Reinforcement Learning with Action Chunking

Qiyang Li, Zhiyuan Zhou, Sergey Levine
UC Berkeley
{qcli,zhiyuan_zhou,svlevine}@eecs.berkeley.edu

Abstract

We present **Q-chunking**, a simple yet effective recipe for improving reinforcement learning (RL) algorithms for long-horizon, sparse-reward tasks. Our recipe is designed for the offline-to-online RL setting, where the goal is to leverage an offline prior dataset to maximize the sample-efficiency of online learning. Effective exploration and sample-efficient learning remain central challenges in this setting, as it is not obvious how the offline data should be utilized to acquire a good exploratory policy. Our key insight is that action chunking, a technique popularized in imitation learning where sequences of future actions are predicted rather than a single action at each timestep, can be applied to temporal difference (TD)-based RL methods to mitigate the exploration challenge. Q-chunking adopts action chunking by directly running RL in a 'chunked' action space, enabling the agent to (1) leverage temporally consistent behaviors from offline data for more effective online exploration and (2) use unbiased n-step backups for more stable and efficient TD learning. Our experimental results demonstrate that Q-chunking exhibits strong offline performance and online sample efficiency, outperforming prior best offline-to-online methods on a range of long-horizon, sparse-reward manipulation tasks.

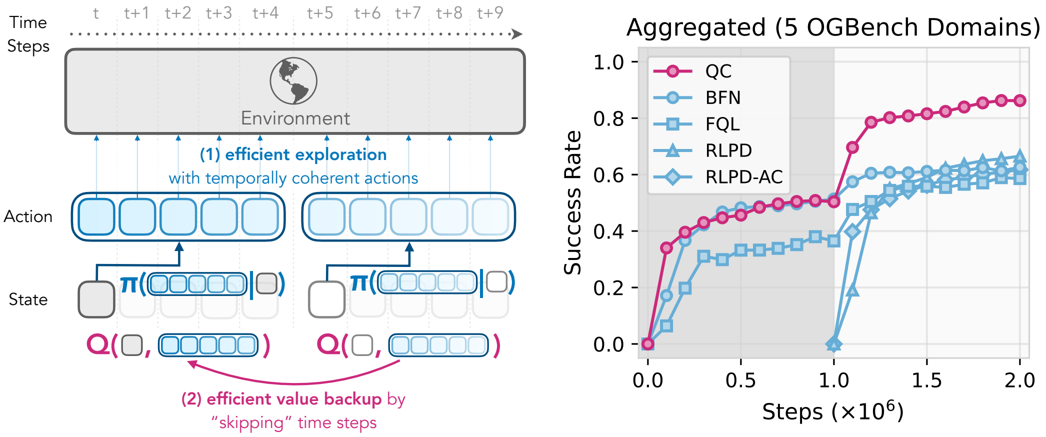


Figure 1: **Q-chunking uses action chunking to enable fast value backups and effective exploration with temporally coherent actions.** *left:* an overview of our approach: Q-chunking operates in a temporally extended action space that allows for (1) efficient value backups and (2) effective exploration via temporally coherent actions; *right:* Our method (**QC**) first pre-trains on an offline dataset for 1M steps (grey) and then updates with online data for another 1M steps (white). Our method achieves strong aggregated performance over five challenging long-horizon sparse-reward domains in OGBench. **Code:** `github.com/ColinQiyangLi/qc`.

First Exploration in AI Today Workshop at ICML (EXAIT at ICML 2025).

1 Introduction

Reinforcement learning (RL) holds the promise of solving any given task based only on a reward function. However, this simple and direct formulation of the RL problem is often impractical: in complex environments, exploring entirely from scratch to learn an effective policy can be prohibitively expensive, as it requires the agent to successfully solve the task through random chance before learning a good policy. Indeed, even humans and animals rarely solve new tasks entirely from scratch, instead leveraging prior knowledge and skills from past experience. Inspired by this, a number of recent works have sought to incorporate prior offline data into online RL exploration [26, 35, 77]. But this poses a new set of challenges: the distribution of offline data might not match the policy that the agent should follow online, introducing distributional shift, and it is not obvious how the offline data should be leveraged to acquire a good online *exploratory* policy.

In the adjacent field of imitation learning (IL), a widely used approach in recent years has been to employ *action chunking*, where instead of training policies to predict a single action based on the state observation from prior data, the policy is instead trained to predict a short sequence of future actions (an "action chunk") [82, 11]. While a complete explanation for the effectiveness of action chunking in IL remains an open question, its effectiveness can be at least partially ascribed to better handling of non-Markovian behavior in the offline data, essentially providing a more powerful tool for modeling the kinds of complex distributions that might occur in (for example) human-provided demonstrations or mixtures of different behaviors [82]. Action chunking has not been used widely in RL, perhaps because optimal policies in fully observed MDPs are Markovian [68], and therefore chunking may appear unnecessary.

We make the observation that, though we might desire a final optimal Markovian policy, the exploration problem can be better tackled with non-Markovian and temporally extended skills, and that action chunking offers a very simple and convenient recipe for obtaining this. Furthermore, action chunking provides a better way to leverage offline data (with a better handling of non-Markovian behavior in the data), and even improves the stability and efficiency of TD-based RL, by enabling unbiased n-step backups (where n matches the length of the chunk). Thus, in combination with pretraining on offline data, action chunking offers a compelling and very simple way to mitigate the exploration challenge in RL.

We present Q-learning with action chunking (or **Q-chunking** in short), a generic recipe for improving TD-based actor-critic RL algorithms in the offline-to-online RL setting (Figure 1). The key idea is to run RL at an action sequence level — (1) the policy predicts a sequence of actions for the next h steps and executes them one-by-one open loop, and (2) the critic takes in the current state and a sequence of actions and estimates the value of carrying out the whole sequence rather than a single action. The benefits of operating RL on this extended action space are two-fold: (1) the policy can be optimized to generate temporally coherent actions by regularizing it towards some prior behavior data that exhibit such coherency, (2) the critic trained with a standard TD-backup loss is effectively performing n-step backups, with no off-policy bias (that typically occurs in naïve n-step return methods), since the critic takes the full action sequence into account.

Our main contribution is **QC**, a practical offline-to-online RL algorithm that is instantiated from our **Q-chunking** recipe. QC is simple to implement, requiring only training (1) an action chunking behavior policy using a standard flow-matching loss, and (2) a temporally extended critic with the standard TD-loss. QC achieves strong performance on a range of six challenging long-horizon, sparse-reward domains, outperforming prior offline-to-online methods. Moreover, we highlight that Q-chunking is a generic recipe that can be applied to many existing offline-to-online algorithms with minimal modification. In this work, we demonstrate one such instantiation by applying it to **FQL** [53], resulting in **QC-FQL**, which shows significant improvements over the original method.

2 Related Work

Offline-to-online reinforcement learning methods focus on leveraging prior offline data to accelerate reinforcement learning online [80, 64, 34, 1, 81, 83, 7, 46, 84, 35]. The simplest way to tackle offline-to-online RL is to use an existing offline RL algorithm to first pretrain on the offline data and then use the same offline optimization objective to continue training online using a growing dataset that combines the original offline data and the replay buffer data [45, 33, 30, 70, 53, 2, 38, 34]. While straightforward, this naïve approach often result in overly pessimistic that hinders exploration and consequently the online sample-efficiency. Several prior works have attempted to address this issue by adjusting the degree of pessimism online [84, 46, 38, 34, 75]. However, these approaches can be difficult to tune and sometimes stills fall short in online sample efficiency compared to a simple, well-regularized online RL algorithm learning from scratch on both offline data and online replay buffer data [7]. Our approach takes a step towards improving the sample efficiency of offline-to-online RL methods via value backup acceleration and temporally coherent exploration.

Action chunking is a technique popularized by roboticists for imitation learning (IL), where the policy predicts and executes a sequence of actions in an open-loop manner ("an action chunk") [82]. Action chunking has been shown to improve policy robustness [82, 21, 8], and handle non-Markovian behavior in offline data [82]. Existing RL methods that incorporate action chunking typically focus on fine-tuning a policy pre-trained with imitation learning [56, 59]. Tian et al. [71] propose to learn a critic on action chunks by integrating n-step returns with a transformer. However, their method only applies chunking to the critic, while still optimizing a single-step actor.

Multi-step latent space planning and search is a technique commonly used in model-based RL methods where they use a learned model to optimize a short-horizon action sequence towards high-return trajectories [48, 58]. These approaches work by training a dynamics model on an encoded latent space, where the model takes in a latent state and an action to predict the next latent state and the associated reward value. This latent dynamics model, along with a value network on the latent state, can then provide an estimate of the Q-value on-the-fly for any given action sequence starting from a given latent state by simply simulating the action sequence in the latent dynamics model. In contrast, we do not learn a latent dynamics model and instead train a Q-network to directly estimate the value of the action sequence. Lastly, these approaches operate in the purely online RL setting whereas we focus on the offline-to-online RL setting.

We include more discussions of the related work for temporal coherency and hierarchical RL in Appendix F.4.

3 Background

Offline-to-online RL. In this paper, we consider an infinite-horizon, fully observable Markov decision process (MDP), $(\mathcal{S}, \mathcal{A}, \rho, T, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T(s'|s,a) : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$ is the transition kernel, $r(s,a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, $\rho : \Delta(\mathcal{S})$ is the initial state distribution and $\gamma \in [0, 1)$ is the discount factor. We also assume there is a prior offline dataset $\mathcal{D}$ that consists of transitions rollouts $\{(s, a, s', r)\}$ from $\mathcal{M}$. The goal of offline-to-online RL is to find a policy $\pi(a|s) : \mathcal{S} \mapsto \Delta(\mathcal{A})$ that maximizes the expected discounted cumulative reward (or discounted return): $\eta(\pi) := \mathbb{E}_{s_{t+1}\sim T(s_t,a_t), a_t\sim\pi(\cdot|s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$. Oftentimes, offline-to-online RL algorithms operate in two distinct phases: an offline phase where a policy is pretrained on the offline data $\mathcal{D}$ and an online phase where the policy is further fine-tuned online with environment interactions. Our approach follows the same regime.

