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.

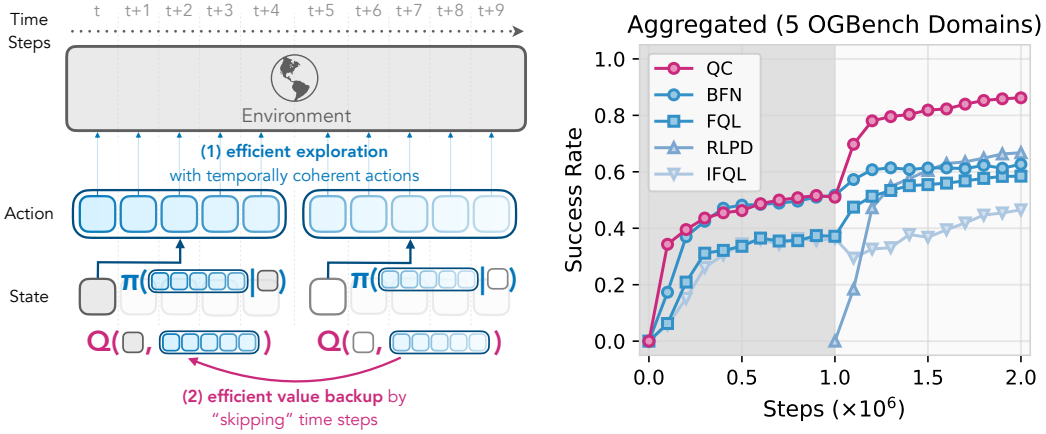


Figure 1: **Q-chunking uses action chunking to enable fast value backups and effective exploration with temporally coherent actions.** *left:* an overview of our approach: Q-chunking operates in a temporally extended action space that allows for (1) efficient value backups and (2) effective exploration 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`

.

39th Conference on Neural Information Processing Systems (NeurIPS 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 [28, 39, 85]. 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") [90, 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 [90]. Action chunking has not been used widely in RL, perhaps because optimal policies in fully observed MDPs are Markovian [74], 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 pre-training 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 recipe for improving generic 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, essentially running RL with behavior regularization in the chunked action space. 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 (Algorithm 1). QC achieves strong performance on a range of six challenging long-horizon, sparse-reward domains, outperforming prior offline-to-online methods. Moreover, Q-chunking is a generic recipe that can be applied to existing offline-to-online algorithms with minimal modification. In this work, we demonstrate one such instantiation by applying it to **FQL** [58], resulting in **QC-FQL** (Algorithm 2), 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 [88, 71, 37, 1, 89, 91, 7, 51, 92, 39]. 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 [50, 36, 33, 77, 58, 2, 42, 37]. 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 [92, 51, 42, 37, 82]. 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") [90]. Action chunking has been shown to improve policy robustness [90, 23, 8], and handle non-Markovian behavior in offline data [90]. Existing RL methods that incorporate action chunking typically focus on fine-tuning a policy pre-trained with imitation learning [61]. Tian et al. [78] 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. Li et al. [38] also observe that learning a critic over short action chunks removes the off-policy bias in n-step return backups, leading to more stable and effective value learning. There are key differences between their work and ours — Li et al. [38] operate in the online episodic RL setting [84, 31] and use Gaussian policies to predict parameters of motion primitives (MP) [63, 54], which are then used to generate full action sequences at the beginning of each episode. In contrast, we operate in the conventional offline-to-online RL setting and leverage more expressive flow-matching-based policies (as we find Gaussian policies to be ineffective as shown in Figure 2) to predict short action sequences directly in the raw action space. Seo and Abbeel [65] also train critics on action chunks and impose a behavior cloning loss that aligns with the principles behind Q-chunking. The key distinction from our work lies in their use of a multi-level, factorized critic architecture [66], which generates and refines action chunks from coarse to fine granularity via iterative discretization. At each level, the action space is discretized into bins, and the Q-function is modeled independently for each action dimension and timestep, conditioned on the whole action chunks predicted at the previous coarser level. While this factorized critic design enables tractable value-maximizing action sampling, it imposes strong structural assumptions on the action space, limiting policy expressiveness at each refinement level. In contrast, we make no such assumptions in our recipe which allows us to derive two general algorithms where both the critic and policy operate directly on action chunks without requiring factorization or iterative discretization/refinement.

Exploration with temporally coherent actions. Existing methods either rely on temporally correlated action noises [40] 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 [60, 85], 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, 16, 80, 13, 35, 81, 59, 62, 49, 3, 67, 60, 22, 87]. 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 [16, 35, 80, 81, 49], or pretrained using offline skill discovery methods [54, 47, 67, 3, 70, 60, 79, 52, 28, 20, 9, 57]. 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 [75, 46, 10, 44, 68, 69, 32, 13, 72, 53, 19, 4, 30, 5, 6, 15]. 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 [49]. To mitigate this, some methods keep the low-level policies frozen after initial pretraining [3, 60, 85] 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. While

prior work in HRL has also explored such idea [45, 17], they often leverage a discrete set of action sequences that is either heuristically extracted from the prior experience or given in advance. In contrast, we use an expressive flow-matching based policy to directly parameterize a continuous space of action sequences, and are directly trained and fine-tuned online with RL.

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 [53, 64]. 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.

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 [74]:

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

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

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

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 [83, 74]:

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

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* [18, 34] 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 [48, 26, 29, 86].

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 $\boldsymbol{a}_{t:t+h}$ followed by the state, and the state h steps into the future, s_{t+h}. Specifically, we train Q_θ with the following TD loss,

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

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 [25].

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:

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

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

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

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 [18]. 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. We formalize this argument in Theorem A.1. As a result, Q-chunking value backup enjoys the value propagation speedup while maintaining an unbiased value estimate.

