.** (a) A representative power insertion rollout. (b) On `Square`, ExRL truncates 20 proposals at $k = 2$ before accepting a useful chunk with $k = 21$. Curves show the full planned end-effector vertical motion; gray denotes the median and 10–90% range of rejected proposals. (c) Mean selected execution lengths during free motion and near interaction, with 95% trajectory-bootstrap intervals ($H = 32$ in simulation; $H = 50$ for real insertion).

Figure 8(a) illustrates how ExRL varies execution length during a real-world *power insertion* rollout. It uses longer executions to move the plug toward the socket, then repeatedly replans near contact before committing to a longer execution to complete insertion.

The predicted chunks help explain how ExRL makes these decisions. In the `Square` rollout in Figure 8(b), ExRL truncates 20 consecutive proposals at $k = 2$, producing negligible end-effector motion, before committing to $k = 21$. The accepted chunk predicts a sustained grasp-and-lift motion, whereas the preceding proposals have little median vertical motion over the same interval. ExRL thus uses short executions to replan until a useful prediction appears, then allows that chunk to progress.

The preference for shorter execution near interaction also holds across tasks (Figure 8(c)). We compare queries during free motion with those within 8 cm of the object or target, before the first grasp or insertion. Mean execution lengths fall from 5.76 to 2.89 across `Lift`, `Can`, and `Square`, and from 13.09 to 3.24 on real-world insertion. Together with the chunk comparison, this shows how ExRL adapts to both the state and predicted actions. Appendix D.1 provides statistical details and a position-correction example; Appendices D.2 and D.3 further explain why conditional length selection matters and how feature aliasing can make frequent replanning counterproductive.

6 Conclusion

We introduced ExRL, which shifts online adaptation of long-chunk policies from action generation to execution-length control. Conditioning this scalar decision on the state and predicted chunk provides substantial behavioral flexibility, while keeping optimization low-dimensional as chunk size grows. Across simulation and real-world tasks, ExRL enables sample-efficient improvement of diverse frozen policies, often surpassing their best fixed execution length. Online imitation learning and offline demonstrations further extend these gains and support continued base-policy improvement during RL. These findings show that learning how to execute a policy's predictions can unlock capabilities already present in the policy, making execution length a practical interface for adapting long-chunk robot policies with limited online interaction.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code implementation, running experiments, and revising the manuscript. The human authors conceived the research ideas, designed and iteratively refined the experiments, and wrote the initial manuscript, including the development and organization of the related work. We have reviewed all AI-assisted work. In particular, the main author carefully verified every experiment and its results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Algorithms 1 and 2 provide pseudocode for ExRL-Q and ExRL-SAC. Appendix A documents the tasks, demonstrations, policy architectures, pretraining settings, and adaptation hyperparameters for our experiments. Our adaptations of the official baseline implementations and their hyperparameter selection are detailed in Appendix A.6. Our anonymous code release includes DP and ACT pretraining, ExRL training and evaluation, and the online BC and demonstration-augmented variants on Robomimic, together with installation instructions and task-specific configurations. The project website provides simulation and real-world rollouts with execution-length visualizations, as well as an uncut real-robot training recording.

REFERENCES

Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. $\pi_{0.7}$: A steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a VLA that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

Lars Ankile, Zhenyu Jiang, Rocky Duan, Guanya Shi, Pieter Abbeel, and Anusha Nagabandi. Residual off-policy RL for finetuning behavior cloning policies. *arXiv preprint arXiv:2509.19301*, 2025a.

Lars Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne, and Pulkit Agrawal. From imitation to refinement—residual RL for precise assembly. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 01–08. IEEE, 2025b.

Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pp. 12–20, 1997.

André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. TempoRL: Learning when to act. In *International Conference on Machine Learning*, pp. 914–924. PMLR, 2021.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

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

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

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

Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Quanlu Zhang, Zhaofei Yu, Guoliang Fan, et al. π_{RL}: Online RL fine-tuning for flow-based vision-language-action models. *arXiv preprint arXiv:2510.25889*, 2025.

Qian Chen, Junqiao Zhao, Hongtu Zhou, Hang Yu, Yanping Zhao, Chen Ye, and Guang Chen. ACSAC: Adaptive chunk size actor-critic with causal transformer Q-network. *arXiv preprint arXiv:2605.11009*, 2026.

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. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Perry Dong, Qiyang Li, Dorsa Sadigh, and Chelsea Finn. EXPO: Stable reinforcement learning with expressive policies. *arXiv preprint arXiv:2507.07986*, 2025a.

Perry Dong, Suvir Mirchandani, Dorsa Sadigh, and Chelsea Finn. What matters for batch online reinforcement learning in robotics? *arXiv preprint arXiv:2505.08078*, 2025b.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on robot learning*, pp. 158–168. PMLR, 2022.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

S Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. DROID: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

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

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. *arXiv preprint arXiv:2601.16163*, 2026.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

Aravind Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic action repetition for deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Filippo Lazzati, Kyle Stachowicz, William Chen, Alberto Maria Metelli, Andrew Wagenmaker, and Sergey Levine. Why does action chunking improve behavioral cloning performance in robotic control? *arXiv preprint arXiv:2608.02547*, 2026.

Kun Lei, Huanyu Li, Dongjie Yu, Zhenyu Wei, Lingxiao Guo, Zhennan Jiang, Ziyu Wang, Shiyu Liang, and Huazhe Xu. RL-100: Performant robotic manipulation with real-world reinforcement learning. *arXiv preprint arXiv:2510.14830*, 2025.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, et al. SimpleVLA-RL: Scaling VLA training via reinforcement learning. *arXiv preprint arXiv:2509.09674*, 2025a.

Qiyang Li, Zhiyuan Paul Zhou, and Sergey Levine. Reinforcement learning with action chunking. *Advances in Neural Information Processing Systems*, 38:55518–55553, 2025b.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. *arXiv preprint arXiv:2604.04161*, 2026.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

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

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

Guanxing Lu, Wenkai Guo, Chubin Zhang, Yuheng Zhou, Haonan Jiang, Zifeng Gao, Yansong Tang, and Ziwei Wang. VLA-RL: Towards masterful and general robotic manipulation with scalable reinforcement learning. *arXiv preprint arXiv:2505.18719*, 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. *arXiv preprint arXiv:2108.03298*, 2021.

Max Sobol Mark, Tian Gao, Georgia Gabriela Sampaio, Mohan Kumar Srirama, Archit Sharma, Chelsea Finn, and Aviral Kumar. Policy agnostic RL: Offline RL and online RL fine-tuning of any class and backbone. *arXiv preprint arXiv:2412.06685*, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Sarvesh Patil, Mitsuhiko Nakamoto, Manan Agarwal, Shashwat Saxena, Jesse Zhang, Giri Anantharaman, Cleah Winston, Chaoyi Pan, Douglas Chen, Nai-Chieh Huang, et al. OGPO: Sample efficient full-finetuning of generative control policies. *arXiv preprint arXiv:2605.03065*, 2026.

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. *arXiv preprint arXiv:2501.09747*, 2025.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Allen Ren, Justin Lidard, Lars Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. In *International Conference on Learning Representations*, volume 2025, pp. 77288–77329, 2025.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

Yongjae Shin, Jongseong Chae, Seongmin Kim, Jongeui Park, and Youngchul Sung. Adaptive action chunking via multi-chunk Q value estimation. *arXiv preprint arXiv:2605.10044*, 2026.

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

Max Simchowitz, Daniel Pfrommer, and Ali Jadbabaie. The pitfalls of imitation learning when actions are continuous. *arXiv preprint arXiv:2503.09722*, 2025.

Zhanyi Sun and Shuran Song. From prior to pro: Efficient skill mastery via distribution contractive RL finetuning. *arXiv preprint arXiv:2603.10263*, 2026.