Temporal difference and multi-step return. TD-based RL algorithms typically learn $Q_\theta(s, a)$ to approximate the maximum expected discounted cumulative reward that a policy can receive starting from state s and action a by using a temporal difference (TD) loss [68]:

$$L(\theta) = \left[Q_\theta(s_t, a_t) - \hat{V}\right]^2, \tag{1}$$

where $\hat{V}$ is an estimate of $Q(s_t, a_t)$ that is commonly chosen as $\hat{V}_{\text{1-step}}$:

$$\hat{V}_{\text{1-step}} := r_t + \gamma Q_{\bar{\theta}}(s_{t+1}, a_{t+1}), a_{t+1} \sim \pi_\psi(\cdot|s_{t+1}), \tag{2}$$

and s_t, a_t, s_{t+1}, r are sampled from some off-policy trajectories and $\bar{\theta}$ is a delayed version of θ that does not allow the gradient to pass through for learning stability. When the TD error is minimized, the Q_θ converges to the expected discounted value of the policy π_ψ. As the effective horizon $\tilde{H} = 1/(1-\gamma)$ goes up, the learning slows down as the value only propagates 1 step backward (from s_{t+1} to s_t). To speed-up long-horizon value backup, a common strategy is to sample a length-n trajectory segment, $(s_t, a_t, s_{t+1}, \cdots, a_{t+n-1}, s_{t+n})$, and construct a n-step return from it [76, 68]:

$$\hat{V}_{\text{n-step}} := \sum_{t'=t}^{t+n-1} \left[\gamma^{t'-t} r'_t\right] + Q_{\bar{\theta}}(s_{t+n}, a_{t+n}), a_{t+n} \sim \pi_\psi(\cdot|s_{t+n}), \tag{3}$$

where again $r_t = r(s_t, a_t)$. This value estimate of $Q(s_t, a_t)$ allows for a n times speed-up in terms of the number of time steps that the value can propagate back across. This estimator is sometimes referred to as the *uncorrected n-step return estimator* [16, 31] because it is biased when the data collection policy is different from the current policy π_ψ. Nevertheless, due to the implementation simplicity of n-step return, it has been commonly adopted in large-scale RL systems [43, 24, 27, 78].

4 Q-Chunking

In this section, we first describe two main design principles of **Q-chunking**: (1) Q-learning on a temporally extended action space (the space of chunks of actions), and (2) behavior constraint in this extended action space, followed by practical implementations of Q-chunking (**QC**, **QC-FQL**) as effective TD-based offline-to-online RL algorithms.

4.1 Q-learning on a temporally extended action space

The first design principle of Q-chunking is to apply Q-learning on the temporally extended action space. Unlike normal 1-step TD-based actor-critic methods, which train a Q-function $Q(s_t, a_t)$ and a policy $\pi(a_t|s_t)$, we instead train both the critic and the actor with a span of h consecutive actions: [1]

$$\textit{Q-Chunking Policy: } \pi_\psi(\boldsymbol{a}_{t:t+h}|s_t) := \pi_\psi(a_t, a_{t+1}, \cdots, a_{t+h-1}|s_t)$$

$$\textit{Q-Chunking Critic: } Q_\theta(s_t, \boldsymbol{a}_{t:t+h}) := Q_\theta(s_t, a_t, a_{t+1}, \cdots, a_{t+h-1})$$

In practice, this involves updating the critic and the actor on batches of transitions consisting of a random state s_t, an action sequence followed by the state $\boldsymbol{a}_t$, and the state h steps into the future, s_{t+h}. Specifically, we train Q_θ with the following TD loss,

$$L(\theta) = \mathbb{E}_{s_t, \boldsymbol{a}_{t:t+h}, s_{t+h} \sim \mathcal{D}} \left[\left(Q_\theta(s_t, \boldsymbol{a}_{t:t+h}) - \sum_{t'=1}^{h} \gamma^{t'} r_{t+t'} - \gamma^h Q_{\bar{\theta}}(s_{t+h}, \boldsymbol{a}_{t+h:t+2h}) \right)^2 \right] \tag{4}$$

with $\boldsymbol{a}_{t+h:t+2h} \sim \pi_\psi(\cdot|s_{t+h})$, and $\bar{\theta}$ being the target network parameters that are often an exponential moving average of θ [23].

The TD loss above shares striking similarity to the n-step return in Equation 3 (with n matches h) but with a crucial difference — the Q-function used in the n-step return backup takes in only one action (at time step t) whereas our Q-function takes in the whole action sequence. The implication of this difference can be best explained after we write out the TD backup equations for standard 1-step TD, n-step return, and Q-chunking:

$$Q(s_t, a_t) \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1}) \qquad \text{(standard 1-step TD)} \tag{5}$$

$$Q(s_t, a_t) \leftarrow \underbrace{\sum_{t'=t}^{t+h-1} \left[\gamma^{t'-t} r_{t'}\right]}_{\text{biased}} + \gamma^h Q(s_{t+h}, a_{t+h}), \qquad (n\text{-step return, } n = h) \tag{6}$$

$$Q(s_t, \boldsymbol{a}_{t:t+h}) \leftarrow \underbrace{\sum_{t'=t}^{t+h-1} \left[\gamma^{t'-t} r_{t'}\right]}_{\text{unbiased}} + \gamma^h Q(s_{t+h}, \boldsymbol{a}_{t+h:t+2h}). \qquad \textbf{(Q-chunking)} \tag{7}$$

For the standard 1-step TD, each backup step propagates the value back by only 1 time step. n-step return propagates the value back $h\times$ faster, but can suffer from a *biased* value estimation issue when $\boldsymbol{s}_{t:t+h}$ and $\boldsymbol{a}_{t:t+h}$ are off-policy [16]. This is because the discounted sum of the n-step rewards $\boldsymbol{r}_{t:t+h}$ from the dataset or replay buffer is no longer an unbiased estimate of the expected n-step rewards under the current policy π. Q-chunking value backup is similar to the n-step return where each step also propagates the value back by h time steps, but *does not* suffer from this biased estimation issue. Unlike n-step return where we are propagating the value to a 1-step Q-function, Q-chunking backup propagates the value back to a h-step Q-function that takes in the exact same actions that are taken to obtain the n-step rewards $\boldsymbol{r}_{t:t+h}$, eliminating the biased value estimation. As a result, Q-chunking value backup enjoys the value propagation speedup while maintaining an unbiased value estimate.

[1] We use $\boldsymbol{a}_{t:t+h}$ to denote a concatenation of h consecutive actions: $\begin{bmatrix} a_t & a_{t+1} & \cdots & a_{t+h-1} \end{bmatrix} \in \mathbb{R}^{Ah}$ for notation convenience. This is similar for $\boldsymbol{s}_{t:t+h}$ and $\boldsymbol{r}_{t:t+h}$.

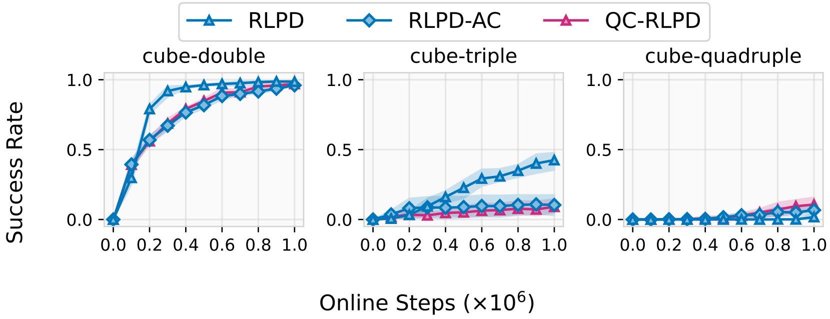


Figure 2: **Naïvely using action chunking for online RL methods can help on some domains but hurt on others.** *(1)* **RLPD** runs online RL on both offline data and online replay buffer [7]. *(2)* **RLPD-AC** is the same algorithm as RLPD but operates in a temporally extended action space (action chunk size of 5). *(3)* **QC-RLPD** additionally uses a behavior cloning loss on the actor (5 seeds).

4.2 Behavior constraints for temporally coherent exploration

The second design principle of Q-chunking addresses the action incoherency issues by leveraging a behavior constraint in the objective for the π_ψ:

$$L(\psi) = -\mathbb{E}_{s_t\sim\mathcal{D},\boldsymbol{a}_{t:t+h}\sim\pi_\psi(\cdot|s_t)}\left[Q_\theta(s_t,\boldsymbol{a}_{t:t+h})\right], \text{s.t. } D(\pi_\psi(\boldsymbol{a}_{t:t+h}|s_t),\pi_\beta(\boldsymbol{a}_{t:t+h}|s_t)) \leq \varepsilon \quad (8)$$

where we denote $\pi_\beta(\boldsymbol{a}_{t:t+h}|s_t)$ as the behavior distribution in the offline data $\mathcal{D}$, and D as some distance metric that measures how different the learned policy π deviates from π_β.

Intuitively, a behavior constraint on the temporally extended action sequence allows us to leverage temporally coherent action sequences in the offline dataset. This is a particularly advantageous thing to do in the temporally extended action space compared to in the original action space because offline data often exhibit non-Markovian structure (e.g., from scripted policies [51], human tele-operators [39], or noisy expert policies for sub-tasks [51, 19]) that cannot be well captured by a Markovian behavior constraint. Temporally coherent actions are desirable for online exploration because they resemble temporally extended skills (e.g., moving in a certain direction for navigation, jumping motions for going over obstacles) that help traverse the environment in a structured way rather than using random actions that often result in data that is localized near the initial states. Imposing behavior constraint for an action chunking policy is a very simple way to approximately extract skills without the need of training policy with bi-level structure as often necessitated by skill-based methods (see more discussion in Section 2). In reality, we do see that Q-chunking, with such behavior constraints, can interact and explore the environment with temporally coherent actions (see Section 5.3), mitigating the exploration challenge in RL.