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 particularly advantageous in

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

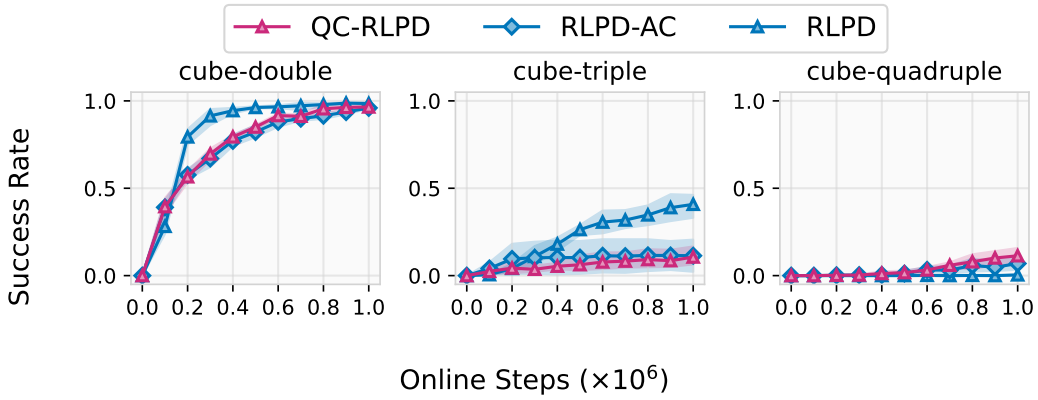


Figure 2: **Naïvely using action chunking for online RL with Gaussian policies leads to poor performance.** *(1)* **RLPD** runs online RL on both offline data and online replay buffer [7]. *(2)* **RLPD-AC** is the same algorithm as RLPD but operates in a temporally extended action space (action chunk size of 5). *(3)* **QC-RLPD** additionally uses a behavior cloning loss on the actor (4 seeds).

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 [56], human tele-operators [43], or noisy expert policies for sub-tasks [56, 21]) 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). We observe that Q-chunking, with such behavior constraints, can explore with temporally coherent actions (see Section 5.3), mitigating the exploration challenge.

4.3 Practical implementations

A 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 prerequisites of imposing a good behavior constraint is the ability of the policy to capture the complex behavior distribution (*e.g.*, with a flow/diffusion policy). A Gaussian policy, a default choice in online RL algorithms, does not suffice. Indeed, if we naïvely take an off-the-shelf online algorithm, **RLPD** [7] for example, and apply Q-chunking with a behavior cloning loss, we find that it often performs poorly (Figure 2).

To enforce a good behavior constraint, we start by using flow-matching objective [41] 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, which serves as an approximation of the true behavior distribution in the offline data ($f_\xi \approx \pi_\beta$). Now, we are ready to present our main method:

QC: Q-chunking with implicit KL behavior constraint. We consider a KL constraint on our policy through the learned behavior distribution:

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

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 [73] to maximize Q-value while imposing this KL constraint implicitly altogether. Practically, this involves first sampling N action chunks from the learned behavior policy $f_\xi(\cdot|s)$,

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

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 [27]:

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

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 and only sample from the behavioral policy $f_\xi(\cdot|s_t)$. 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. [24]. 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 QC is simple and easy to implement, it does come with some additional computational costs associated with best-of-N sampling. In our experiments, we experiment with two other variants (**QC-FQL** and **QC-IFQL**) that leverage cheaper off-the-shelf offline/offline-to-online RL methods (FQL and IFQL respectively [58]). We present the FQL-version below as it performs better empirically.

QC-FQL: Q-chunking with 2-Wasserstein distance behavior constraint. For this variant of our method, we leverage the optimal transport framework to impose a Wasserstein distance (W_2) constraint, again, through the learned behavior policy $f_\xi(\cdot|s)$:

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

Following **FQL** [58], we parameterize the policy π_ψ with a noise-conditioned action prediction model, $\mu_\psi(s, \boldsymbol{z}) : \mathcal{S} \times \mathbb{R}^{Ah} \mapsto \mathbb{R}^{Ah}$, which directly outputs an action from Gaussian noise in one network forward pass. This noise-conditioned policy is trained to maximize the Q-chunking critic $Q_\theta(s_t, \boldsymbol{a}_{t:t+h})$ while being regularized to be close to the behavioral cloning flow-matching policy via a BC loss that is shown to be an upper-bound on the squared 2-Wasserstein distance [58]:

$$\begin{aligned} L(\psi) &= \mathbb{E}_{s_t \sim \mathcal{D}, \boldsymbol{z}^0 \sim \mathcal{N}(0, \boldsymbol{I}_{Ah})} \left[\alpha \left\| \boldsymbol{z}^1 - \mu_\psi(s_t, \boldsymbol{z}^0) \right\|_2^2 - Q(s_t, \mu_\psi(s_t, \boldsymbol{z})) \right] && (13) \\ &\geq \mathbb{E}_{s_t \sim \mathcal{D}, \boldsymbol{z}^0 \sim \mathcal{N}(0, \boldsymbol{I}_{Ah})} \left[\alpha W_2(\pi_\psi(\cdot|s_t), f_\xi(\cdot|s_t))^2 - Q(s_t, \mu_\psi(s_t, \boldsymbol{z})) \right], && (14) \end{aligned}$$

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

Offline-to-online RL considerations. Since both variants of our methods use behavior constraint (implicit KL for QC, explicit W_2 for QC-FQL), we can also directly run them for offline RL pre-training. For both offline and online training, we use the same behavior constraint strength (*e.g.*, N for QC and α for QC-FQL). We use the same algorithm for both offline and online training, and the only difference is whether there is environment interactions. See Section C, Algorithm 1 and Algorithm 2 for an overview of QC and QC-FQL during online training.

5 Experimental Results

We conduct extensive 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:

		puzzle-3x3-sparse (5 tasks)	scene-sparse (5 tasks)	cube-double (5 tasks)	cube-triple (5 tasks)	cube-quadruple (5 tasks)	overall (25 tasks)
From Scratch (1-step TD)	RLPD	- → **100**	- → 94	- → **98**	- → 41	- → 0	- → 67
	RLPD-AC	- → **100**	- → 91	- → 96	- → 11	- → 7	- → 61
	SUPE-GT	- → **100**	- → **92**	- → 66	- → 0	- → 0	- → 52
Gaussian (1-step TD)	IQL	0 → 20	0 → 39	0 → 0	0 → 0	0 → 0	0 → 12
	ReBRAC	55 → **100**	11 → **99**	3 → 30	0 → 0	1 → 20	14 → 50
Flow (1-step TD)	IFQL	**98** → 97	68 → 73	11 → 59	0 → 3	0 → 0	36 → 47
	FQL	**100** → **100**	57 → 95	28 → 76	1 → 18	0 → 3	37 → 58
	BFN	**98** → **100**	**85** → **99**	**68** → 79	**4** → 23	1 → 12	**51** → 63
Flow (n-step TD)	IFQL-n	94 → **100**	68 → 92	5 → 29	0 → 0	0 → 0	34 → 44
	FQL-n	**98** → **100**	18 → 70	11 → 77	0 → 1	**7** → 37	27 → 57
	BFN-n	58 → 90	57 → 97	11 → 65	0 → 0	0 → 0	25 → 50
Q-chunking (Ours)	QC-IFQL	**100** → **100**	**83** → 98	12 → 78	0 → 0	1 → 19	39 → 59
	QC-FQL	63 → **100**	**84** → **99**	39 → **100**	**4** → **53**	1 → **77**	38 → **86**
	QC	**100** → **100**	**84** → **99**	**67** → **98**	**6** → **64**	**4** → 74	**52** → **86**