Sunday Robotics. ACT-2 preview: Generalizing reliability. *Sunday Robotics Blog*, jul 2026. URL https://www.sunday.ai/blog/act-2-preview.

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.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *arXiv preprint arXiv:2506.15799*, 2025.

Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. BridgeData V2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pp. 1723–1736. PMLR, 2023.

Rui Wang, Yue Zhang, Jiehong Lin, Kuncheng Luo, Jianan Wang, Zhongrui Wang, and Xiaojuan Qi. When to trust imagination: Adaptive action execution for world action models. *arXiv preprint arXiv:2605.06222*, 2026.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3–4):279–292, 1992.

Yueyang Weng, Xiaopeng Zhang, Yongjin Mu, Yingcong Zhu, and Yanjie Li. Temporal action selection for action chunking. *arXiv preprint arXiv:2511.04421*, 2025.

Charles Xu, Jost Tobias Springenberg, Michael Equi, Ali Amin, Adnan Esmail, Sergey Levine, and Liyiming Ke. RL Token: Bootstrapping online RL with vision-language-action models. *arXiv preprint arXiv:2604.23073*, 2026.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Dongjie Yu, Kun Lei, Zhennan Jiang, Jia Pan, and Huazhe Xu. Beyond action residuals: Real-world robot policy steering via bottleneck latent reinforcement learning. *arXiv preprint arXiv:2605.19919*, 2026.

Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.

Thomas T Zhang, Daniel Pfrommer, Chaoyi Pan, Nikolai Matni, and Max Simchowitz. Action chunking and exploratory data collection yield exponential improvements in behavior cloning for continuous control. *arXiv preprint arXiv:2507.09061*, 2025.

Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 5628–5635. Ieee, 2018.

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

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

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

Algorithm 1 Execution-Length Control via Q-Learning (ExRL-Q)

input: pretrained action-chunked policy π_{base}, online environment $\mathcal{M}$, execution-length space $\mathcal{K} = \{1, \dots, H\}$
Initialize replay buffer $\mathcal{D}_{\text{replay}}$
Initialize execution-length critic Q_ϕ
Initialize target critic $Q_{\bar{\phi}}$ with $\bar{\phi} \leftarrow \phi$
for each training iteration **do**
  ▷ Online Rollout
  Sample an action chunk $a_{t:t+H} \sim \pi_{\text{base}}(\cdot \mid s_t)$ at the current query state s_t
  Select execution length k_t with Equation (6)
  Execute the first k_t actions in the chunk and observe
    $\hat{r}_t \leftarrow \sum_{i=0}^{k_t-1} \gamma^i r(s_{t+i}, a_{t+i})$ and $s' \leftarrow s_{t+k_t}$
  Add $(s_t, a_{t:t+H}, k_t, \hat{r}_t, s')$ to $\mathcal{D}_{\text{replay}}$
  ▷ Train Critic
  Sample $(s, a, k, \hat{r}, s') \sim \mathcal{D}_{\text{replay}}$ and sample $a' \sim \pi_{\text{base}}(\cdot \mid s')$
  Compute next-state value $V(s', a')$ with Equation (4)
  Compute TD target y_t with Equation (3)
  Compute $\mathcal{L}_{\text{critic}}(\phi)$ with CQL using Equation (9)
  Train critic Q_ϕ by minimizing $\mathcal{L}_{\text{critic}}(\phi)$
  Periodically update target: $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$
return Q_ϕ

Algorithm 2 Execution-Length Control via Discrete Soft Actor-Critic (ExRL-SAC)

input: pretrained action-chunked policy π_{base}, online environment $\mathcal{M}$, execution-length space $\mathcal{K} = \{1, \dots, H\}$, temperature α
Initialize replay buffer $\mathcal{D}_{\text{replay}}$
Initialize execution-length critic Q_ϕ
Initialize target critic $Q_{\bar{\phi}}$ with $\bar{\phi} \leftarrow \phi$
Initialize execution-length actor π_ψ
for each training iteration **do**
  ▷ Online Rollout
  Sample an action chunk $a_{t:t+H} \sim \pi_{\text{base}}(\cdot \mid s_t)$ at the current query state s_t
  Sample execution length $k_t \sim \pi_\psi(\cdot \mid s_t, a_{t:t+H})$
  Execute the first k_t actions in the chunk and observe
    $\hat{r}_t \leftarrow \sum_{i=0}^{k_t-1} \gamma^i r(s_{t+i}, a_{t+i})$ and $s' \leftarrow s_{t+k_t}$
  Add $(s_t, a_{t:t+H}, k_t, \hat{r}_t, s')$ to $\mathcal{D}_{\text{replay}}$
  ▷ Train Critic and Actor
  Sample $(s, a, k, \hat{r}, s') \sim \mathcal{D}_{\text{replay}}$ and sample $a' \sim \pi_{\text{base}}(\cdot \mid s')$
  Compute the soft value $V(s', a')$ with Equation (7)
  Compute TD target y_t with Equation (3)
  Compute $\mathcal{L}_{\text{critic}}(\phi)$ with CQL using Equation (9)
  Compute actor loss $\mathcal{L}_\pi(\psi)$ with Equation (8)
  Train critic Q_ϕ by minimizing $\mathcal{L}_{\text{critic}}(\phi)$
  Train actor π_ψ by minimizing $\mathcal{L}_\pi(\psi)$
  Periodically update target: $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$
return π_ψ

A Experimental Details

A.1 Simulation Tasks and Demonstrations

Figure 9 illustrates the simulation tasks. **Robomimic** uses state observations comprising robot proprioception and object states, with control at 20 Hz. Lift requires lifting a cube, Can requires placing a can in a bin, and Square requires fitting a nut onto a peg. Tool Hang involves assembling a frame and hanging a tool on it, while Transport requires two arms to transfer an object between bins. We train the base policies on 300 Multi-Human (MH) demonstrations per task for Lift, Can, Square, and Transport, and on 200 Proficient-Human (PH) demonstrations for Tool Hang (Mandlekar et al., 2021).

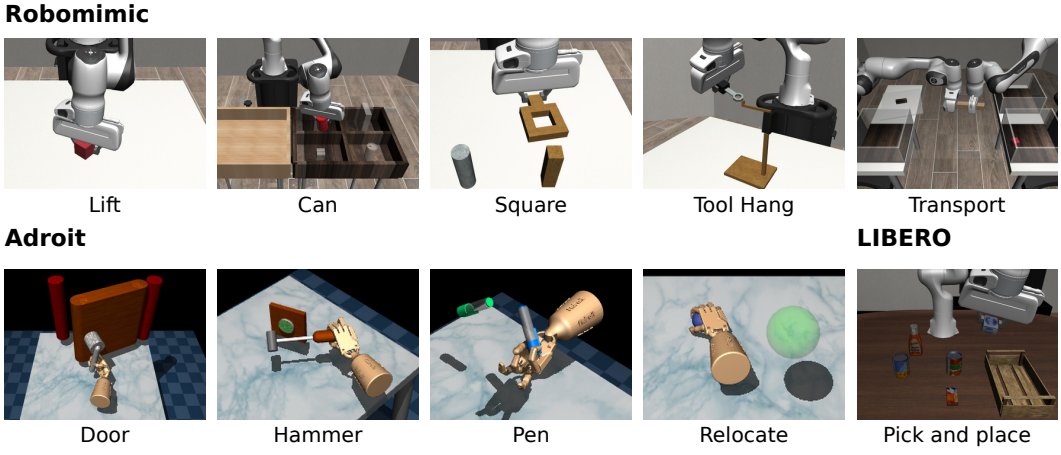


Figure 9: Robomimic (top), Adroit (bottom left), and LIBERO (bottom right) simulation tasks.