4.3 Practical implementations

The key implementation challenge of Q-chunking is to enforce a good behavior constraint that captures the non-Markovian behavior at the action sequence level. One of the prerequisites of imposing a good behavior constraint is the ability of the policy to capture the complex behavior distribution (e.g., using a flow/diffusion policy). A Gaussian policy, a default choice in online RL algorithms, would not suffice (Figure 2). To enforce a good behavior constraint, we first use flow-matching objective [37] to train a behavior cloning flow policy to capture the behavior distribution. The flow policy is parameterized by a state-conditioned velocity field prediction model $f(s,\boldsymbol{z},u):\mathcal{S}\times\mathbb{R}^{Ah}\times[0,1]\mapsto\mathbb{R}^{Ah}$ and we denote $f_\xi(\cdot|s)$ as the action distribution that the flow policy parameterizes as an approximation of the true behavior distribution in the offline prior data ($f_\xi \approx \pi_\beta$). Now we are ready to present our main method, **QC**:

We consider a KL constraint on our policy through the learned behavior distribution:

$$D_{\mathrm{KL}}(\pi_\psi \| f_\xi(\cdot|s)) \leq \varepsilon \quad (9)$$

While it is possible include the KL as part of the loss, estimating the KL divergence or log probability for flow models is practically challenging. Instead, we use best-of-N sampling [66] to maximize Q-value while imposing this KL constraint implicitly altogether. Practically, this involves first sampling N action chunks from the learned behavior policy $f_\xi(\cdot|s_t)$,

$$\{\boldsymbol{a}^1, \boldsymbol{a}^2, \cdots, \boldsymbol{a}^N\} \sim f_\xi(\cdot|s),$$

and then picking the action chunk sample that maximizes the temporally extended Q-function:

$$\boldsymbol{a}^\star \leftarrow \arg\max_{\boldsymbol{a}\in\{\boldsymbol{a}^1,\boldsymbol{a}^2,\cdots,\boldsymbol{a}^N\}} Q(s, \boldsymbol{a})$$

It has been shown in prior work that best-of-N sampling admits a closed-form upper-bound on the KL divergence from the original distribution [25]:

$$D_{\mathrm{KL}}(\boldsymbol{a}^\star \| f_\xi(\cdot|s)) \le \log N - \frac{N-1}{N}, \tag{10}$$

which approximately satisfies KL constraint implicitly (Equation 9). Tuning the value of N directly corresponds to the strength of the constraint.

Since we approximate the policy optimization (Equation 8) with the best-of-N sampling, we can completely avoid separately parameterizing a policy π_ψ. In particular, we use the best-of-N sampling to generate actions to both (1) interact with the environment, and (2) provide the action samples in the TD backup following Ghasemipour et al. [22]. As a result, our algorithm has only one additional loss function:

$$L(\theta) = \mathbb{E}_{\substack{s_t, \boldsymbol{a}_t \sim D \\ \{\boldsymbol{a}^i_{t+h}\}_{i=1}^N \sim f_\xi(\cdot|s_{t+h})}} \left[\left(Q_\theta(s_t, \boldsymbol{a}_t) - \sum_{t'=1}^{h} \gamma^{t'} r_{t+t'} - \gamma^h Q_{\bar\theta}(s_{t+h}, \boldsymbol{a}^\star_{t+h}) \right)^2 \right] \tag{11}$$

where again $\boldsymbol{a}^\star_{t+h} := \arg\max_{\boldsymbol{a}\in\{\boldsymbol{a}^i_{t+h}\}} Q(s, \boldsymbol{a})$.

While our method is simple and easy to implement, it does come with some additional computational costs (sampling $N\times$ action chunks). We include a variant of our method that leverages a cheaper off-the-shelf offline/offline-to-online RL method, **FQL** [53], in Appendix C.

Offline-to-online RL considerations. Since both variants of our methods use beahvior constraint (implicit KL for QC, explicit W_2 for QC-FQL), we can also directly run them for offline RL pre-training, which provides further sample efficiency gain. For both offline and online training, we use the same behavior strength (e.g., N for QC and α for QC-FQL). For offline training, we use the same algorithm and simply remove the environment interaction part.

5 Experimental Results

We conduct a series of experiments to analyze the empirical effectiveness of our method on a range of long-horizon, sparse-reward domains. In particular, we are going to answer the following questions:

- **(Q1)** *How well do Q-chunking methods perform compared to prior offline-to-online RL methods?*
- **(Q2)** *Why does action chunking helps online learning?* (Appendix A.1)
- **(Q3)** *How does chunk length, critic ensemble size, and update-to-data ratio affect performance?* (Appendix A.2)

5.1 Environments and Datasets

We first consider six sparse reward robotic manipulation domains with tasks of varying difficulties. This includes 5 domains from OGBench [50], `scene-sparse`, `puzzle-3x3-sparse`, `cube-double/triple/quadruple` (5 tasks each) and 3 tasks in the Robomimic benchmark [39]. For OGBench, we use the default play-style datasets except for `cube-quadruple` where we use a large 100M-size dataset. For robomimic, we use the multi-human datasets. See more details of these environments and datasets in Appendix B.

5.2 Comparisons

We primarily compare with prior methods that speedup value backup as well as the previous best offline-to-online RL methods.

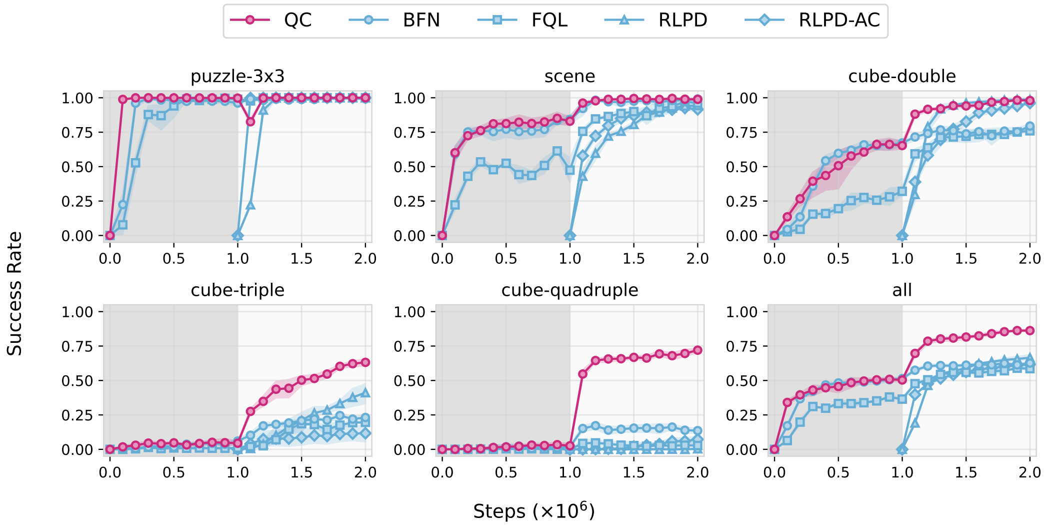


Figure 3: **Aggregated performance per OGBench domain.** Our method, **QC**, achieves strong performance across all five challenging OGBench domains. We also include an aggregation performance plot for all the domains at the bottom right. The first 1M steps are offline training and the next 1M steps are online training with one environment step per training step (5 seeds per task; 5 tasks per domain).

BFN (best-of-n) is a baseline that we propose to combine the expected-max Q operator [22] with an expressive behavior flow policy. BFN operates in the original action space and implicitly parameterizes a policy by sampling multiple actions from the behavior flow policy and picking the one that maximizes the current Q-value. This baseline is a direct ablation for the effect of Q-chunking from **QC**.

FQL [53] is a recently proposed offline RL method that achieves strong offline and offline-to-online RL performance. This baseline is a direct ablation for the effect of Q-chunking from **QC-FQL**.

BFN-n/FQL-n. These baselines are the same as BFN/FQL but uses n-step backup with $n = 1$ (Equation 3) instead of the standard 1-step TD backup. This baseline enjoys the benefits of value backup speedup, but does not use chunked critic or actor, and potentially suffer from the bias issue..

RLPD [7], **RLPD-AC**. RLPD is a sample-efficient RL algorithm that treats offline data as additional off-policy data and learn from scratch online. RLPD-AC is the same as RLPD but operates on the temporally extended action space. Both of them do not use a behavior constraint.

See more implementation details of our method and the baselines in Appendix D.

5.3 How well does our method compare to prior offline-to-online RL methods?

We report the aggregated performance of Q-chunking and the baselines for each of the five OGBench domains (Figure 3) and individual performance on three robomimic tasks (Appendix F, Figure 15). **QC** achieves competitive performance offline (in grey), often matching or sometimes outperforming best prior methods. In the online phase (in white), **QC** shows strong sample-efficiency, especially on the two hardest OGBench domains (`cube-triple/quadruple`), where it outperforms all prior methods (especially on `cube-quadruple`) by a large margin. We also conduct an ablation study where we compare **QC** with a variant of our method **QC-FQL** and two n-step return baselines (**BFN-n** and **FQL-n**) in Figure 4. The n-step return baselines, which do not leverage a temporally extended critic or policy, perform significantly worse than our methods (**QC** and **QC-FQL**). In fact, they often underperform even the 1-step baselines (**BFN** and **FQL**), highlighting the importance of learning in the temporally extended action space.

Acknowledgments

This research was partially supported by RAI, and ONR N00014-22-1-2773. This research used the Savio computational cluster resource provided by the Berkeley Research Computing program at UC Berkeley. We would like to thank Seohong Park for providing the code infrastructure and the 100M dataset for `cube-quadruple`. We would also like to thank Ameesh Shah, Chuer Pan, Oleg Rybkin, Andrew Wagenmaker, Seohong Park, Yifei Zhou, Fangchen Liu, William Chen for discussions on the method and feedback on the early draft of the paper.

References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 28955–28971. Curran Associates, Inc., 2022.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. *Advances in neural information processing systems*, 35:28955–28971, 2022.

[3] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. OPAL: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=V69LGwJ0lIN.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[5] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2019.

[6] Akhil Bagaria, Ben Abbatematteo, Omer Gottesman, Matt Corsaro, Sreehari Rammohan, and George Konidaris. Effectively learning initiation sets in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[7] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[8] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4788–4795. IEEE, 2024.

[9] Boyuan Chen, Chuning Zhu, Pulkit Agrawal, Kaiqing Zhang, and Abhishek Gupta. Self-supervised reinforcement learning that transfers using random features. *Advances in Neural Information Processing Systems*, 36, 2024.

[10] Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17, 2004.

[11] 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*, page 02783649241273668, 2023.

[12] Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34:21847–21859, 2021.

[13] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 17(93):1–50, 2016.

[14] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

[15] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[16] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International conference on machine learning*, pages 3061–3071. PMLR, 2020.

[17] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

[18] Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Unsupervised zero-shot reinforcement learning via functional reward encodings. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 13927–13942. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/frans24a.html.

[19] 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.

[20] Jonas Gehring, Gabriel Synnaeve, Andreas Krause, and Nicolas Usunier. Hierarchical skills for efficient exploration. *Advances in Neural Information Processing Systems*, 34:11553–11564, 2021.

[21] Abraham George and Amir Barati Farimani. One act play: Single demonstration behavior cloning with action chunking transformers. *arXiv preprint arXiv:2309.10175*, 2023.

[22] Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, pages 3682–3691. PMLR, 2021.

[23] 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*, pages 1861–1870. PMLR, 2018.

[24] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[25] Jacob Hilton. Kl divergence of max-of-n, 2023. URL https://www.jacobh.co.uk/bon_kl.pdf.

[26] Hao Hu, Yiqin Yang, Jianing Ye, Ziqing Mai, and Chongjie Zhang. Unsupervised behavior extraction via random intent priors. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=4vGVQVz5KG.