Table 1: **Summary table for OGBench offline-to-online RL results.** For each cell, we report the offline performance after 1M of training steps and then the online performance after 1M of additional online steps. The best method(s) for each column is highlighted in bold and color. Q-chunking methods outperform all prior methods at the end of the online training. See the full results in Table 6 (complete table) and Figure 10 (individual plot).

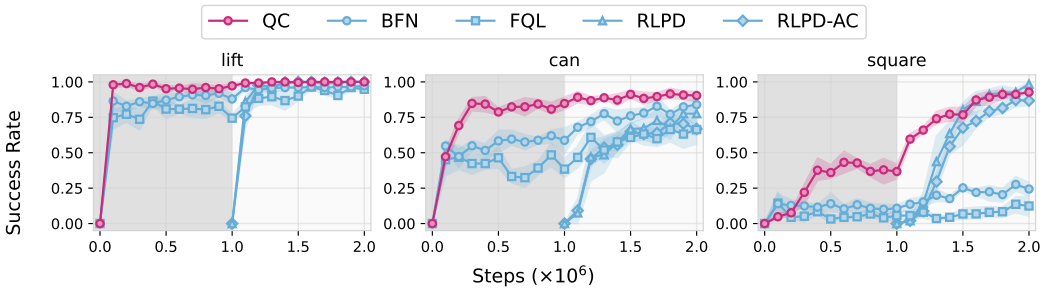


Figure 3: **Robomimic results.** QC achieves strong performance across all three robomimic tasks. The first 1M steps are offline and the next 1M steps are online with one environment step per training step (5 seeds).

(Q1) *How well do Q-chunking methods perform compared to prior offline-to-online RL methods?*

(Q2) *Why does action chunking helps online learning?*

(Q3) *How does chunk length, critic ensemble size, and update-to-data ratio affect performance?*

5.1 Environments and Datasets

We consider six sparse reward robotic manipulation domains with tasks of varying difficulties. This includes 5 domains (5 tasks each) from OGBench [55], `scene-sparse`, `puzzle-3x3-sparse`, `cube-double/triple/quadruple` and 3 tasks from robomimic [43]. For OGBench, we use the default play-style datasets except for `cube-quadruple` where we use the larger 100M-size dataset. For robomimic, we use the multi-human datasets. See more details in Appendix B.

5.2 Comparisons

We compare with prior methods that speedup value backup as well as the previous best offline-to-online RL methods. We include a brief description of them below with more details in Section C.

BFN (best-of-N) is a baseline that we propose to combine the expected-max Q operator [24] with an expressive behavior flow policy. BFN operates in the original action space and uses best-of-N

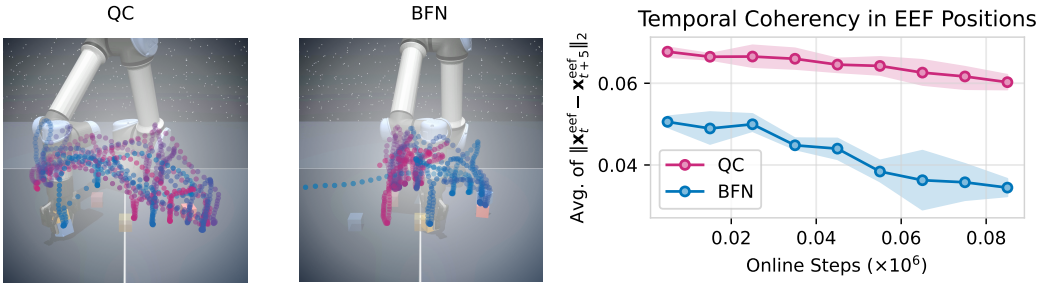


Figure 5: **End-effector movements early in the training and temporal coherency analysis on** `cube-triple-task3`**.** *Left:* **QC** covers a more diverse set of states compared to **BFN** in the first 1000 environment steps. *Right:* **QC** exhibits a higher temporal coherency in end-effector compared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

sampling to pick the action (out of N) that maximizes the current Q-value. This baseline is an ablation to isolate the benefit of Q-chunking in **QC**.

FQL [58] is a recently proposed offline RL method that achieves strong offline and offline-to-online RL performance. This baseline is an ablation to isolate the benefit of Q-chunking in **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.

We also compare with **SUPE-GT** [85], a recently proposed skill-based that is designed for offline-to-online RL. The original method is designed to deal with unlabeled dataset by learning a reward model with reward bonuses. We adapt it to our setting by directly using the ground truth rewards.

Finally, we compare with additional baselines: **IQL** [33], **ReBRAC** [76], **IFQL** (a baseline implemented in Park et al. [58] that combines IQL with rejection sampling policy extraction), and **IFQL-n** (IFQL with n-step return backup).

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

We report the performance of all Q-chunking methods and the baselines in Table 1 (OGBench) and a selective (strong) subset in Figure 3 (robomimic). **QC** achieves competitive performance offline, often matching or sometimes outperforming best prior methods. In the online phase, **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. In addition, on OGBench domains, all Q-chunking methods (QC, QC-FQL, QC-IFQL) outperforms both their corresponding 1-step TD counterpart (BFN, FQL, IFQL) and their correspdoning n-step return counterpart (BFN-n, FQL-n, IFQL-n). A similar trend continues on robomimic tasks as shown in Figure 4. Across OGBench and robomimic, n-step return baselines, which do not use chunked critics or policies, perform significantly worse than Q-chunking methods.