Adroit uses state observations and controls a dexterous hand at 100 Hz. Its four tasks require opening a door, driving a nail with a hammer, reorienting a pen, and relocating a ball to a target. The

DP base policies are trained on the D4RL (Fu et al., 2020) expert-v2 datasets, containing 5,000 trajectories each for Door, Hammer, and Relocate, and 4,958 for Pen.

LIBERO uses image observations, proprioception, and a language instruction. We use the LIBERO-90 task "pick up the cream cheese and put it in the tray," which runs at 20 Hz and provides 50 human demonstrations (Liu et al., 2023). These demonstrations are used for task-specific fine-tuning of the generalist base policies described below.

Across all three benchmarks, ExRL uses a sparse reward of -1 at each primitive timestep before success and 0 upon success, when the episode terminates. Success is determined by the benchmark's task-specific success condition.

A.2 Policy Architectures and Pretraining

DP and ACT. Both policies condition on the current state and predict an H-step action chunk. On Robomimic, DP uses a residual MLP denoiser with three hidden layers. The hidden width is 512 for Lift and Can, 1024 for Square, 2048 for Tool Hang, and 4096 for Transport. On Adroit, DP uses a temporal 1D U-Net with channel widths $(128, 256, 512)$ and a 128-dimensional diffusion-step embedding. All DP models use 100 diffusion steps during training; inference uses 5 DDIM steps on Lift and Adroit and 20 on the other Robomimic tasks.

ACT uses a transformer encoder–decoder to predict the action chunk from state inputs. For Lift, Can, and Square, it has a hidden dimension of 512, 8 attention heads, 4 encoder layers, 1 decoder layer, and a feedforward dimension of 3200. For Tool Hang and Transport, these values are 768, 12, 6, 1, and 5120, respectively. We train ACT without the variational latent encoder and execute its predicted chunks without temporal ensembling.

Table 2 reports the training epochs of the Robomimic base-policy checkpoints. For Adroit, we use DP checkpoints at epoch 2 for Door, Hammer, and Pen, and epoch 10 for Relocate, with 1000 gradient updates per epoch and a batch size of 1024.

	DP		ACT	
Task	$H = 32$	$H = 50$	$H = 32$	$H = 50$
Lift	20,000	10,000	0.25	0.12
Can	20,000	20,000	0.92	0.82
Square	20,000	20,000	1190.48	1388.89
Tool Hang	4,000	28,000	100.10	100.07
Transport	10,000	10,000	328.59	250.08

Table 2: Training epochs of the Robomimic base policies used to initialize online adaptation. Fractional ACT epochs correspond to checkpoints saved within a data pass.

Generalist base policies. On LIBERO, we use the released π_0 LIBERO checkpoint and task-specific checkpoints of $\pi_{0.5}$, π_0-FAST, and Cosmos-Policy after 50, 10,000, and 1500 fine-tuning updates, respectively. All receive 224×224 external-camera and wrist-camera images, proprioception, and the task instruction, and predict chunks of length $H = 50$.

ExRL on image observations. On LIBERO, ExRL encodes a 64×64 external-camera image using four 32-channel convolutional layers with 3×3 kernels, ReLU activations, and strides $(2, 1, 1, 1)$. A fully connected layer projects the flattened features to a 50-dimensional tanh bottleneck, which is concatenated with proprioception and the flattened predicted chunk. Ten 512×3 MLP critic heads share the encoder and output values for each execution length. The SAC actor uses its own encoder and MLP of the same dimensions, with a categorical output. Random crops and color jitter augment images during RL updates.

A.3 Action-Chunk Length Study

Figure 1 examines how action-chunk length affects online adaptation. For convenience, we construct policies with different chunk sizes by exposing only the first $L \in \{2, 4, 8, 16, 25, 50\}$ actions of one pretrained DP checkpoint with $H = 50$ on `Tool Hang`. Each truncated policy presents the same

black-box interface to the baseline RL policies as a base policy with chunk size L: actions beyond the prefix are neither executed nor visible to their actors or critics. ResFiT and DICE-RL learn residual corrections only for the exposed L actions, whereas ExRL-SAC selects an execution length $k \leq L$ from the exposed prefix.

For DSRL, we also restrict the learnable latent space to the prefix. Its actor outputs an $L \times d_a$ noise tensor, where $d_a = 7$ is the primitive action dimension. Inside the base-policy wrapper, the remaining $(H - L) \times d_a$ noise entries are independently sampled from a clipped Gaussian. The original diffusion policy then decodes the complete H-step chunk, but the wrapper returns only the first L actions. The action critic receives only this action prefix, and the noise critic receives only the actor's L-step noise; neither sees the hidden suffix. The pretrained diffusion model remains unchanged, while DSRL's learned control dimension scales with L. All variants use 20 DDIM steps and a task horizon of 800 primitive steps. We report three-seed means and 95% confidence intervals.

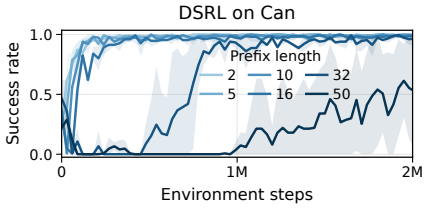


Figure 10: **DSRL on Can.** Three-seed means and 95% CIs. Prefixes 2–32 share an $H = 32$ checkpoint; prefix 50 uses an $H = 50$ checkpoint.

The frozen DP on `Tool Hang` is itself less successful when executing long prefixes (Figure 16). To separate the effect of chunk length on RL adaptation from differences in base-policy performance, we additionally compare DSRL on `Can`, where DP has similar success rates across prefixes. As shown in Figure 10, DSRL quickly reaches near-perfect success with prefixes of 2–16 actions, but learns substantially more slowly with 32 and 50 actions. The sample-efficiency gap thus persists even when the base policy performs similarly at different prefix lengths.

A.4 Shared Hyperparameters

Most settings of ExRL are shared across tasks and base policies. Table 3 gives the defaults for ExRL-Q and ExRL-SAC, and Table 4 lists the main task-specific changes. The same RL settings apply when Online BC is enabled.

Parameter	ExRL-Q	ExRL-SAC
Optimizer	Adam	Adam
Critic learning rate	3×10^{-4}	3×10^{-4}
Critic hidden layers	512×3	512×3
Activation	ReLU	ReLU
Critic ensemble size	10	10
Target critic subsample size	2	2
Target update rate τ	0.005	0.005
Batch size	256	256
Replay capacity (transitions)	10^6	10^6
Primitive discount γ	0.997	0.997
CQL coefficient	0.3	0.01
CQL temperature	1	1
TD target clipping	$[-1000, 0]$	$[-1000, 0]$
UTD	$1/4$	1
Warmup size	512	512
Updates on warmup replay	5,000 (critic)	5,000 (actor–critic)
Actor hidden layers	—	Same as critic
Actor learning rate	—	5×10^{-5}
Initial entropy temperature	—	0.3
Temperature learning rate	—	10^{-5}
Target entropy	—	$0.98 \log H$

Table 3: Shared ExRL hyperparameters.

Setting	Changes to the shared defaults
Transport, all base policies	Critic and SAC actor hidden layers: 2048×3.
Transport, DP; ACT with ExRL-SAC	$\gamma = 0.999$.
Tool Hang, all base policies	$\gamma = 0.999$.
LIBERO, all base policies	Replay capacity: 10^5; ExRL-Q UTD: 1.

Table 4: Main task-specific hyperparameter changes.

Default episode horizons are 300 for Lift and Can, 400 for Square, 800 for Tool Hang and Transport, 200 for Adroit, and 250 for LIBERO. Base-policy architectures and the LIBERO image encoder are described in Appendix A.2.