[27] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[28] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. *Advances in Neural Information Processing Systems*, 32, 2019.

[29] George Dimitri Konidaris. *Autonomous robot skill acquisition*. University of Massachusetts Amherst, 2011.

[30] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[31] Tadashi Kozuno, Yunhao Tang, Mark Rowland, Rémi Munos, Steven Kapturowski, Will Dabney, Michal Valko, and David Abel. Revisiting peng's q (λ) for modern reinforcement learning. In *International Conference on Machine Learning*, pages 5794–5804. PMLR, 2021.

[32] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[33] 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.

[34] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic Q-ensemble. In *Conference on Robot Learning*, pages 1702–1712. PMLR, 2022.

[35] Qiyang Li, Jason Zhang, Dibya Ghosh, Amy Zhang, and Sergey Levine. Accelerating exploration with unlabeled prior data. *Advances in Neural Information Processing Systems*, 36, 2024.

[36] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[37] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[38] Yicheng Luo, Jackie Kay, Edward Grefenstette, and Marc Peter Deisenroth. Finetuning from offline reinforcement learning: Challenges, trade-offs and practical solutions. *arXiv preprint arXiv:2303.17396*, 2023.

[39] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[40] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71, 2004.

[41] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *Machine Learning: ECML 2002: 13th European Conference on Machine Learning Helsinki, Finland, August 19–23, 2002 Proceedings 13*, pages 295–306. Springer, 2002.

[42] Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. *arXiv preprint arXiv:1811.11711*, 2018.

[43] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PmLR, 2016.

[44] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[45] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[46] Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-QL: Calibrated offline RL pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

[47] Soroush Nasiriany, Tian Gao, Ajay Mandlekar, and Yuke Zhu. Learning and retrieval from prior data for skill-based imitation learning. In *Conference on Robot Learning*, 2022.

[48] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *Advances in neural information processing systems*, 30, 2017.

[49] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. *Advances in neural information processing systems*, 26, 2013.

[50] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *ArXiv*, 2024.

[51] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

[52] Seohong Park, Tobias Kreiman, and Sergey Levine. Foundation policies with hilbert representations. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=LhNsSaAKub.

[53] Seohong Park, Qiyang Li, and Sergey Levine. Flow Q-learning. *arXiv preprint arXiv:2502.02538*, 2025.

[54] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *Acm transactions on graphics (tog)*, 36(4):1–13, 2017.

[55] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR, 2021.

[56] Allen Z Ren, Justin Lidard, Lars L Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

[57] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pages 4344–4353. PMLR, 2018.

[58] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[59] Younggyo Seo and Pieter Abbeel. Reinforcement learning with action sequence for data-efficient robot learning. *arXiv preprint arXiv:2411.12155*, 2024.

[60] Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pages 8624–8633. PMLR, 2020.

[61] Özgür Şimşek and Andrew G Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 95, 2004.

[62] Özgür Şimşek and Andrew G. Barto. Betweenness centrality as a basis for forming skills. Workingpaper, University of Massachusetts Amherst, April 2007.

[63] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ysuv-WOFeKR.

[64] Yuda Song, Yifei Zhou, Ayush Sekhari, Drew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid RL: Using both offline and online data can make RL efficient. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=yyBis80iUuU.

[65] Aravind Srinivas, Ramnandan Krishnamurthy, Peeyush Kumar, and Balaraman Ravindran. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*, 2016.

[66] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

[67] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

[68] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[69] 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.

[70] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[71] Dong Tian, Ge Li, Hongyi Zhou, Onur Celik, and Gerhard Neumann. Chunking the critic: A transformer-based soft actor-critic with n-step returns. *arXiv preprint arXiv:2503.03660*, 2025.

[72] Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does zero-shot reinforcement learning exist? In *The Eleventh International Conference on Learning Representations*, 2022.

[73] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. *Advances in neural information processing systems*, 29, 2016.

[74] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International conference on machine learning*, pages 3540–3549. PMLR, 2017.

[75] Shenzhi Wang, Qisen Yang, Jiawei Gao, Matthieu Lin, Hao Chen, Liwei Wu, Ning Jia, Shiji Song, and Gao Huang. Train once, get a family: State-adaptive balances for offline-to-online reinforcement learning. *Advances in Neural Information Processing Systems*, 36:47081–47104, 2023.

[76] Christopher John Cornish Hellaby Watkins et al. Learning from delayed rewards. 1989.

[77] Max Wilcoxson, Qiyang Li, Kevin Frans, and Sergey Levine. Leveraging skills from unlabeled prior data for efficient online exploration. *arXiv preprint arXiv:2410.18076*, 2024.

[78] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896): 223–228, 2022.

[79] Kevin Xie, Homanga Bharadhwaj, Danijar Hafner, Animesh Garg, and Florian Shkurti. Latent skill planning for exploration and transfer. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jXe91kq3jAq.

[80] Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. Policy finetuning: Bridging sample-efficient offline and online reinforcement learning. *Advances in neural information processing systems*, 34:27395–27407, 2021.

[81] Haichao Zhang, Wei Xu, and Haonan Yu. Policy expansion for bridging offline-to-online reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=-Y34L45JR6z.

[82] 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.

[83] Han Zheng, Xufang Luo, Pengfei Wei, Xuan Song, Dongsheng Li, and Jing Jiang. Adaptive policy learning for offline-to-online reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11372–11380, 2023.

[84] Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data. *arXiv preprint arXiv:2412.07762*, 2024.

A Additional Experiments

A.1 Why does action chunking help exploration?

We hypothesize in Section 4.2 that action chunking policy produce more temporally coherent actions and thus lead to better state coverage and exploration. In this section, we study to what degree that holds empirically. We first visualize the end-effector movements early and late in the training for **QC** and **BFN** (Figure 5 (left and center)). **BFN**'s trajectory contains many pauses (as indicated by a very big and dense cluster near the center of the visualization), especially when the end-effector is being lowered to pickup a cube. In contrast, **QC** has many fewer pauses (fewer and shallower clusters) and a more diverse state coverage in the end-effector space. We include additional examples in Appendix F, Figure 9 and Figure 10. To get a quantitative measure of the temporal coherency in the actions produced by the agent, we record a subset of the actions, the 3-dimensional end-effector position of the manipulator, throughout training every 5 time steps: $\{\boldsymbol{x}_0^{\text{eef}}, \boldsymbol{x}_5^{\text{eef}}, \cdots\}$ and compute the average L_2 norm of the difference vector of two adjacent end-effector positions $\|\boldsymbol{x}_t^{\text{eef}} - \boldsymbol{x}_{t+5}^{\text{eef}}\|_2$. This average norm would be small if there are any pauses or jittery motions, making a good proxy for measuring the temporal coherency in actions. As shown in Figure 5 (right), **QC** exhibits a higher action temporal coherency throughout training compared to **BFN**. This suggests that Q-chunking improves temporal coherency in actions, which explains the improved sample-efficiency that Q-chunking brings.

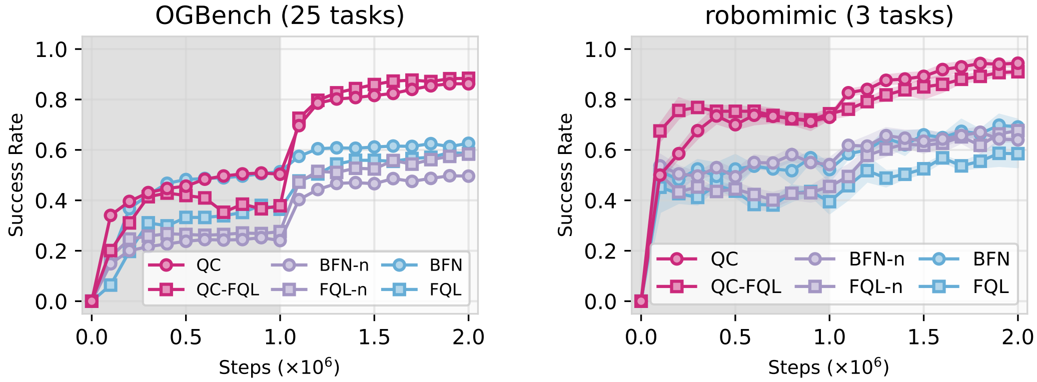


Figure 4: **QC-FQL and n-step return ablation on OGBench and robomimic. QC-FQL** obtains a similar performance compared to **QC**. QC is slightly better than QC-FQL on OGBench offline and Robomimic online, and slightly worse than QC-FQL on Robomimic offline. For an individual task breakdown, see Appendix F, Figure 12 and 14.

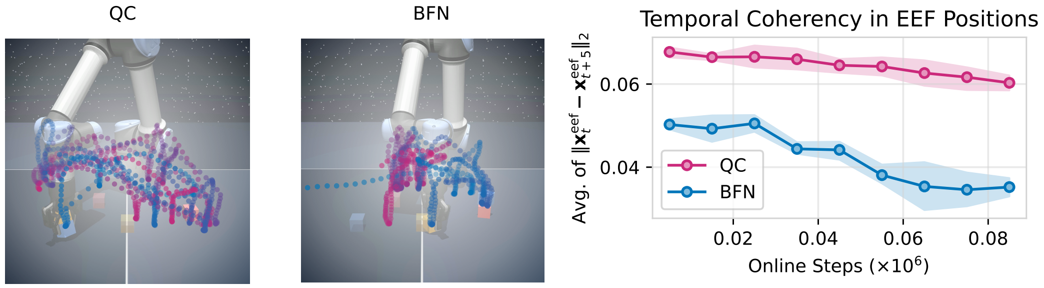


Figure 5: **End-effector movements early in the training and temporal coherency analysis on `cube-triple-task3`. QC** (left) covers a more diverse set of states compared to **BFN** (center) in the first 1000 environment steps. **QC** exhibits a higher temporal coherency in end-effector compared to **BFN** (right).

A.2 How does action chunk length, critic ensemble size affect the performance of our method?

In Figure 6 (left), we report the performance of **QC-FQL** with different action chunk sizes ($h \in \{5, 10, 25\}$) on the `cube-triple` domain. In general, a higher action chunk length helps but not significantly. We use $h = 5$ in all our other experiments as $h = 5$ is cheap to run. In Figure 6 (center), we study how the critic ensemble size affects the performance of our method. Using 10 critics improves both **QC** and **BFN**. We use $K = 2$ in our other experiments as it is cheap to run. Using $K = 10$ could potentially make Q-chunking perform much better on the benchmark tasks we consider. Finally, we observe that increasing the update-to-data ratio (UTD) does not improve the sample efficiency of **QC** (Figure 6, right).