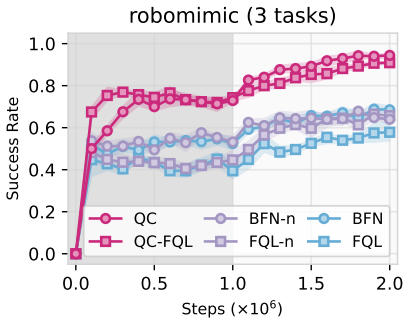


Figure 4: **n-step return ablations on robomimic.** Both Q-chunking methods consistently outperform their n-step and 1-step TD counterparts (5 seeds).

5.4 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

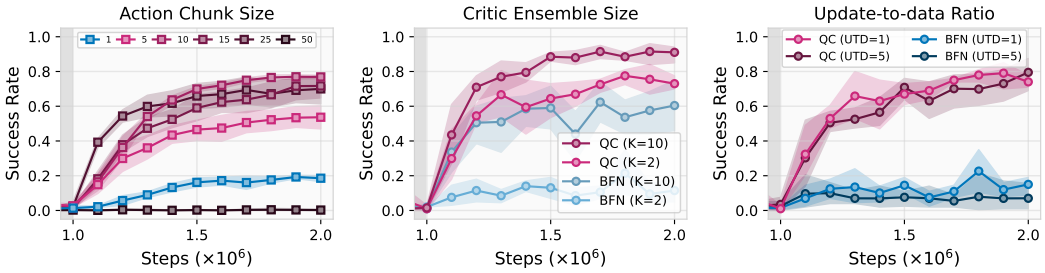


Figure 6: **Sensitivity analysis: action chunk size (h), critic ensemble size (K), and update-to-data ratio (UTD).** *Left:* **QC-FQL** with different h on all 5 `cube-triple` tasks (5 seeds). QC-FQL with $h = 1$ is equivalent to **FQL**. *Center:* Increasing the ensemble size to $K = 10$ improves performance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

holds empirically. We first visualize the end-effector movements early in the training for **QC** and **BFN** (Figure 5, left). **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 fewer pauses (fewer and shallower clusters) and a more diverse state coverage in the end-effector space. We include additional examples in Appendix E, Figure 8 and Figure 9. To get a quantitative measure of the temporal coherency in actions, we record the 3-D end-effector position 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. 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.

5.5 How does action chunk length, critic ensemble size, and UTD ratio affect performance?

In Figure 6 (left), we examine the performance of **QC-FQL** across different action chunk lengths ($h \in \{1, 5, 10, 25, 50\}$) on the `cube-triple` domain. Increasing the chunk length helps up to $h = 10$, after which the asymptotic performance starts to drop. Although $h = 25$ shows faster early learning, it fails to achieve the same performance as $h = 10$ at the end of online fine-tuning. An even larger chunk length ($h = 50$) fails to achieve any success. We suspect that overly large chunk sizes either hurt policy reactivity too much or make policy learning too difficult, as the network must predict a much longer action sequence at once. We use $h = 5$ in all our other experiments as $h = 5$ generally performs well and is cheaper to run. We also include the individual task breakdown in Figure 14 and additional ablation results on `cube-quadruple` in Figure 15. 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, increasing the update-to-data ratio (UTD) does not improve the sample efficiency of **QC** (Figure 6, right).

6 Discussions

We demonstrate how action chunking can be integrated into an offline-to-online RL agent with a simple recipe. Our approach speeds up value backup and explores more effectively online with temporally coherent actions. As a result, it outperforms prior offline-to-online methods on a range of challenging long-horizon tasks. Our work serves as a 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. 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. Developing practical techniques for training more general non-Markovian policies for online exploration would further improve the online sample efficiency of offline-to-online RL algorithms.

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. Finally, we would like to thank NeurIPS review commitee for thoughtful comments and suggestions.

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] Anita de Mello Koch, Akhil Bagaria, Bingnan Huo, Zhiyuan Zhou, Cameron Allen, and George Konidaris. Learning transferable sub-goals by hypothesizing generalizing features. 2025.

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

[17] Ishan P Durugkar, Clemens Rosenbaum, Stefan Dernbach, and Sridhar Mahadevan. Deep reinforcement learning with macro-actions. *arXiv preprint arXiv:1606.04615*, 2016.

[18] 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.

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

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

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

[31] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, 21, 2008.

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

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Ge Li, Dong Tian, Hongyi Zhou, Xinkai Jiang, Rudolf Lioutikov, and Gerhard Neumann. TOP-ERL: Transformer-based off-policy episodic reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=N4NhVN30ph.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] Amy McGovern and Richard S Sutton. Macro-actions in reinforcement learning: An empirical analysis. 1998.

[46] 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.

[47] 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.

[48] 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.

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

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

[51] 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.

[52] 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.

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

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

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

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

[57] 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.

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

[59] 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.

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

[61] 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.

[62] 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.

[63] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

[64] 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.

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

[66] Younggyo Seo, Jafar Uruç, and Stephen James. Continuous control with coarse-to-fine reinforcement learning. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=WjDR48cL3O.

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

[68] 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.

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

[70] 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.

[71] 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.

[72] 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.

[73] 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.

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

[75] 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.

[76] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36:11592–11620, 2023.

[77] 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.

[78] 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.

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

[80] 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.

[81] 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.

[82] 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.

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

[84] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W Anderson. Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13(2):259–284, 1993.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The claims in the abstract and introduction are supported by the results in Section 4 and Section 5.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

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

   Answer: [Yes]

   Justification: Yes, we discuss some limitations of our method in Section 6. Our assumptions and problem statement is included in Section 3.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: the proof of our theoretical result is available in Section A.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Yes, we provide the algorithm in Algorithm 1 and Algorithm 2, and implementation details, including hyperparameter choices in Section C. We provide a public repo (`github.com/ColinQiyangLi/qc`) of our code that reproduces results in Section 5.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

Answer: [Yes]

Justification: Yes, we have open-sourced our code at github.com/ColinQiyangLi/qc, which also includes instructions to download the data.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: we provide the details on the design choices in Section 5 and the hyperparameter values as well as the hyperparameter tuning process in Section C.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: All results in the paper have confidence intervals, denoting 95% confidence interval with four random seeds (five for robomimic results). We estimate the confidence intervals with bootstrapped sampling.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We provide additional information on computer resources in Section D.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We adhere to the NeurIPS code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [NA]