A.5 Online BC and Demonstration Augmentation

Table 5 summarizes the default base-policy update settings. Successful trajectories are sampled uniformly over episodes, and BC starts once enough successful experience is available. For DP and ACT, we add a teacher constraint to the online BC loss using a frozen copy of the initial base policy. DP minimizes the mean squared difference between student and teacher noise predictions at identical observations, noisy chunks, and diffusion timesteps; ACT matches their inference-time action chunks at identical observations. The teacher stays fixed throughout RL, limiting drift from pretrained behavior as the student learns from successful online trajectories.

Parameter	DP	ACT
BC learning rate	10^{-5}	10^{-5}
BC batch size	256	128
BC updates / RL update	1/2	1/32
Success-buffer capacity (windows)	25,000	25,000
Teacher weight	1	1

Table 5: Online BC hyperparameters. DP settings apply to both Robomimic and Adroit.

DP Tool Hang delays BC until 200k environment steps. ACT uses a gradient-norm limit of 10.

Demonstration augmentation. The default demonstration fraction in each BC batch is 0.5, and the frozen-teacher constraint is disabled. The number of BC updates per RL update decreases linearly from 5 to 1 over the first 600k environment steps after BC begins, and remains at 1 thereafter.

Demonstrations in RL replay. For the `Can` comparison in Figure 5, the critic batch contains 20% relabeled demonstration transitions and 80% online transitions. The demonstration buffer holds up to 2.5M transitions, separately from the online replay.

A.6 Baseline Implementations and Settings

We adapt the released baselines to our policy backbones and chunk sizes. Table 6 summarizes hyperparameter selection; below, we describe changes to their policy interfaces and learning procedures.

DSRL. The released Robomimic configurations of Wagenmaker et al. (2025) typically use chunks of 4 or 8 actions. We connect the official learner to our DP checkpoints and extend its latent action space to H-step chunks. The diffusion sampler and normalization follow each checkpoint. We consider both DSRL-NA and DSRL-SAC during tuning. For example, the selected DP `Transport` and `Tool Hang` configurations use DSRL-SAC, a latent-noise bound of 0.75, and a learning rate of 5×10^{-5}, with network widths of 2048 and 512, respectively. The restricted latent interface used in the prefix study is described in Appendix A.3.

Policy Decorator. We port the residual SAC learner of Yuan et al. (2024) to our training framework while retaining bounded residual actions and progressive exploration. The residual has the same shape as the executed action chunk, and the actor conditions on both the observation and the base chunk, using the observation-plus-base-action option of the official code instead of its observation-only default. For DP with $H = 50$, the selected residual scales are 0.1, 0.15, 0.05, 0.02, and 0.02 on

Baseline	Hyperparameter selection
DSRL	Task-specific searches over the DSRL-NA/SAC variant, latent-noise range, learning rate, update frequency, discount, and network width.
Policy Decorator	Task-specific searches over residual scale, entropy coefficient, learning rates, progressive exploration, and actor/critic update frequencies.
PA-RL	Searches over critic capacity and update frequency, action-optimization steps and step size, and the learning rate, start time, and frequency of policy distillation.
ResFiT	Search on DP `Square`, then reuse the selected optimizer, residual, exploration, and update settings across tasks and backbones.
OGPO	Adapt the released learning objectives to DP and adjust update frequency for our collection setup and longer chunks.
DICE-RL	Use the paper's Can recipe for the main learning hyperparameters, with adaptations to our policies, replay, and demonstration datasets.

Table 6: Hyperparameter selection for the RL baselines.

`Lift`, `Can`, `Square`, `Tool Hang`, and `Transport`, respectively. The two long-horizon tasks also use smaller entropy coefficients.

PA-RL. We implement Mark et al. (2024) in our PyTorch framework, replacing its original base-policy interface with our DP and ACT models. Action optimization operates on complete chunks, and distillation uses the corresponding native training objective: denoising for DP and action reconstruction for ACT. We retain the official distributional critic ensemble and Cal-QL regularization, with 10 critics, 2-critic target subsampling, and regularization coefficient 0.005. Our main changes concern the interaction between critic learning, action optimization, and policy distillation. We first warm up the critic, then collect experience with optimized actions and periodically distill them into the base policy. For DP, we sample 32 candidates and retain 10 for optimization. ACT retains its deterministic inference, so repeated base candidates coincide; local optimization and the inclusion of replay actions provide the action-improvement step. Local search uses 10–20 gradient steps, with its step size and the distillation schedule tuned by task. On DP `Lift`, distillation uses globally selected candidates without local refinement, whereas rollout actions still receive local optimization. `Lift`, `Can`, and `Square` use online replay only. The $H = 50$ `Transport` and `Tool Hang` runs additionally mix equal proportions of offline demonstrations and online experience, following the official mixed-data recipe. Reported performance evaluates the optimized-action policy.

ResFiT. We extend the primitive-action residual formulation of Ankile et al. (2025a) to complete action chunks: the residual actor receives the state and base chunk, and the critic evaluates the composed chunk. Replay stores one transition per executed chunk, with accumulated discounted rewards and a bootstrap discount determined by its actual duration. We retain the TD3-style updates, 10-critic ensemble, bounded residuals, and equal mixing of demonstration and online data. After searching on DP `Square`, we reuse actor/critic learning rates of $5 \times 10^{-6}/10^{-4}$, residual scale 0.1, exploration/target noise standard deviations of $0.025/0.01$, and target update rate 0.01. Each collected chunk transition receives one critic update and one actor update every four critic updates. We use one-chunk TD targets instead of the released primitive-step multi-step targets. Network width and discount remain task-specific. The same learner is used with ACT and Adroit; for LIBERO, we add image encoders and connect it to each policy's native observation and action transforms.

OGPO. The released implementation of Patil et al. (2026) uses flow-matching policies in JAX. To fine-tune our pretrained DP models directly, we implement its learning objectives in PyTorch and construct stochastic denoising trajectories around the DP's DDIM transitions. We retain OGPO's clipped likelihood-ratio objective and conservative group advantages, evaluating old and current likelihoods on the same denoising trajectory. We replace flow-matching regularization with the DP denoising loss on successful online experience. The standard runs use 32 samples per advantage group and best-of-8 action selection during collection; evaluation samples a single trajectory from the target policy. We use lower update frequencies than the released configuration, with task-specific schedules for parallel collection and long chunks.

DICE-RL. We adapt Sun & Song (2026) to our frozen DP and ACT policies while retaining its noise-conditioned residual, critic-guided candidate selection, and selective behavior regularization. For DP, the same noise conditions the base policy and residual actor. For ACT, the base chunk

uses its standard deterministic inference, while noise conditions only the residual actor. We use the paper's Can settings: 10 critics, 16 rollout/actor candidates, 4 target candidates, behavior-regularization weight 50, and an offline replay fraction that decreases from 0.7 to 0.1 over 160k environment steps. Relative to the released Can configuration, we enable the paper's candidate-level Q filter and use constant learning rates of 10^{-4}. Demonstrations come from the datasets used for our base policies. Our replay stores non-overlapping chunk transitions, replacing the release's overlapping action windows; three-step TD targets therefore span three chunk decisions, with rewards summed within each chunk and discounting applied between decisions. The RL batch size is 256. Robomimic uses 2.5 critic updates per collected chunk transition, and Adroit uses one; the actor and target networks update half as frequently.

AAC. We adapt the released inference-time AAC selector (Liang et al., 2026) to each policy's native chunk size. Robomimic and LIBERO use 20 candidate chunks and the original entropy-increase rule with its motion constraint. On bimanual `Transport`, we sum the two arms' entropies and average their motion measures. Adroit uses an entropy-only adaptation over joint actions with 64 candidates, since the original Cartesian motion constraint does not apply. For π_0-FAST, auxiliary candidates use token sampling at temperature 1 to estimate entropy, while the executed candidate retains greedy decoding. In every case, AAC selects a prefix without averaging or re-ranking the candidate actions.