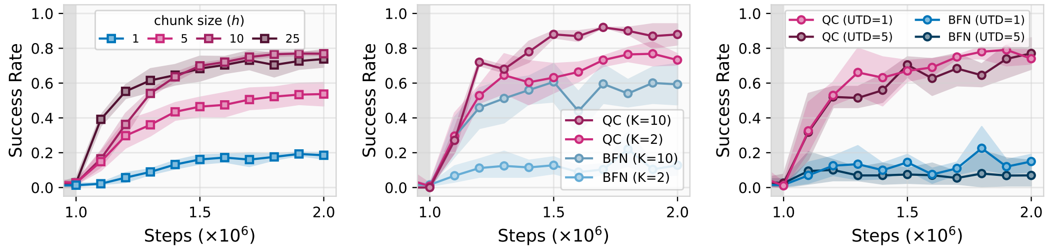


Figure 6: **Sensitivity analysis: action chunk size (h) and critic ensemble size (K).** *Left:* **QC-FQL** with different action chunks on all 5 `cube-triple` tasks (5 seeds). QC-FQL with $h = 1$ is equivalent to **FQL**. *Center:* Increasing the ensemble size to $K = 10$ improves performance of both **QC** and **BFN** on `cube-triple-task3` (5 seeds). *Right:* **QC** with update-to-data (UTD) ratio of 5 on `cube-triple-task3` (4 seeds). We report only the online phase results, as all methods achieve near-zero success rates during the offline phase.

A.3 How computationally efficient is Q-chunking?

In Figure 7, we report the runtime for our approach and our baselines on a representative task `cube-triple-task1`. In general, **QC-FQL** has a comparable run-time as our baselines (e.g., **FQL** and **RLPD**) for both offline and online. **QC** is slower for offline training as it requires sampling 32 actions for each training example for the agent update (**BFN** is faster because it only needs to sample 4 actions). For online training, we are doing one gradient update per environment step, and it makes **QC** only around 50% more expensive than other methods.

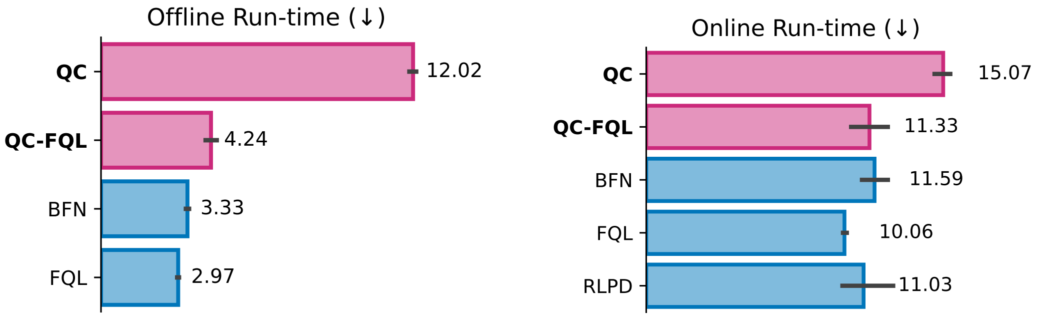


Figure 7: **How long does each method take for one step in milliseconds.** Left: offline. Right: online (one agent training step and an environment step). The runtime is measured using the default hyperparameters in our paper on `cube-triple-task1` on a single RTX-A5000.

B Domain Details

See an overview of the six domains we use in our experiments in Figure 8. We also include the dataset size, episode length and the action dimension in Table 1. In the following sections, we describe each domain in details.

B.1 OGBench environments.

We consider five manipulation domains from OGBench [50] and take the publicly available single-task versions of it in our experiments. For `scene-sparse` and `puzzle-3x3-sparse`, we sparsify the reward function such that the reward values are -1 when the task is incomplete and 0 when the task is completed. For `cube-double/triple/quadruple`, the RL agent needs to command an UR-5 arm to pick and place two/three/four cubes to target locations. In particular, `cube-triple` and `cube-quadruple` are extremely difficult to solve from offline data only, and often achieve zero success rate. The RL agent must explore efficiently online in these domains to solve the tasks. The `cube-*` domains provide a great test ground for sample-efficiency of offline-to-online RL algorithms which we primarily focus on. For `cube-quadruple`, we use the 100M-size dataset. The dataset is too big to fit our CPU memory, so we periodically (after every 1000 gradient steps) load in a 1M-size chunk of the dataset for offline training. For online training of **RLPD**, **QC-RLPD**, we use the same strategy where we load in a 1M-size chunk of the dataset as the offline data and perform 50/50 sampling (e.g., 50% of the data comes from the 1M-chunk of the offline data, 50% of the data comes from the online replay buffer). For online fine-tuning of **QC-***, **FQL**, **FQL-n**, **BFN**, and **BFN-n**, we keep a fixed 1M-size chunk of the offline dataset as the initialization of $\mathcal{D}$ and adds new data to $\mathcal{D}$ directly. The remaining 99M transitions in the offline data are not being used online. We now describe each of the five domains in details:

`scene-sparse`: This domain involves a drawer, a window, a cube and two button locks that control whether the drawer and the window can be opened. These tasks typically involve a sequence of actions. For example, `scene-task2` requires the robotic arm to unlock both locks, move the drawer and the window to the desired position, and then lock both locks. `scene-task4` requires the robotic arm to unlock the drawer, open the drawer, put the cube into the drawer, close the drawer. The reward is binary: -1 if the desired configuration is not yet reached and 0 if the desired configuration is reached (and the episode terminates).

`puzzle-3x3-sparse`: This domain contains a 3×3 grid of buttons. Each button has two states represented by its color (blue or red). Pressing any button causes its color and the color of all its adjacent buttons to flip (red → blue and blue → red). The goal is to achieve a pre-specified configuration of colors. `puzzle-3x3-task2` starts with all buttons to be blue, and the goal is to flip exactly one button (the top-left one) to be red. `puzzle-3x3-task4` starts with four buttons (top-center, bottom-center, left-center, right-center) to be blue, and the goal is to turn all the buttons to be blue. The reward is binary: -1 if the desired configuration is not yet reached and 0 if the desired configuration is reached (and the episode terminates).

`cube-double/triple/quadruple`: These three domains contain 2/3/4 cubes respectively. The tasks in the three domains all involve moving the cubes to their desired locations. The reward is $-n_{\text{wrong}}$ where n_{wrong} is the number of the cubes that are at the wrong position. The episode terminates when all cubes are at the correct position (reward is 0).

B.2 Robomimic environments.

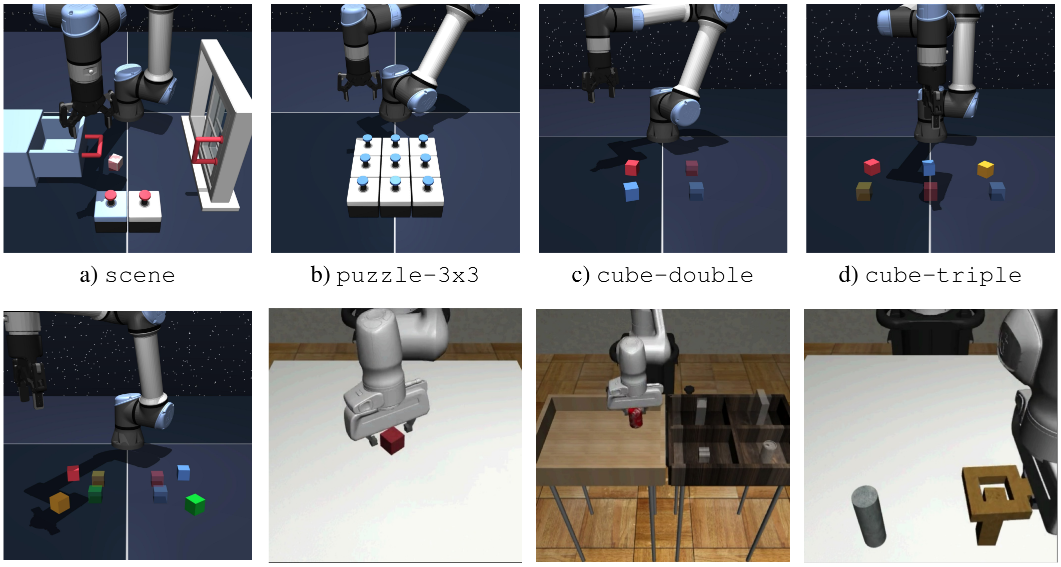
We use three challenging tasks from the robomimic domain [39]. We use the multi-human datasets that were collected by six human operators. Each dataset contains 300 successful trajectories. The three tasks are as described as follows.

- `lift`: This task requires the robot arm to pick a small cube. This is the simplest task of the benchmark.
- `can`: This task requires the robot arm to pick up a coke can and place in a smaller container bin.
- `square`: This task requires the robot arm to pick a square nut and place it on a rod. The nut is slightly bigger than the rod and requires the arm to move precisely to complete the task successfully.

All of the three robomimic tasks use binary task completion rewards where the agent receives -1 reward when the task is not completed and 0 reward when the task is completed.

Tasks	Dataset Size	Episode Length	Action Dimension (A)
scene-sparse-⋆	1M	750	5
puzzle-3x3-sparse-⋆	1M	500	5
cube-double-⋆	1M	500	5
cube-triple-⋆	3M	1000	5
cube-quadruple-100M-⋆	100M	1000	5
lift	31 127	500	7
can	62 756	500	7
square	80 731	500	7

Table 1: **Domain metadata.** Dataset size (number of transitions), episode length, and the action dimension. For OGBench tasks, the action dimension is 5 (x position, y position, z position, gripper yaw and gripper opening). For robomimic tasks, the action dimension is 7 for square to control one arm (3 degree of freedoms (DoF) for translation, 3 DoF for rotation, and one final DoF for the gripper opening).



a) scene b) puzzle-3x3 c) cube-double d) cube-triple

e) cube-quadruple f) lift g) can h) square

Figure 8: **We experiment on several challenging long-horizon, sparse-reward domains.** See detailed task description for each domain in Appendix B. The rendered images of the robomimic tasks above are taken from Mandlekar et al. [39].