    Justification: Our work is on core reinforcement learning algorithm that has minimal societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [NA]

    Justification: Our paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: all code packages and datasets are properly cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: We will release code as an asset and provide detailed documentation.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

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

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

    Answer: [NA]

    Justification: We do not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [NA]

    Justification: We do not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [NA]

    Justification: We do not use LLM as an important component of this work.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

A Theoretical Justification

Proposition A.1 (Q-chunking performs unbiased n-step return backup)**.** *Let $s_t, a_t, \cdots, s_{t+n}$ be a trajectory segment generated by following a data collection policy $\pi_\beta(a_t, \cdots, a_{t+n}|s_t)$* (i.e., *$s_{t+k} \sim T(\cdot \mid s_{t+k-1}, a_{t+k-1}), \forall k \in \{1, \cdots, n\}$, and $r_t, r_{t+1}, \cdots r_{t+n-1}$ be the reward received at each corresponding time step* (i.e., *$r_{t+k} = r(s_{t+k}, a_{t+k})$). Let $V^\pi(s_t)$ be the value for an action chunking policy $\pi(a_t, \cdots, a_{t+n}|s_t)$ starting from state s_t. Let $Q^\pi(s_t, a_t, a_{t+1}, \cdots, a_{t+n-1})$ be the Q-value for π starting from state s_t and executing the action sequence $a_t, a_{t+1}, \cdots, a_{t+n-1}$.*

The n-step return estimate $\hat{V}_{\text{n-step}} := \sum_{t'=t}^{t+n-1} \left[\gamma^{t'-t} r_{t'}\right] + \hat{V}(s_{t+n})$ under the trajectory segment distribution described above is unbiased for $Q^\pi(s_t, a_t, \cdots, a_{t+n-1})$ as long as $\hat{V}(s_{t+n})$ is unbiased for $V^\pi(s_{t+n})$.

Proof. From the definition of $Q^\pi(s_t, a_t, a_{t+1}, \cdots, a_{t+n-1})$, we can write it as an expectation:

$$Q^\pi(s_t, a_t, a_{t+1}, \cdots, a_{t+n-1}) = \mathbb{E}_{s_{t'} \sim T(\cdot|s_{t'}, a_{t'})} \left[\sum_{t'=t}^{t+n-1} \left[\gamma^{t'-t} r(s_{t'}, a_{t'})\right] + V^\pi(s_{t+n}) \right] \tag{16}$$

$$= \mathbb{E}_{s_{t+n}, r_{t+1}, r_{t+2}, \cdots, r_{t+n-1}} \left[\sum_{t'=t}^{t+n-1} \left[\gamma^{t'-t} r_{t'}\right] + V^\pi(s_{t+n}) \right] \tag{17}$$

$$= \mathbb{E}_{s_{t+n}, r_{t+1}, r_{t+2}, \cdots, r_{t+n-1}} \left[\sum_{t'=t}^{t+n-1} \left[\gamma^{t'-t} r_{t'}\right] + \hat{V}(s_{t+n}) \right] \tag{18}$$

$$= \mathbb{E}\left[\hat{V}_{\text{n-step}}\right]. \tag{19}$$

The second line uses the fact that the rewards $r_t, r_{t+1}, \cdots, r_{t+h-1}$ are associated with the trajectory segment $s_t, a_t, \cdots, s_{t+n}$. The third line uses the fact that $\hat{V}(s_{t+n})$ is unbiased for $V^\pi(s_{t+n})$. The fourth line uses the definition of $\hat{V}_{\text{n-step}}$. □

The implication of this theorem is that if we use the n-step return backup for the chunked Q-function $Q(s_t, a_t, a_{t+1}, \cdots, a_{t+n-1})$ (with a value estimate of $\hat{V}_s \leftarrow Q(s_t, a_t, a_{t+1}, \cdots, a_{t+n-1})$), it converges to the ground truth Q-value under the same policy $Q^\pi(s_t, a_t, a_{t+1}, \cdots, a_{t+n-1})$. In contrast, using n-step return backup in the original action space does not converge to the correct Q-value (*i.e.*, $Q^\pi(s_t, a_t, a_{t+1}, \cdots, a_{t+n-1})$).

B Domain Details

See an overview of the six domains we use in our experiments in Figure 7. We also include the dataset size, episode length and the action dimension in Table 2. In the following sections, we describe each domain in details.

B.1 OGBench environments.

We consider five manipulation domains from OGBench [55] 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

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 2: **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).

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).

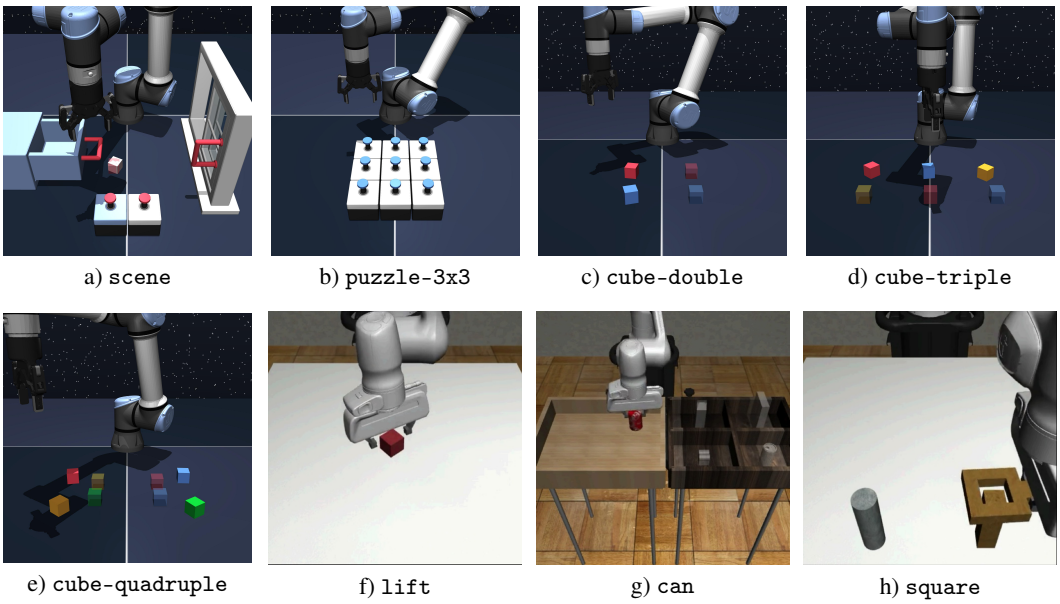
`puzzle-3x3-sparse`: This domain contains a 3×3 grid of buttons. Each button has two states represented by its color (blue or red). Pressing any button causes its color and the color of all its adjacent buttons to flip (red $\rightarrow$ blue and blue $\rightarrow$ red). The goal is to achieve a pre-specified 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 [43]. 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.