Other non-RL baselines. Random-k samples uniformly from $\{1, \ldots, H\}$ at each query, and optimal fixed-k reports the best fixed execution length from a sweep. Online BC executes full chunks and learns only from successful online trajectories, using the imitation objective and teacher constraint described in Appendix A.5, without RL updates or a learned execution-length policy.

A.7 Details of Real-World Experiment

Setting	Power insertion	Parts organization
Language command	Insert the charger into the power strip	place the scattered pieces into the compartment box and close the lid
BC demonstrations	128 (22,865 frames)	21 (19,288 frames)
Base action chunk length	50	50
Warmup rollouts	30	29
Learning rate	3e−4	3e−4
Offline critic updates	5,000	5,000
Online updates per rollout	400	400
Maximum episode length	450 steps	1021 steps
Evaluation trials	25	25

Table 7: Real-world ExRL training settings. The base $\pi_{0.5}$ policies are frozen during ExRL adaptation; only the execution-length critic is trained.



Power insertion task Parts organization task

Figure 11: Real-world task setups.

All real-world experiments use a agilex Piper arm controlled at 20Hz. The frozen base policy is a $\pi_{0.5}$ policy fine-tuned with behavior cloning on task-specific human demonstrations. At each query, the base policy receives two RGB camera views (a third-person camera and a wrist camera) together

with the robot proprioceptive state, and predicts an action chunk of length $H = 50$. ExRL does not change the predicted actions or update the base policy. Instead, it observes the current robot observation and the predicted action chunk, and selects an execution length $k \in \{1, \ldots, 50\}$. The robot executes the first k primitive actions, after which the base policy is queried again. We use primitive discount $\gamma = 0.997$.

We evaluate two real-world tasks, each using a separate $\pi_{0.5}$ base policy and a separate task-specific behavior-cloning dataset. In *power insertion*, the robot must insert a charger plug into a power strip. We use the language instruction "Insert the charger into the power strip" and train the base policy from 128 demonstrations (22,865 frames). We treat this as a single-stage success task: each rollout succeeds when the plug is inserted and otherwise terminates at the horizon. In *parts organization*, the robot must place several different small parts into a compartment storage box and then close the lid. We use the language instruction "place the scattered pieces into the compartment box and close the lid" and train the base policy from 21 demonstrations (19,288 frames). This task has five progress stages: four successful part placements and one final lid-closing stage. During real-world interaction, the operator only provides sparse event annotations: for power insertion, the operator labels the trial as success or failure; for parts organization, the operator presses a key whenever a stage is completed. The operator does not provide action corrections or teleoperation during RL.

Reward and online training protocol. For power insertion, we use a sparse time-to-success reward. Each primitive timestep before success receives reward -1, the terminal success timestep receives reward 0, and failed rollouts terminate at the horizon. For parts organization, we use a sparse stage-completion reward: completing a stage gives reward $+1$ at the corresponding primitive timestep, while all other timesteps have zero reward.

We initialize ExRL with random execution-length exploration before online adaptation. For power insertion, we collect 30 random-k warmup rollouts, producing 419 transitions (10,513 primitive robot steps). For parts organization, we use roughly the same warmup scale, collecting 29 random-k rollouts. We then train the ExRL-Q critic for 5,000 critic-only updates on the warmup replay. During online adaptation, each new real-world rollout is inserted into replay after receiving the sparse human annotation, and the critic is updated for 400 gradient steps before the next rollout. The critic uses an ensemble of 10 Q-functions with mean reduction, hidden dimensions $(1024, 1024, 1024)$, batch size 64, and target critic subsampling size 2. We continue online adaptation for at most a few hundred real-world rollouts; the reported power insertion run used 142 total rollouts, and the parts organization run used 250 total rollouts. After training, we freeze the learned execution-length critic and evaluate for 25 trials per task.

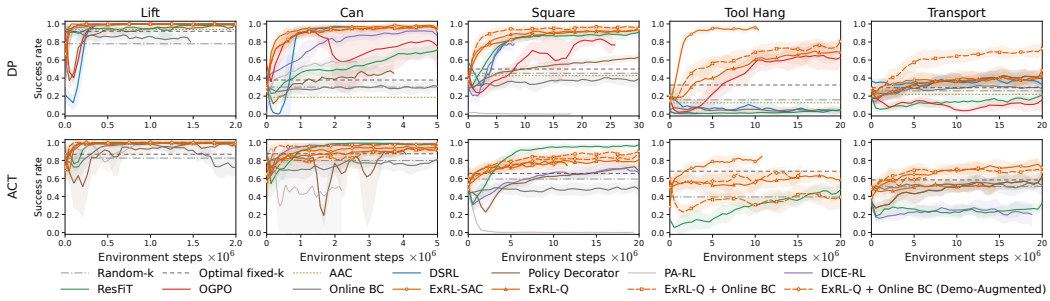


Figure 12: Online adaptation of DP and ACT on `Robomimic` with chunk size $H = 32$.

B On-Policy versus Off-Policy Adaptation

To examine how the choice of RL algorithm affects sample efficiency, we compare ExRL-Q with a PPO execution-length adapter, motivated by the PPO-based approach of Zhao et al. (2026). Both adapters use the same frozen DP checkpoint on `Can`, with chunk size $H = 50$, the same state and action-chunk inputs, and the same execution-length choices.

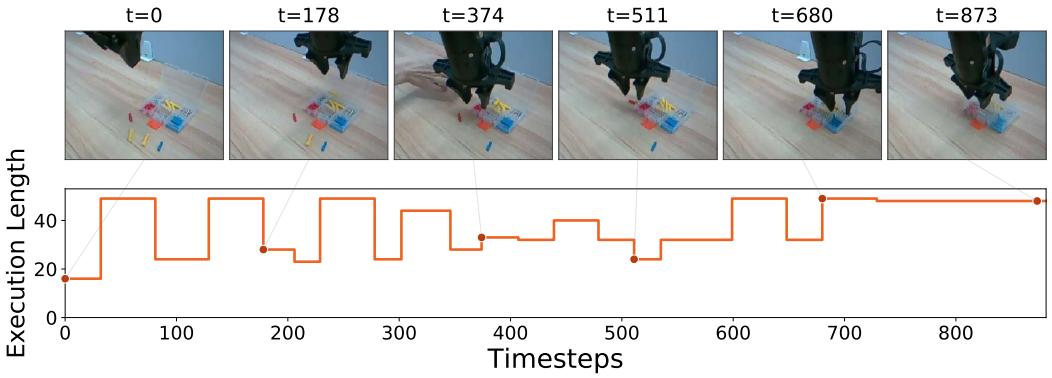


Figure 13: **A representative rollout of ExRL on parts organization.** ExRL selects long execution lengths for confident pick-and-place segments, while adjusting the execution length across stages as the robot places parts into the storage box and closes it.

As shown in Figure 14, ExRL-Q reaches over 90% success within approximately 3M environment steps, while PPO remains below 40% after approximately 40M steps. Curves average success over the most recent 200 training episodes; the horizontal axis counts primitive environment steps. This substantial gap supports the use of off-policy learning for sample-efficient execution-length adaptation.

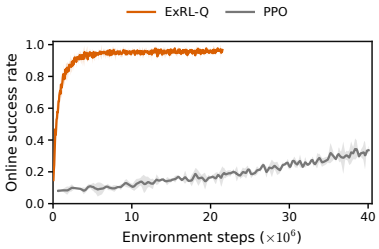


Figure 14: **PPO versus ExRL-Q on Can.** Mean online success over 200 episodes, with 95% CIs.