C A variant of QC – QC-FQL

For this variant of our method, we are going to leverage the optimal transport framework to impose a Wasserstein distance constraint, again through the learned behavior policy $f_\xi(\cdot|s)$:

$$W_2(\pi_\psi, f_\xi(\cdot|s)) \leq \varepsilon \tag{12}$$

To impose this constraint, we parameterize our policy π_ψ with a noise-conditioned action prediction model, $\mu_\psi(s, \boldsymbol{z}) : \mathcal{S} \times \mathbb{R}^{Ah} \mapsto \mathbb{R}^{Ah}$, directly outputs an action from Gaussian noise in one network forward pass. While maximizing the Q-value, this noise-conditioned policy is regularized to be close to the behavioral cloning flow-matching policy via a distillation loss that is shown to be an upper-bound on the square 2-Wasserstein distance [53]:

$$L(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}, \boldsymbol{z}^0 \sim \mathcal{N}(0, \boldsymbol{I}_{Ah})} \left[\alpha \left\| \boldsymbol{z}^1 - \mu_\psi(s_t, \boldsymbol{z}^0) \right\|_2^2 - Q(s_t, \mu_\psi(s_t, \boldsymbol{z})) \right] \tag{13}$$

$$\geq \mathbb{E}_{s_t \sim \mathcal{D}, \boldsymbol{z}^0 \sim \mathcal{N}(0, \boldsymbol{I}_{Ah})} \left[\alpha W_2(\pi_\psi(\cdot|s_t), f_\xi(\cdot|s_t))^2 - Q(s_t, \mu_\psi(s_t, \boldsymbol{z})) \right], \tag{14}$$

where $\boldsymbol{z}^1$ is the ODE solution from $u = 0$ to $u = 1$ following $\mathrm{d}\boldsymbol{z}^u = f_\xi(s_t, \boldsymbol{z}^u, u)\mathrm{d}u$ (the initial value $\boldsymbol{z}^0$ is sampled from the unit Gaussian). The real-valued hyperparameter α directly controls the magnitude of the distillation loss. Finally, the TD loss remains the same as the previous section with the only difference in how we parameterize the policy:

$$L(\theta) = \mathbb{E}_{s_t, \boldsymbol{a}_t, s_{t+h} \sim \mathcal{D}, \boldsymbol{z}} \left[\left(Q_\theta(s_t, \boldsymbol{a}_t) - \sum_{t'=1}^{h} \gamma^{t'} r_{t+t'} - \gamma^h Q_{\bar{\theta}}(s_{t+h}, \mu_\psi(s_{t+h}, \boldsymbol{z})) \right)^2 \right] \tag{15}$$

where again $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I}_{Ah})$.

D Implementation Details

In this section, we provide more implementation details on both of our Q-chunking methods and all our baselines used in our experiments.

D.1 QC-FQL

We build the implementation of our method on top of **FQL** [53], a recently proposed offline RL/offline-to-online RL method that uses TD3+BC-style objective. It is implemented with a one-step noise-conditioned policy (instead of a Gaussian policy that is commonly used in RL) and it uses a distillation loss from a behavior flow-matching policy as the BC loss. To adapt this method to use action chunking, we simply apply **FQL** on the temporally extended action space – the behavior flow-matching policy generates a sequence of actions, the one-step noise-conditioned policy predicts a sequence of actions, and the Q-network also takes in a state and a sequence of actions. More concretely, we train three networks:

1. $Q_\theta(s, a_1, \cdots a_h) : \mathcal{S} \times \mathcal{A}^h \mapsto \mathbb{R}$ — the value function that takes in a state and a sequence of actions (action chunk). In practice, we train an ensemble of Q networks. We denote the weight of the ensemble element as $\theta = (\theta_1, \cdots, \theta_K)$.
2. $\mu_\psi(s, z) : \mathcal{S} \times \mathbb{R}^{Ah} \mapsto \mathbb{R}^{Ah}$ — the one-step noise-conditioned policy that takes in a state and a noise, and outputs a sequence of actions conditioned on them.
3. $f_\xi(s, m, u) : \mathcal{S} \times \mathbb{R}^{Ah} \times [0, 1] \mapsto \mathbb{R}^{Ah}$ — the flow-matching behavior policy parameterized by a velocity prediction network. The network predicts takes in a state, an intermediate state of the flow and a time, and outputs the velocity direction that the intermediate action sequence should move in at the specified time. See Algorithm 1 for more details on how this velocity prediction network is used to generate an action from a noise vector.

We denote our policy as $\pi_\psi(\cdot|s)$. It is implemented by first sampling a Gaussian noise $z \sim \mathcal{N}(0, I_{Ah})$ and run it through the one-step noise-conditioned policy $[a_1 \quad \cdots \quad a_h] \leftarrow \mu_\psi(s, z)$. To train these three networks, we sample a high-level transition, $w = (s_t, a_t, a_{t+1}, \cdots, a_{t+h-1}, s_{t+h}, r_t^h) \sim \mathcal{D}$ where $r_t^h = \sum_{t'=0}^{h-1} \gamma^{t'} r_{t+t'}$, to construct the following losses:

(1) Critic loss:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t, \cdots, a_{t+h-1}) - r_t^h - \frac{1}{K} \sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+h}, a_{t+h}, \cdots, a_{t+2h-1}) \right)^2, \tag{16}$$

where $[a_{t+h} \quad \cdots \quad a_{t+2h-1}] \sim \pi_\psi(\cdot|s_{t+h}), k \in \{1, 2, \cdots, K\}$.

(2) Actor loss:

$$L(\psi, w) = -Q_\theta(s_t, \mu_\psi(s_t, z_t)) + \alpha \left\| \mu_\psi(s_t, z_t) - \left[a_t^\xi \quad \cdots \quad a_{t+h-1}^\xi \right] \right\|_2^2, \tag{17}$$

where $z_t \sim \mathcal{N}(0, I_{Ah})$ and $\left[a_t^\xi \quad \cdots \quad a_{t+h-1}^\xi \right]$ is the result of running the behavior policy $f_\xi(s, m, t)$ with Algorithm 1 from z_t, and $\alpha \in \mathbb{R}$ is a tunable parameter that controls the strength of the behavior regularization (higher α leads to stronger behavior regularization).

(3) Flow-matching behavior policy loss:

$$L(\xi, w) = \| f_\xi(s_t, t\,[a_t \quad \cdots \quad a_{t+h-1}] + (1-u) z_t, u) - ([a_t \quad \cdots \quad a_{t+h-1}] - z_t) \|_2^2, \tag{18}$$

where $u \sim U([0,1])$, $z_t \sim \mathcal{N}(0, I_{Ah})$.

Practically, we sample a batch of transitions $\{w_1, w_2, \cdots, w_M\}$ and optimize the average loss for each network: $\mathcal{L}(\theta) = \frac{1}{M}\sum_{i=1}^{M}\sum_{k=1}^{K}\mathcal{L}(\theta_k, w_i), \mathcal{L}(\psi) = \frac{1}{N}\sum_{i=1}^{M}\mathcal{L}(\psi, w_i), \mathcal{L}(\xi) = \frac{1}{N}\sum_{i=1}^{M}\mathcal{L}(\xi, w_i)$.

D.2 FQL

FQL [53] is a recently proposed offline RL/offline-to-online RL method that uses TD3+BC-style objective. It is equivalent to our method with $h = 1$. For completeness, we write out the objectives for a transition sample $w = (s_t, a_t, r_t, s_{t+1})$:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t) - r_t - \frac{1}{K}\sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+1}, \mu_\psi(s_{t+1}, z_t^{k'}))\right)^2, z_t^{k'} \sim \mathcal{N}(0, I_A), \tag{19}$$

$$L(\psi, w) = -Q_\theta(s_t, \mu_\psi(s_t, z_t)) + \alpha \left\|\mu_\psi(s_t, z_t) - a_t^\xi\right\|_2^2, \tag{20}$$

$$a_t^\xi \leftarrow \texttt{FlowODE\_Euler}(s_t, z_t, f_\xi, T), z_t \sim \mathcal{N}(0, I_A), \tag{21}$$

$$L(\xi, w) = \|f_\xi(s_t, ua_t + (1-u)z_t, u) - (a_t - z_t)\|_2^2, z_t \sim \mathcal{N}(0, I_A), u \sim U([0,1]). \tag{22}$$

In practice, we sample a batch of transitions $\{w_1, w_2, \cdots, w_N\}$ and optimize the average loss for each network: $\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\mathcal{L}(\theta_k, w_i), \mathcal{L}(\psi) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\psi, w_i), \mathcal{L}(\xi) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\xi, w_i)$.

D.3 FQL-n

To implement the n-step return baseline, we take FQL and replace the 1-step TD update with the h-step TD update:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t) - \sum_{t'=0}^{h-1}(\gamma^{t'} r_{t+t'}) - \frac{1}{K}\sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+h}, \mu_\psi(s_{t+h}, z_t^{k'}))\right)^2, \tag{23}$$

where $z_t^{k'} \sim \mathcal{N}(0, I_A)$ for all $k' \in \{1, 2, \cdots, K\}$. The actor loss and flow-matching loss remain the same as FQL.

D.4 QC

The flow-matching behavior policy is trained with the same loss as used in QC-FQL (Equation (18)). On top of the flow-matching behavior policy, we simply parameterize the policy π implicitly by sampling multiple action chunks from the behavior policy and pick the one that maximizes the Q-value. Specifically, let $\{z_t^1, z_t^2, \cdots, z_t^N\} \sim \mathcal{N}(0, I_{Ah})$ and

$$\begin{bmatrix} a_t^i & \cdots & a_{t+h-1}^i \end{bmatrix} = \texttt{FlowODE\_Euler}(s_t, z_t^i, f_\xi, T)$$

The policy outputs the one action chunk out of N that maximizes the Q-value, $\begin{bmatrix} a_t^{i^\star} & \cdots & a_{t+h-1}^{i^\star} \end{bmatrix}$, where

$$i^\star \leftarrow \arg\max_{i \in [N]} Q(s, \begin{bmatrix} a_t^i & \cdots & a_{t+h-1}^i \end{bmatrix}).$$

Finally, we directly use this implicitly parameterize policy to generate actions for computing the TD target for our TD loss:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t, \cdots, a_{t+h-1}) - r_t^h - \frac{1}{K}\sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+h}, a_{t+h}^{i^\star}, \cdots, a_{t+2h-1}^{i^\star})\right)^2 \tag{24}$$

where $a_{t+h}^{i^\star}, \cdots, a_{t+2h-1}^{i^\star} \sim \pi(\cdot|s_{t+h})$.

The baselines BFN-n and BFN are implemented similarly to FQL-n and FQL by operating in the original action space.