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

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

Figure 7: **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. [43].

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.

C Implementation Details

In this section, we provide more implementation details on both of our Q-chunking methods (QC, QC-FQL) and our baselines used in our experiments. For IFQL, ReBRAC, IQL, we directly use the implementation from Park et al. [58].

Algorithm 1 QC

Input: Behavior policy and critic: $f_\xi(\boldsymbol{a}_{t:t+h}|s)$ and $Q_\theta(s_t, \boldsymbol{a}_{t:t+h})$.
$\mathcal{D} \leftarrow$ offline prior data.
for every environment step t **do**
  if $t \mod h \equiv 0$ **then**
    $\boldsymbol{a}^1_{t:t+h} \cdots \boldsymbol{a}^N_{t:t+h} \sim f_\xi(\cdot|s_t)$
    $\boldsymbol{a}^\star_{t:t+h} \leftarrow \arg\max_{\boldsymbol{a}^i_{t:t+h}} Q_\theta(s, \boldsymbol{a}^i_{t:t+h})$
  Act with $a^\star_t$ and receive s_{t+1}, r_t.
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a^\star_t, s_{t+1}, r_t)\}$
  Update f_ξ via flow-matching loss using $\mathcal{D}$.
  Update Q_θ via Eq (11) using $\mathcal{D}$.
Output: f_ξ, Q_θ.

Algorithm 2 QC-FQL

Input: Behavior policy, critic, one-step policy: $f_\xi(\boldsymbol{a}_{t:t+h}|s)$, $Q_\theta(s_t, \boldsymbol{a}_{t:t+h})$, $\mu_\psi(s, \boldsymbol{z})$.
$\mathcal{D} \leftarrow$ offline prior data.
for every environment step t **do**
  if $t \mod h \equiv 0$ **then**
    $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I}_{Ah})$
    $\boldsymbol{a}_{t:t+h} \leftarrow \mu_\psi(s_t, \boldsymbol{z})$
  Act with a_t and receive s_{t+1}, r_t.
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, s_{t+1}, r_t)\}$
  Update f_ξ via flow-matching loss using $\mathcal{D}$.
  Update μ_ψ and Q_θ via Eq. (13, 15) using $\mathcal{D}$.
Output: $f_\xi, Q_\theta, \mu_\psi(s, \boldsymbol{z})$.

C.1 QC-FQL

We build the implementation of our method on top of **FQL** [58], 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 3 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:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t, \cdots, a_{t+h-1}) - r_t^h - \frac{1}{K} \sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+h}, a_{t+h}, \cdots, a_{t+2h-1}) \right)^2, \quad (20)$$

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:

$$L(\psi, w) = -Q_\theta(s_t, \mu_\psi(s_t, z_t)) + \alpha \left\| \mu_\psi(s_t, z_t) - \begin{bmatrix} a_t^\xi & \cdots & a_{t+h-1}^\xi \end{bmatrix} \right\|_2^2, \quad (21)$$

where $z_t \sim \mathcal{N}(0, I_{Ah})$ and $\begin{bmatrix} a_t^\xi & \cdots & a_{t+h-1}^\xi \end{bmatrix}$ is the result of running the behavior policy $f_\xi(s, m, t)$ with Algorithm 3 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:

$$L(\xi, w) = \| f_\xi(s_t, u [a_t \quad \cdots \quad a_{t+h-1}] + (1-u) z_t, u) - ([a_t \quad \cdots \quad a_{t+h-1}] - z_t) \|_2^2, \quad (22)$$

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: $\mathcal{L}(\theta) = \frac{1}{M} \sum_{i=1}^{M} \sum_{k=1}^{K} \mathcal{L}(\theta_k, w_i), \mathcal{L}(\psi) = \frac{1}{N} \sum_{i=1}^{M} \mathcal{L}(\psi, w_i), \mathcal{L}(\xi) = \frac{1}{N} \sum_{i=1}^{M} \mathcal{L}(\xi, w_i)$.

C.2 FQL

FQL [58] 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 $w = (s_t, a_t, r_t, s_{t+1})$:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t) - r_t - \frac{1}{K} \sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+1}, \mu_\psi(s_{t+1}, z_t^{k'})) \right)^2, z_t^{k'} \sim \mathcal{N}(0, I_A), \quad (23)$$

$$L(\psi, w) = -Q_\theta(s_t, \mu_\psi(s_t, z_t)) + \alpha \left\| \mu_\psi(s_t, z_t) - a_t^\xi \right\|_2^2, \quad (24)$$

$$a_t^\xi \leftarrow \texttt{FlowODE\_Euler}(s_t, z_t, f_\xi, T), z_t \sim \mathcal{N}(0, I_A), \quad (25)$$

$$L(\xi, w) = \| f_\xi(s_t, u a_t + (1-u) z_t, u) - (a_t - z_t) \|_2^2, z_t \sim \mathcal{N}(0, I_A), u \sim U([0, 1]). \quad (26)$$

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

Algorithm 3 `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.

C.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:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t) - \sum_{t'=0}^{h-1} (\gamma^{t'} r_{t+t'}) - \frac{1}{K} \sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+h}, \mu_\psi(s_{t+h}, z_t^{k'})) \right)^2, \tag{27}$$

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.

C.4 QC

The flow-matching behavior policy is trained with the same loss as used in **QC-FQL** (Equation (22)). 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:

$$L(\theta_k, w) = \left(Q_{\theta_k}(s_t, a_t, \cdots, a_{t+h-1}) - r_t^h - \frac{1}{K} \sum_{k'=1}^{K} Q_{\bar{\theta}_{k'}}(s_{t+h}, a_{t+h}^{i^\star}, \cdots, a_{t+2h-1}^{i^\star}) \right)^2 \tag{28}$$

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.