C Evolution of Learned Execution Lengths

Figure 15 examines how execution-length strategies develop during online adaptation. Alongside the mean execution length, we plot the categorical policy entropy averaged over training minibatches, which measures how concentrated the adapter's decisions are for a given state and predicted chunk. Across tasks and base policies, the learned mean remains well below the full chunk size, but its evolution differs substantially. All four `LIBERO` policies rapidly settle at approximately 3–5 actions, whereas ACT on `Square` gradually favors longer execution, reaching about 14 actions with $H = 50$ and 11 with $H = 32$. Adroit `Pen` exhibits a similar increase.

Policy entropy further distinguishes these strategies. On ACT `Square` and Adroit `Pen`, increasing execution length accompanies decreasing entropy: the adapter becomes more decisive while allowing longer motions. In contrast, the `LIBERO` policies retain nonzero entropy after their mean execution lengths stabilize, so frequent replanning does not require deterministic length selection. The same adapter can therefore learn sustained frequent replanning or an increasing preference for longer motions, depending on the task and base policy.

D Additional Execution-Length Analysis

Section 5.6 illustrates how ExRL adapts execution to the current state and predicted chunk. We provide the details of the phase comparison and another example of short executions, then examine why assigning lengths to the right situations matters and why replanning after every action can prevent progress. The `Robomimic` analyses use frozen base policies with $H = 32$; real-world insertion uses $H = 50$.

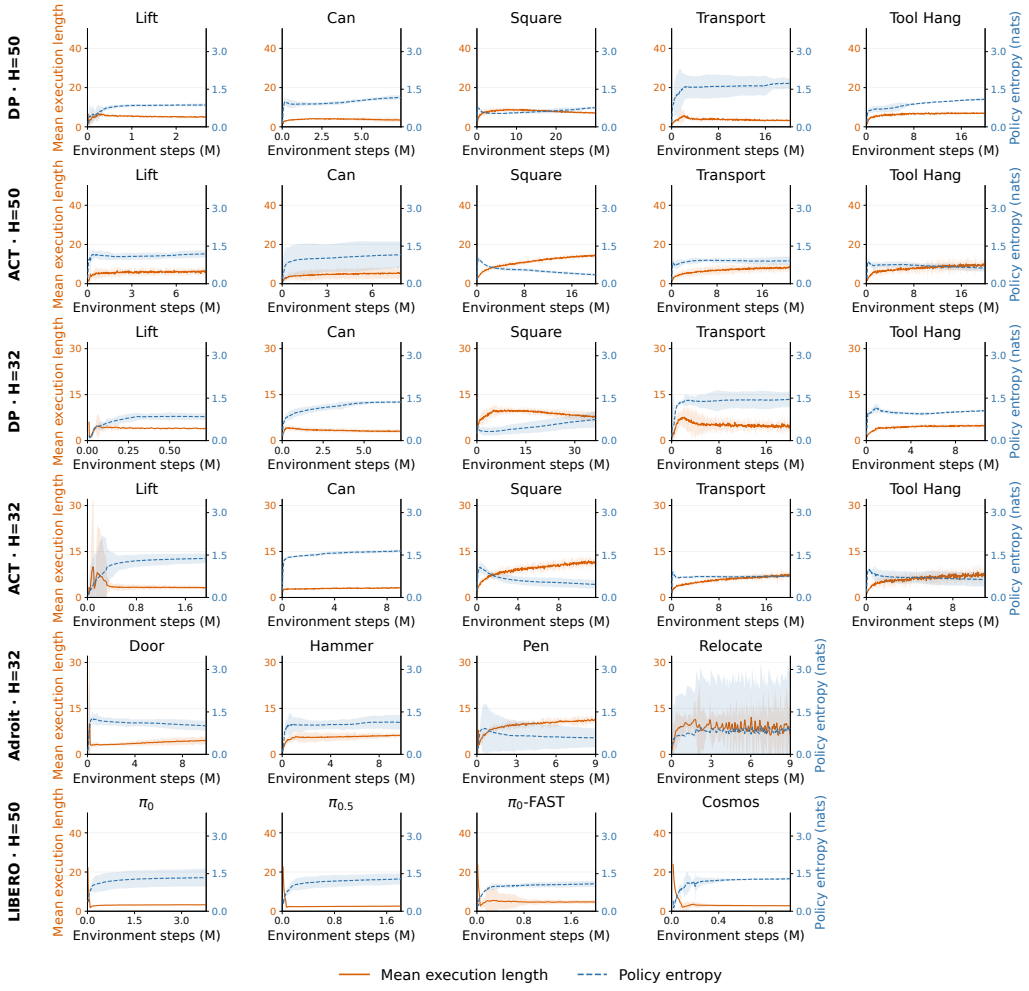


Figure 15: **Evolution of execution lengths and policy entropy learned by ExRL-SAC.** Orange solid curves show mean execution length; blue dashed curves show the categorical policy entropy on training minibatches. Shaded bands indicate 95% confidence intervals over three training seeds.

D.1 Execution Length near Interaction

The phase comparison in Figure 8(c) asks whether ExRL's preference for shorter execution near interaction holds across trajectories and tasks. To focus on the approach to interaction, we exclude the query that first produces a grasp in simulation or completes the insertion stage on the real robot, along with all subsequent queries. Each preceding query is classified by its starting distance from the end effector or tool to the object or target: within 8 cm is *near interaction*, and farther away is *free motion*. We measure the length selected by ExRL, before any truncation by episode termination.

We collect 50 rollouts from each of three trained critics per simulation task and 25 real insertion rollouts. The comparison uses trajectories that reach the interaction and contain queries in both phases: 86 for `Lift`, 145 for `Can`, 121 for `Square`, and 21 for real insertion. We average lengths within each phase of a trajectory, then equally weight trajectories, so both phase estimates use the same set of rollouts. Error bars in Figure 8(c) show 95% trajectory-bootstrap intervals. Shorter execution near interaction occurs in all nine simulation task–seed pairs, and the task-averaged difference remains negative at distance thresholds of 6 and 10 cm. This consistency supports frequent replanning during the approach to interaction, beyond the particular trajectories visualized in the main text.