Algorithm 1 `FlowODE_Euler`(s_t, z_t, f_ξ, T): generate actions from the behavior flow policy $f_\xi(s, m, u)$ with Euler's method.

```
**Input:** State $s_t$, noise $z_t$ and flow model $f_\xi(s, m, u)$, number of flow steps $T$.
$m^0 \leftarrow z_t$
**for** $i \in \{1, \cdots, T\}$ **do**
    $m^i \leftarrow f_\xi(s_t, m^{i-1}, (i-1)/T)$
**Output:** $m^T$.
```

D.5 RLPD, RLPD-AC, QC-RLPD

All the **RLPD** baseline results are obtained by running the official codebase (as linked in Ball et al. [7]) with additional modification to incorporate action chunking and behavior cloning. This baseline runs online RL from scratch using off-policy transitions where 50% of them come from the offline dataset and the other 50% come from the online replay buffer. It essentially up-weights the online data more, allowing the online RL agent to learn more quickly. This is different from how **QC-***, **BFN**, **BFN-n**, **FQL**, **FQL-n** samples off-policy transitions (where we sample from the dataset that combines the offline dataset and online replay buffer data with no weighting). **RLPD** baselines all use Gaussian policy. This is also different from our method as our method uses noise-conditioned policy that can represent a wider range of distributions. For **RLPD-AC**, we change all the actor and critic networks such that they work with an action chunk rather than a single action. The baseline is exactly the same as our method except that actor and the critic are updated the same as how **RLPD** updates its actor and critic. For **QC-RLPD**, we add a behavior cloning loss in the actor loss as follows (highlighted in red below):

$$\mathcal{L}(\psi) = \mathbb{E}_{a'_t \sim \pi_\psi(\cdot|s_t)} \left[-\frac{1}{K} \sum_{k=1}^{K} Q_{\theta_k}(s_t, a'_t) - \alpha \log \pi_\psi(a_t|s_t) \right]. \tag{25}$$

E Experiment Details

E.1 Evaluation protocol

For our method and all baselines, we run 5 seeds on each task. All plots use 95% confidence interval with stratified sampling (5000 samples). The success rate is computed by running the policy in the environment for 50 episodes and record the number of times that the policy succeeds at solving the task (and divide it by 50).

E.2 Hyperparameter tuning

Parameter	Value
Batch size (M)	256
Discount factor (γ)	0.99
Optimizer	Adam
Learning rate	3×10^{-4}
Target network update rate (τ)	5×10^{-3}
Critic ensemble size (K)	10 for **RLPD**, **RLPD-AC**, **QC-RLPD** 2 for **QC-FQL**, **FQL**, **FQL-n**, **QC-BFN**, **BFN**, **BFN-n**
UTD Ratio	1
Number of flow steps (T)	10
Number of offline training steps	10^6 except **RLPD**-based approaches (0)
Number of online environment steps	1×10^6
Network width	512
Network depth	4 hidden layers

Table 2: **Common hyperparameters.**

Environments	FQL	FQL-n	QC-FQL
scene-sparse-⋆	300	100	300
puzzle-3x3-sparse-⋆	100	100	300
cube-double-⋆	300	100	300
cube-triple-⋆	300	100	100
cube-quadruple-100M-⋆	300	100	100
lift	10000	10000	10000
can	10000	10000	10000
square	10000	10000	10000

Table 3: **Behavior regularization coefficient (α).**

Environments	BFN	BFN-n	QC-BFN
scene-sparse⋆	4	4	32
puzzle-3x3-sparse-⋆	4	4	64
cube-double-⋆	4	4	32
cube-triple-⋆	4	4	32
cube-quadruple-100M-⋆	4	4	32
lift	4	4	16
can	4	4	16
square	4	4	16

Table 4: **Number of actions sampled for the expected-max Q operator (N) for BFN methods.**

QC, **BFN**, **BFN-n**. We tune the number of actions sampled, N, for the expcted-max Q operator. On OGBench domains, we sweep over $\{2, 4, 8, 16, 32, 64, 128\}$ and select the best parameter for each domain and for each method on `task2`. We report the performance of each method with the best α in Figure 3 and Figure 1 (on all tasks). Table 4 summarizes the α value we use for each task.

QC-FQL, **FQL**, **FQL-n**. We tune the behavior regularization coefficient α. On OGBench domains, we take the default hyperparameter of **FQL** for each domain α_{default} and tune all methods on `task2` of each domain with three choices of α: $\{\alpha_{\text{default}}/3, \alpha_{\text{default}}, 3\alpha_{\text{default}}\}$ (our α_{default} comes from Table 6 in Park et al. [53]). On Robomimic domain, we sweep over much large α values: $\{100, 1000, 10000\}$. We report the performance of each method with the best α in Figure 3 and Figure 1 (on all tasks). Table 3 summarizes the α value we use for each task.

RLPD, **RLPD-AC**, **QC-RLPD**. We sweep over (1) whether or not to use clipped double Q-learning (CDQ), and (2) whether or not to use entropy backup. We find that not using CDQ and not using entropy backup to perform the best for all of the RLPD baselines and use that across all domains. Even though our method and the other FQL baselines use $K = 2$ critic ensemble size, we use $K = 10$ critic ensemble size for **RLPD** to keep it the same as the hyperparameter in the original paper [7]. For **QC-RLPD**, we sweep over behavior regularization coefficient $\alpha \in \{0.001, 0.01, 0.1\}$ and pick 0.01 since it works the best.

F Full Results

F.1 End-effector visualization

We provide more examples of the trajectory rollouts from **QC** and **BFN** over the course of online training on `cube-triple-task3`. In Figure 9, we show the first 9000 time steps (broken down into 9 subplots where each visualizes 1000 time steps). In Figure 10, we show another 9000 time steps but late in the training (from environment step 9×10^5). The first example is the same as the one used in Figure 5.

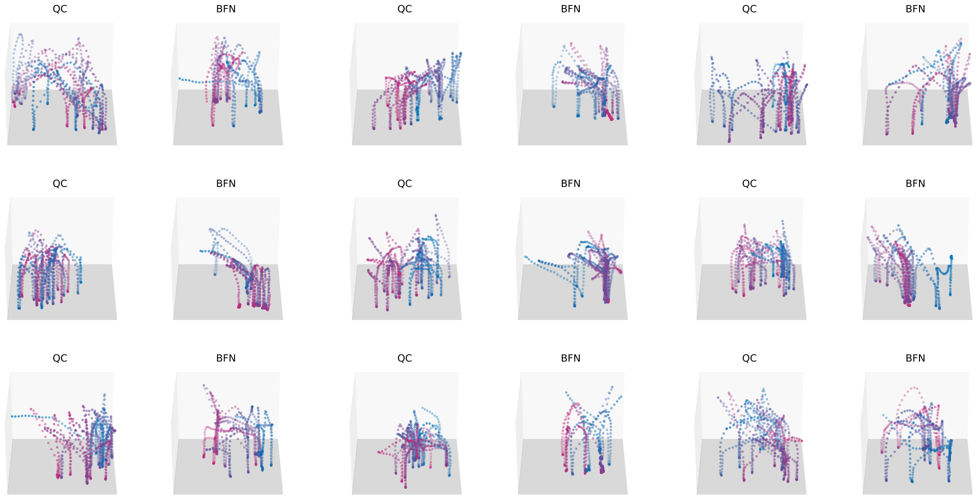


Figure 9: **End-effector trajectory early in the training.** Each subplot above shows the trajectory for a consecutive of 1000 time steps. We include up to Step 9000.

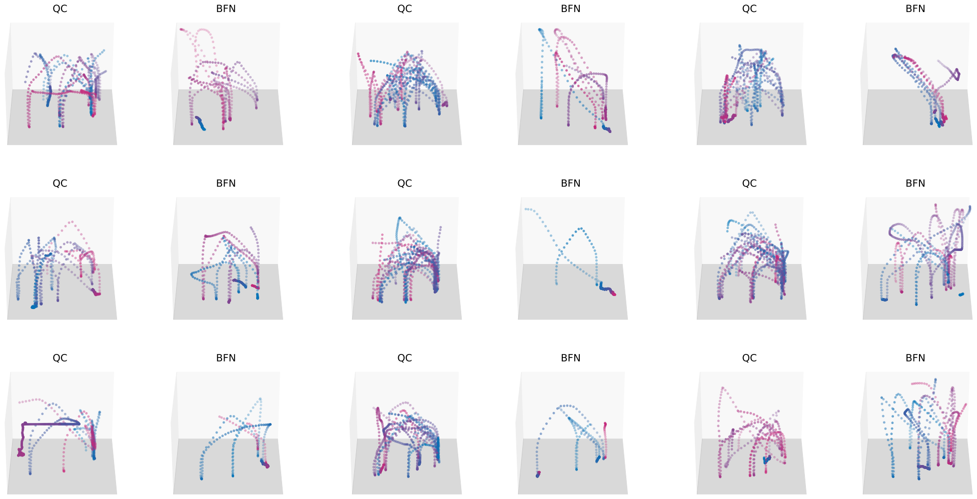


Figure 10: **End-effector trajectory visualization late in the training.** Each subplot above shows the trajectory for a consecutive of 1000 time steps. We include the trajectories from Step 900000 to Step 99000.

F.2 OGBench results by individual task

Main results by task. The following plot (Figure 12 shows the performance breakdown for Figure 3.

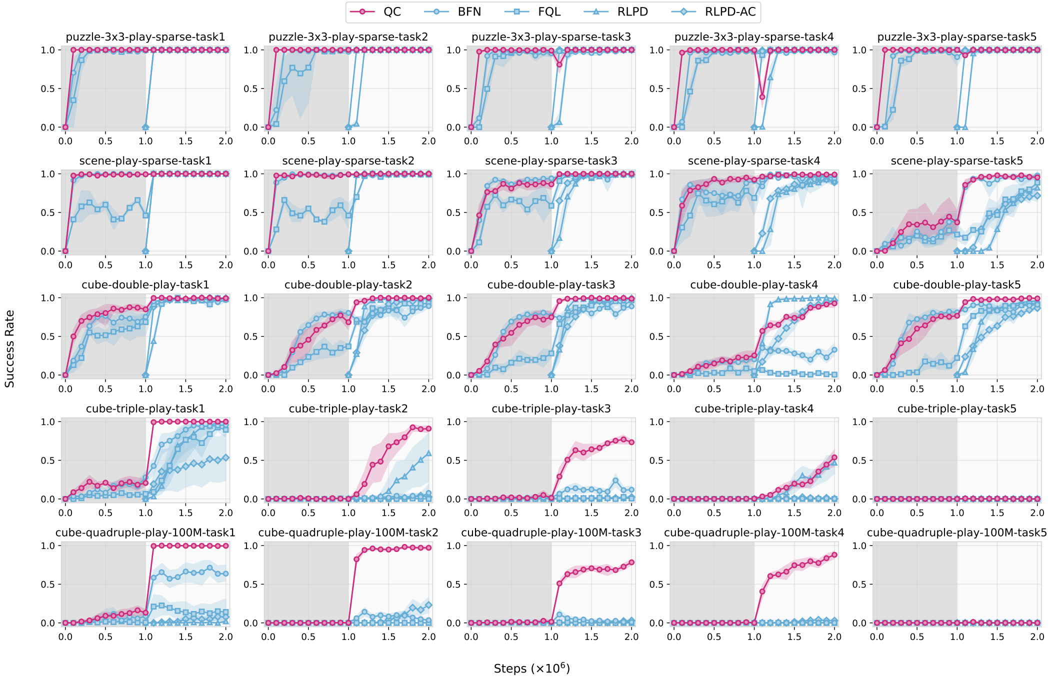


Figure 11: **Full OGBench results by task.** For each method on each task, we use 5 seeds.