C.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):

$$\mathcal{L}(\psi) = \mathbb{E}_{a'_t \sim \pi_\psi(\cdot|s_t)} \left[-\frac{1}{K} \sum_{k=1}^{K} Q_{\theta_k}(s_t, a'_t) \, -\alpha \log \pi_\psi(a_t|s_t) \right]. \tag{29}$$

C.6 SUPE-GT

For this baseline, we adapt from a recently proposed skill-based offline-to-online RL method [85]. The original method has additional modules such reward models and random distillation networks to deal with the problem setting where offline dataset is unlabeled (*e.g.*, reward information is missing). We remove these two modules as we have ground truth reward information in our offline dataset. The method has two stages: (1) skill-pretraining and (2) online RL with skills. In the first stage, the baseline uses a trajectory VAE to learn latent-conditioned skill policy. In the second stage, the baseline uses RLPD to learn a high-level policy that outputs latent vector to command the latent-conditioned skill policy after every h steps. Such a skill-based method in theory also allows temporally coherent actions for exploration and value backup speedup.

D Experiment Details

D.1 Computational resources

We use NVIDIA RTX-A5000 GPU to run all our experiments. Each complete offline-to-online experiment run takes around 4-7 hours. To reproduce all our results in Table 6, we estimate that it would take around $\underbrace{6}_{\text{hours per single run}} \times \underbrace{15}_{\text{\# of methods}} \times \underbrace{25}_{\text{\# of tasks}} \times \underbrace{4}_{\text{\# of seeds}} = 9\,000$ GPU hours. Each robomimic experiment run takes longer (around 8-12 hours). While each single run takes longer, we only run a subset of strong methods/baselines on robomimic tasks. We estimate it would take around $\underbrace{10}_{\text{hours per single run}} \times \underbrace{8}_{\text{\# of methods}} \times \underbrace{3}_{\text{\# of tasks}} \times \underbrace{5}_{\text{\# of seeds}} = 1\,350$ GPU hours. In total, we estimate that it would take around 10 350 GPU hours to reproduce all the main results in our paper.

D.2 Evaluation protocol

Unless specified otherwise, for all methods, we run 4 seeds on each OGBench task and 5 seeds on each robomimic 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).

D.3 Hyperparameter tuning

QC, **BFN**, **BFN-n**. We tune the number of actions sampled, N, for the expected-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 Table 1 and Figure 1 (on all tasks). Table 5 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. [58]). 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 in Table 1 and Figure 1 (on all tasks). Table 4 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].

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**, and **SUPE-GT** 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 3: **Common hyperparameters.**

Environments	FQL	FQL-n	QC-FQL	ReBRAC
`scene-sparse-*`	300	100	300	0.1
`puzzle-3x3-sparse-*`	100	100	300	0.1
`cube-double-*`	300	100	300	0.1
`cube-triple-*`	300	100	100	0.1
`cube-quadruple-100M-*`	300	100	100	0.1
`lift`	10000	10000	10000	-
`can`	10000	10000	10000	-
`square`	10000	10000	10000	-

Table 4: **Behavior regularization coefficient (α).**

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.

SUPE-GT. We tune the KL coefficient for the VAE skill-pretraining from $\{0.001, 0.003, 0.01, 0.03, 0.1\}$ and pick 0.003 as it works the best.

ReBRAC. We tune the behavior regularization coefficient from $\{0.01, 0.03, 0.1, 0.3, 1.0\}$ and pick 0.1 as it works the best.

IFQL. We directly use the default hyperparameter from Park et al. [58] ($N = 32, \tau = 0.9$) for this baseline. The first parameter, $N = 32$, means that for both online policy rollout and policy evaluation, we sample 32 actions and pick the action that has the highest Q-value. The second parameter, $\tau = 0.9$ is the expectile coefficient.

IQL. We directly use the default hyperparameter from Park et al. [58] for this baseline. For `cube-*`, we use $\alpha = 0.3$. For `scene` and `puzzle-3x3`, we use $\alpha = 10.0$. For the expectile coefficient, we use $\tau = 0.9$.

E Full Results

E.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 8, we show the first 9000 time steps (broken down into 9 subplots where each visualizes 1000 time steps). In Figure 9, 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.

Environments	BFN	BFN-n	QC-BFN
`scene-sparse*`	4	4	32
`puzzle-3x3-sparse-*`	4	4	64
`cube-*`	4	4	32
`lift, can, square`	4	4	16

Table 5: **Number of actions sampled for the expected-max Q operator (N) for BFN methods.**

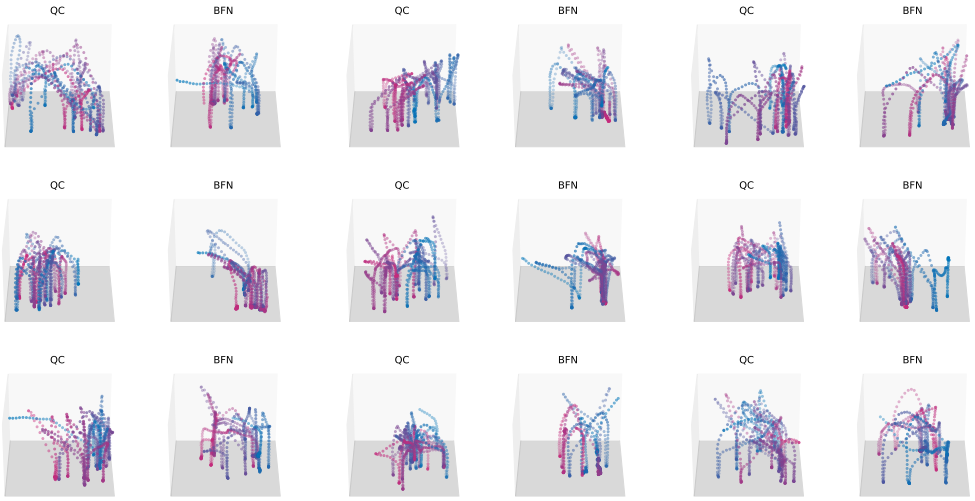


Figure 8: **End-effector trajectory early in the training.** Each subplot above shows the trajectory for a consecutive of 1000 time steps. We include up to Step 9000.