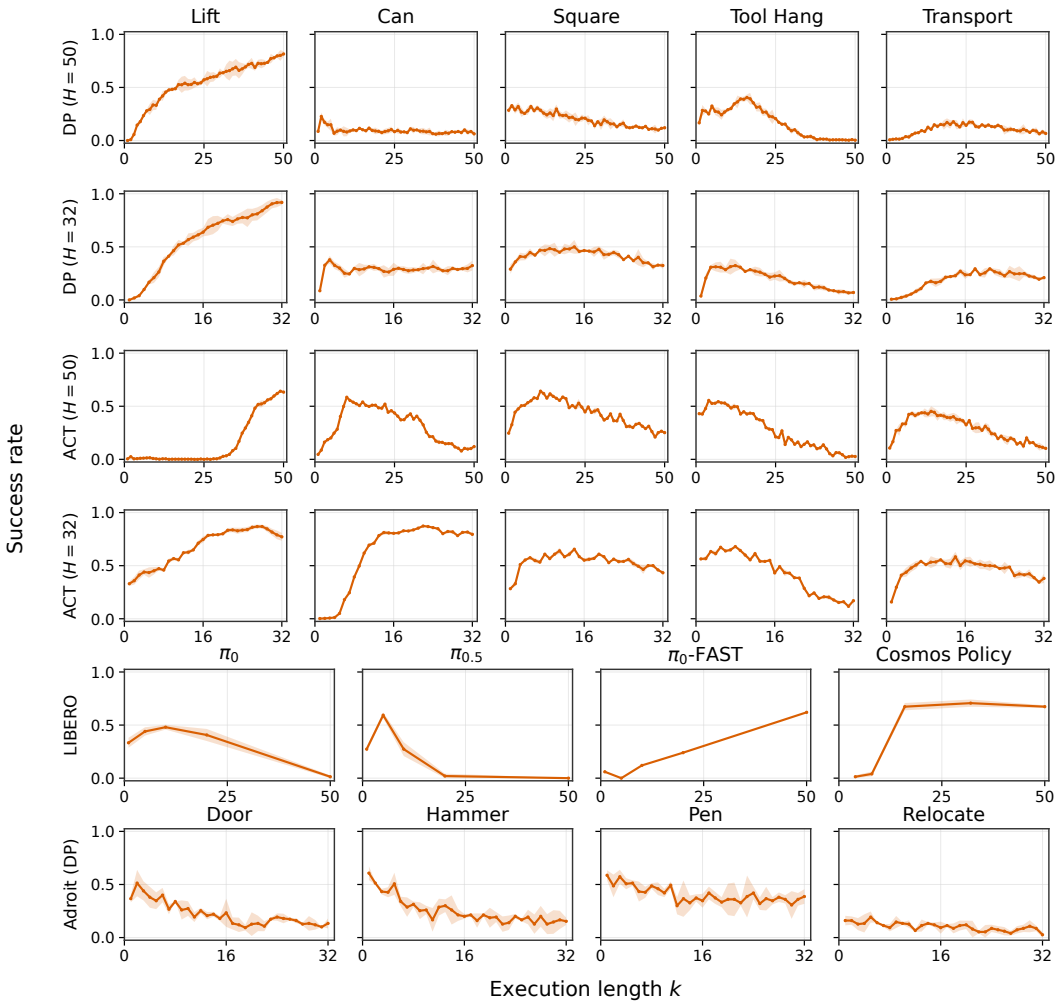


Figure 16: **Fixed-k sweep across base policies and tasks.** We evaluate each base policy under different fixed execution lengths on `Robomimic`, `LIBERO`, and `Adroit`. Performance varies substantially with k, and the optimal fixed execution length differs across tasks and policies, highlighting the importance of adaptive execution-length control. Shaded bands indicate 95% confidence intervals over three seeds.

Short execution can also improve the state before a longer motion, complementing the near-stationary chunk rejection in Figure 8(b).

In Figure 17, 27 one-action executions followed by one three-action execution bring the nut closer to the peg in the horizontal plane. Replanning from these adjusted states eventually leads ExRL to select $k = 28$. The two examples show how frequent replanning can either refine the robot's position or replace an unsuitable prediction before committing to longer execution.

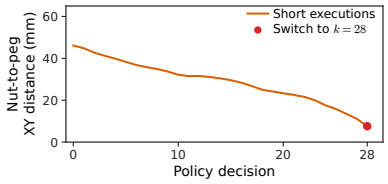


Figure 17: **Position correction on Square.** Short executions align the nut before $k = 28$.

D.2 Does Conditional Execution-Length Assignment Matter?

A policy that uses both short and long executions could benefit simply from a favorable distribution of lengths. To determine whether ExRL also benefits from choosing when to use each length, we compare ExRL-Q with a shuffled-length control on `Square`. At each synchronized batch of queries, ExRL-Q selects

a length for each current state–chunk pair. A second group of environments receives a random permutation of these same lengths. Both groups therefore use exactly the same multiset of lengths at every batch, while the control breaks their original assignment to states and chunks.

Table 8 shows that shuffling the lengths reduces success by 38–58 percentage points across both base policies. The length distribution alone is therefore insufficient to explain ExRL's performance: matching execution lengths to the current state and predicted chunk provides a substantial additional benefit.

Table 8: **Conditional assignment matters beyond the length distribution.** `Square` success rates (counts), and gains over the permuted control in percentage points with 95% cluster-bootstrap intervals.

Base policy	ExRL-Q	Permuted lengths	Gain (95% CI)
DP	95.8% (23/24)	37.5% (9/24)	58.3 [37.5, 79.2]
ACT	70.8% (17/24)	33.3% (8/24)	37.5 [16.7, 58.3]

We use three independently trained ExRL-Q checkpoints per base policy. Paired rollouts share the initial state, initial chunk, and policy-inference randomness; confidence intervals resample checkpoint–environment-slot clusters.

D.3 Feature Aliasing and Repeated Replanning

Replanning after every action can interrupt progression through an otherwise useful prediction. Each query discards the remaining actions and starts again from the first action of a new chunk. When successive states produce similar policy inputs or internal features, this can repeatedly reproduce the same initial behavior. We refer to this local ambiguity as *feature aliasing*: the robot may move without its policy features changing enough to indicate when to proceed to the next part of the behavior.

This interpretation is related to the non-Markovian behavior analyzed by Lazzati et al. (2026). In their drawer-opening example, the demonstrator pauses at a "decision boundary" before changing direction, although the observed state changes little. Whether to wait or move is therefore ambiguous from the current observation alone. Their compounding-error analysis further shows that conditioning on past observations can reduce error amplification, since earlier observations may be less affected by accumulated rollout errors. These benefits can also be obtained with delayed policies, pointing to the value of temporal conditioning beyond simply reducing the frequency of decisions. In our setting, advancing through a useful chunk may similarly avoid repeatedly deciding how to continue from locally ambiguous features.

Our feature–action analysis is consistent with this interpretation. Across DP and ACT on `Lift`, `Can`, and `Square`, we examine repeated $k = 1$ replanning over 32-action windows from 72 states. Smaller changes in policy features tend to accompany smaller changes in the newly predicted first arm action: Figure 18(a) shows positive mean correlations in all six model–task pairs, with an overall Spearman correlation of $\rho = 0.374$ (95% CI $[0.237, 0.494]$). The association persists with 16-action windows ($\rho = 0.383$), suggesting that repeated first actions are linked to local feature similarity across both architectures.

Figure 18(b) makes the distinction between physical motion and progression through a chunk concrete. Starting from the same ACT `Square` state and initial prediction, repeated replanning produces nearly unchanged first actions, while the original chunk contains a sequence of varying commands. During the highlighted interval, the replanning branch moves the end effector by 19.4 mm, yet its mean change between consecutive arm actions is only 3.6% of that along the original chunk. The robot is moving, but repeatedly querying the policy does not recover the later actions of the original prediction.

Allowing a chunk to progress can help, but the appropriate duration still depends on the current state and prediction. Table 9 compares continuations from three ACT `Square` states with low feature variation and repeated first actions, selected before observing the outcomes. Each comparison starts from the same state and chunk and uses the same task horizon. ExRL-Q succeeds in a case where both persistent $k = 1$ and a full 32-action execution followed by ExRL-Q fail. This diagnostic com-

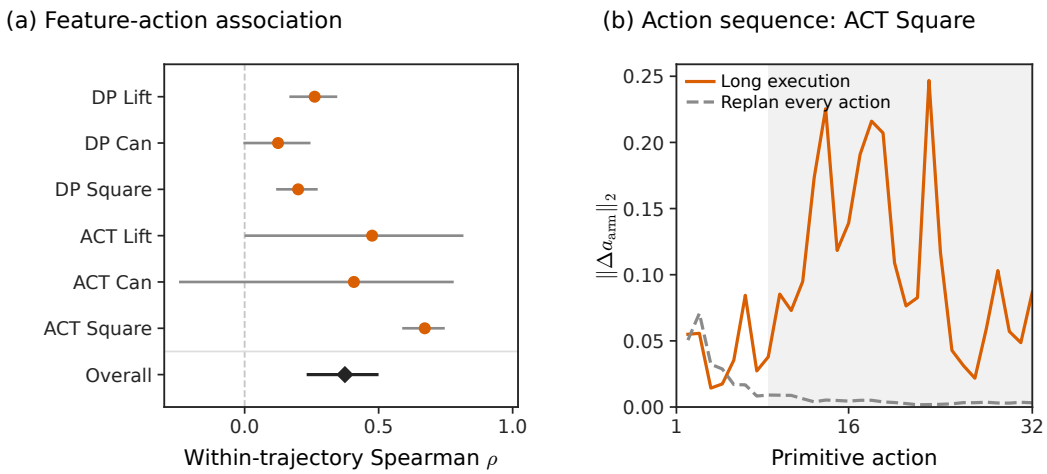


Figure 18: **Feature invariance and repeated first actions.** (a) Feature–action correlations with 95% episode-bootstrap intervals for DP and ACT. (b) Two branches from the same ACT `Square` state and initial chunk: executing all 32 actions or replanning after every action. In the shaded interval (actions 9–32), repeated replanning produces nearly unchanged first arm actions, while the original chunk continues to vary its commands.

parison illustrates the value of adaptive execution over either fixed choice. Together with Section 5.6, these observations suggest complementary roles for short and long executions: short executions enable corrections and new predictions, while longer executions preserve useful progression through a chunk.