Ablation results by task. The following plot (Figure 12) shows the performance breakdown for Figure 4.

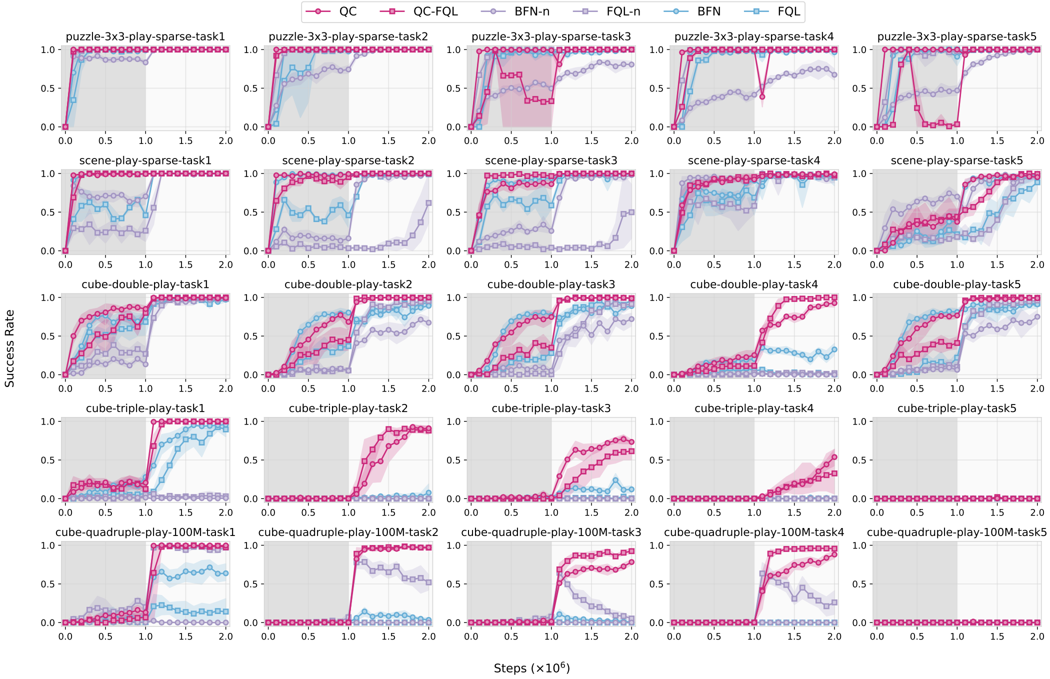


Figure 12: **Full OGBench results by task.** For each method on each task, we use 5 seeds.

Q-chunking with Gausian policies. The following plot shows the performance breakdown for Figure 2. In addition, we include a new method for comparison, **QC-RLPD**, where we add a behavior cloning loss to **RLPD-AC** (**RLPD** with action chunking).

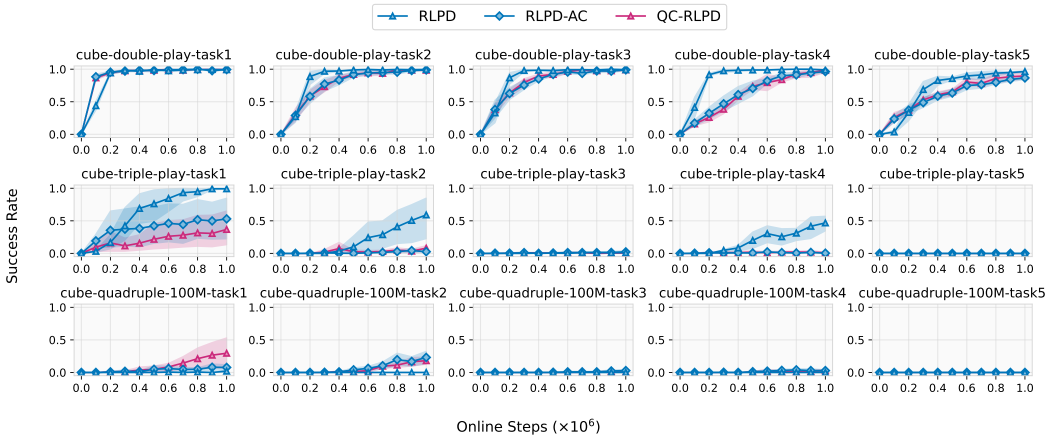


Figure 13: **Full RLPD results by task.** For each method on each task, we use 5 seeds. **QC-RLPD** is **RLPD-AC** (**RLPD** on the temporally extended action space) where we additionally add a fixed behavior cloning coefficient of 0.01.

F.3 Robomimic results

Figure 14 shows the performance of **QC**, **QC-FQL**, **BFN-n**, **FQL-n**, **BFN**, **FQL** our three robomimic tasks. This plot shows the performance breakdown for Figure 4 (right).

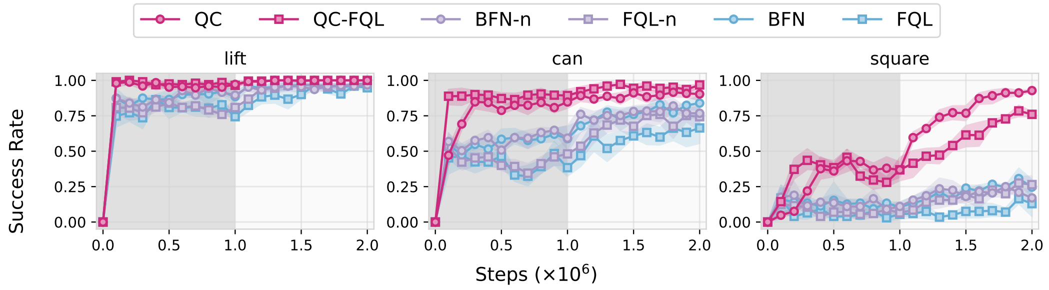


Figure 14: **Full robomimic ablation by task.** For each method on each task, we use 5 seeds.

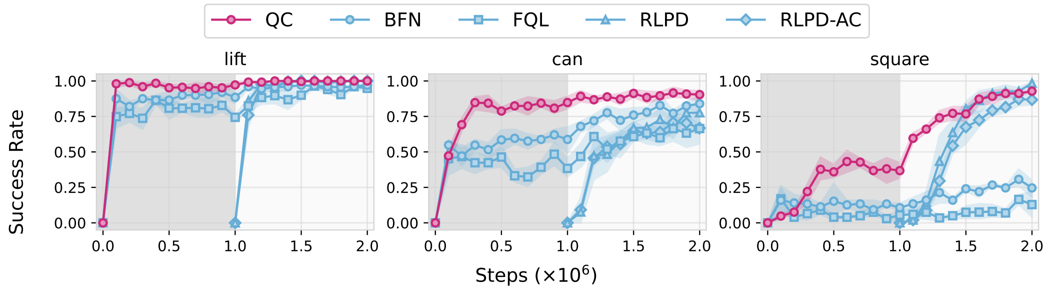


Figure 15: **Robomimic results.** **QC** achieves strong performance across all three robomimic tasks. The first 1M steps are offline and the next 1M steps are online with one environment step per training step (5 seeds).

F.4 Additional Related Work

Exploration with temporally coherent actions. Existing methods either rely on temporally correlated action noises [36] that are constructed through heuristics; hierarchically structured policies (see the next paragraph), which are often tricky to stabilize during online training; or pre-trained frozen skill policies [55, 77], which are not amendable for fine-grained online fine-tuning. Our method uses a single network to represent the policy to generate temporally extended action chunk and it is trained using a single objective function that is stable to optimize. There is also no frozen, pretrained components in our approach, ensuring its online fine-tuning flexibility.

Hierarchical reinforcement learning, options framework. Learning temporally extended actions have also been widely studied in the hierarchical reinforcement learning (HRL) literature [14, 15, 73, 13, 32, 74, 54, 57, 44, 3, 60, 55, 20, 79]. HRL methods typically train a space of low-level policies that can directly interact with the environment along with a high-level policy that selects among these low-level policies. These low-level policies can be hand-crafted [12], automatically discovered online [15, 32, 73, 74, 44], or pretrained using offline skill discovery methods [49, 42, 60, 3, 63, 55, 72, 47, 26, 18, 9, 52]. The options framework provides a slightly more sophisticated and more powerful formulation, where the low-level policy is additionally associated with learnable initiation condition and termination condition that makes utilization of the low-level policy more flexible [69, 41, 10, 40, 61, 62, 29, 13, 65, 48, 17, 4, 28, 5, 6]. A long-lasting challenge in HRL is its bi-level optimization problem: when both low-level and high-level policies are updated during training, the high-level policies must optimize a moving objective function, which can lead to instability [44]. To mitigate this, some methods keep the low-level policies frozen after initial pretraining [3, 55, 77] to improve stability during online training. Our approach is a special case of HRL where the low-level skill executes a sequence of actions open-loop. This design choice allows us to collapse the bi-level optimization problem into a standard RL objective in a temporally extended action space, while retaining many of the exploration benefits associated with HRL methods.

G Future Research

We hope our work will serve as a first step towards training non-Markovian policy for effective online exploration from prior offline data. Several challenges remain, opening promising directions for future research. First, our approach use a fixed action chunk, but it is unclear how to choose this size other than task-specific hyperparameter tuning. A natural next step would be to develop mechanisms that automatically determine chunk boundaries, akin to initiation and termination conditions in the options framework [67]). Second, action chunking represents only a limited subclass of non-Markovian policies and may perform poorly in settings where a high-frequency control feedback loop is essential (e.g., on-board controller of a quadcopter). We believe that developing practical techniques for training a more general class of non-Markovian policy for online exploration would further improve the online sample efficiency of offline-to-online RL algorithms.