Table 9: **Completion from states with repeated first actions.** Success counts from the same three selected ACT `Square` states; these are diagnostic cases, not benchmark success rates.

Base policy / task	Always $k = 1$	$k = 32$, then ExRL-Q	ExRL-Q
ACT / `Square`	1/3	1/3	2/3

Measurement details. For ACT, we use the first decoder token as the policy feature; for DP, we use the normalized conditioning input, which has no learned observation encoder. Feature changes are measured by ℓ_2 distance normalized by the mean feature norm, and arm-action changes by ℓ_2 distance excluding the gripper. The analysis contains 2,093 adjacent query pairs from 45 episodes. We compute Spearman correlations within each window and combine them in Fisher-z space within episodes, weighting by $n - 3$ for n valid pairs. Episodes and model–task pairs receive equal weight at their respective aggregation levels, and confidence intervals bootstrap episodes within each pair.

E Execution-Length Control as an Induced SMDP

In this section, we formalize the connection between ExRL and semi-Markov decision processes (SMDPs) (Sutton et al., 1999). Recall that the underlying environment is a primitive-time MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where each primitive action advances the environment by one timestep. ExRL does not change this environment. Instead, by freezing an action-chunked policy π_{base} and learning only how many actions from each predicted chunk to execute, ExRL induces a higher-level decision process over execution-length decisions.

Let $\mathbf{a} = a_{t:t+H}$ denote an action chunk sampled from the frozen base policy at state s, i.e., $\mathbf{a} \sim \pi_{\text{base}}(\cdot \mid s)$. The high-level decision state is the pair

$$x = (s, \mathbf{a}) \in \mathcal{X} := \mathcal{S} \times \mathcal{A}^H,$$

and the high-level action is the execution length $k \in \mathcal{K} = \{1, \dots, H\}$. Thus, ExRL induces an SMDP

$$\mathcal{M}_{\text{ex}} = (\mathcal{X}, \mathcal{K}, P_{\text{ex}}, r_{\text{ex}}, F_{\text{ex}}, \rho_{\text{ex}}, \gamma),$$

where $\mathcal{X}$ is the decision-state space, $\mathcal{K}$ is the execution-length action space, $P_{\text{ex}} : \mathcal{X} \times \mathcal{K} \to \Delta(\mathcal{X})$ is the transition kernel over execution-level decision states, $r_{\text{ex}} : \mathcal{X} \times \mathcal{K} \to \mathbb{R}$ is the reward function for one execution-length decision, $F_{\text{ex}} : \mathcal{X} \times \mathcal{K} \to \Delta(\mathbb{N})$ is the duration distribution over primitive timesteps, $\rho_{\text{ex}} \in \Delta(\mathcal{X})$ is the initial decision-state distribution, and γ is the primitive-time discount factor inherited from the underlying MDP.

The initial decision-state distribution is induced by the environment initial state distribution and the frozen base policy:

$$s_0 \sim \rho_0, \qquad \mathbf{a}_0 \sim \pi_{\text{base}}(\cdot \mid s_0), \qquad x_0 = (s_0, \mathbf{a}_0) \sim \rho_{\text{ex}}.$$

Given a decision state $x = (s_t, \mathbf{a}_t)$ and an execution length k_t, the environment executes the prefix of the chunk for k_t primitive timesteps:

$$a_t, a_{t+1}, \dots, a_{t+k_t-1}.$$

This produces a trajectory segment $(s_t, a_t, r_t, \dots, s_{t+k_t})$ under the primitive dynamics P. The reward of the high-level decision is the discounted reward accumulated along this segment,

$$r_{\text{ex}}(x, k_t) = \mathbb{E}\left[\sum_{i=0}^{k_t-1} \gamma^i r(s_{t+i}, a_{t+i}) \,\middle|\, x, k_t\right].$$

After the prefix execution ends, the base policy is queried again at the new state s_{t+k_t} to produce the next chunk $\mathbf{a}' \sim \pi_{\text{base}}(\cdot \mid s_{t+k_t})$. Therefore, the next high-level decision state is

$$x' = (s_{t+k_t}, \mathbf{a}').$$

Let K denote the realized duration of executing length k_t. Starting from $x = (s_t, \mathbf{a}_t)$, the primitive dynamics generate states $s_{t+1}, \dots, s_{t+K}$ by executing the prefix of $\mathbf{a}_t$ until either k_t actions have been executed or the episode terminates. The duration distribution is

$$F_{\text{ex}}(K \mid x, k_t) = \Pr(K \text{ primitive timesteps elapse before the next decision} \mid x, k_t).$$

In the non-terminal case, $F_{\text{ex}}(K \mid x, k_t) = \mathbb{1}\{K = k_t\}$; if the episode terminates early, then $K < k_t$ is the actual number of primitive steps executed and no bootstrap term is used.

The transition kernel over decision states is

$$P_{\text{ex}}(x' \mid x, k_t) = \Pr(s_{t+K} = s',\ \mathbf{a}' \sim \pi_{\text{base}}(\cdot \mid s') \mid x, k_t), \qquad x' = (s', \mathbf{a}').$$

That is, P_{ex} is induced by first rolling out the primitive dynamics under the selected prefix, and then querying the frozen base policy at the resulting state to form the next decision state.

For an execution-length policy $\pi_{\text{ex}}(k \mid x)$, the SMDP Bellman equation is

$$Q^{\pi_{\text{ex}}}(x, k) = \mathbb{E}\left[R + \gamma^K V^{\pi_{\text{ex}}}(x') \,\middle|\, x, k\right],$$

where

$$R = \sum_{i=0}^{K-1} \gamma^i r(s_{t+i}, a_{t+i}), \qquad V^{\pi_{\text{ex}}}(x') = \mathbb{E}_{k' \sim \pi_{\text{ex}}(\cdot \mid x')}\left[Q^{\pi_{\text{ex}}}(x', k')\right],$$

and K is the realized duration of the high-level execution-length decision. When the episode does not terminate during the prefix execution, $K = k$.

This recovers the duration-aware TD target used in ExRL. Writing $x = (s_t, a_{t:t+H})$, $x' = (s', a')$, and $K = k_t$, the target becomes

$$y_t = \sum_{i=0}^{k_t-1} \gamma^i r(s_{t+i}, a_{t+i}) + \gamma^{k_t} V(s', a'),$$

where V is instantiated by the particular RL algorithm, such as a greedy value for ExRL-Q or a soft value for ExRL-SAC. Thus, the only difference from a standard one-step Bellman backup is that the selected high-level action k_t determines the duration of the transition. As a result, k_t affects not only the accumulated reward and next decision state, but also the bootstrap discount through the factor γ^{k_t}.