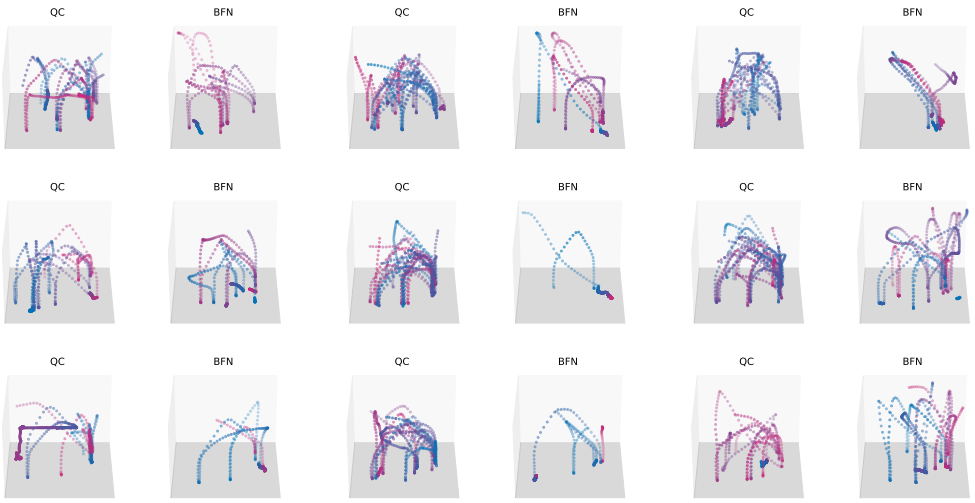


Figure 9: **End-effector trajectory visualization late in the training.** Each subplot above shows the trajectory for a consecutive of 1000 time steps (from Step 900000 to Step 909000).

E.2 OGBench results by individual task

Main results by task. Table 1 and Figure 10 shows the performance breakdown for all methods.

		RLPD	RLPD-AC	SUPE-GT	IQL	ReBRAC	IFQL	FQL	BFN	IFQL-n	FQL-n	BFN-n	QC-RLPD	QC-IFQL	QC-FQL	QC
puzzle-3x3	task1	- → **100**	- → **100**	- → **100**	0 → **99**	96 → **100**	100 → **100**	99 → **100**	98 → **100**	**97** → **100**	100 → **100**	88 → **100**	- → **100**	**100** → **100**	**100** → **100**	**100** → **100**
	task2	- → **100**	- → **100**	- → **100**	0 → 0	70 → **100**	98 → **100**	99 → **100**	98 → **100**	98 → **100**	100 → **100**	72 → **100**	- → **100**	**100** → **100**	**86** → **100**	**100** → **100**
	task3	- → **100**	- → **100**	- → **100**	0 → 0	27 → **100**	**100** → 98	**100** → **100**	95 → **100**	95 → **100**	100 → **100**	56 → 82	- → **100**	**100** → **100**	50 → **100**	**100** → **100**
	task4	- → **100**	- → **100**	- → **100**	0 → 0	29 → **100**	97 → 87	**100** → **100**	98 → **100**	86 → **100**	100 → **100**	37 → 66	- → **100**	**100** → **100**	**75** → **100**	**100** → **100**
	task5	- → **100**	- → **100**	- → **100**	0 → 0	54 → **100**	**98** → **99**	**100** → **100**	97 → **100**	96 → **100**	**91** → **100**	47 → **100**	- → **100**	**100** → **100**	4 → **100**	**100** → **100**
	avg. (5 tasks)	- → **100**	- → **100**	- → **100**	0 → 20	55 → **100**	**98** → 97	**100** → **100**	98 → **100**	94 → **100**	**98** → 90	58 → 90	- → **100**	**100** → **100**	63 → **100**	**100** → **100**
scene	task1	- → **100**	- → **100**	- → **100**	1 → 97	4 → **100**	98 → **100**	69 → **100**	99 → **100**	86 → **99**	22 → **100**	64 → **100**	- → **100**	93 → **100**	**99** → **100**	**100** → **100**
	task2	- → **100**	- → **100**	- → **97**	0 → **97**	51 → **100**	89 → **93**	51 → **99**	**99** → **99**	52 → **97**	4 → 42	15 → 98	- → **100**	68 → **100**	88 → **100**	**98** → **100**
	task3	- → **99**	- → **100**	- → **97**	0 → 0	0 → **99**	49 → 88	68 → **100**	93 → **100**	38 → 97	1 → 24	34 → 97	- → **100**	65 → **99**	**98** → **100**	**90** → **100**
	task4	- → 89	- → 92	- → **95**	0 → **98**	73 → 86	73 → **96**	**86** → **97**	81 → **97**	30 → **94**	**92** → **94**	- → **97**	**93** → **98**	**92** → **97**	**93** → **99**
	task5	- → 82	- → 74	- → **75**	0 → 0	0 → **97**	32 → 1	25 → 81	38 → **99**	**83** → 72	14 → **91**	71 → **98**	- → 82	**93** → 91	41 → **99**	43 → **95**
	avg. (5 tasks)	- → 94	- → 91	- → **92**	0 → 39	11 → **99**	68 → 73	57 → 95	**85** → **99**	68 → 93	18 → 70	57 → **97**	- → 96	**83** → **98**	**84** → **99**	**84** → **99**
cube-double	task1	- → **99**	- → **99**	- → **100**	1 → 0	12 → **99**	24 → 98	63 → **99**	**70** → **97**	9 → 89	33 → **99**	23 → **100**	- → **100**	17 → **100**	66 → **100**	**84** → **99**
	task2	- → **99**	- → **98**	- → 77	0 → 0	0 → 16	11 → 63	36 → 92	**79** → 88	5 → 12	8 → 96	8 → 65	- → **99**	15 → **100**	43 → **100**	**79** → **100**
	task3	- → **99**	- → **99**	- → 83	0 → 0	0 → 34	6 → 61	22 → **96**	**83** → 87	5 → 10	1 → 92	7 → 73	- → **99**	8 → 92	38 → **99**	**68** → **99**
	task4	- → **99**	- → 97	- → 5	0 → 0	0 → 0	1 → 1	9 → 1	**22** → 38	1 → 0	0 → 2	1 → 0	- → **98**	5 → 3	**11** → **100**	**22** → 92
	task5	- → **96**	- → 88	- → 85	0 → 0	1 → 1	14 → 72	12 → 92	**82** → 96	4 → 35	13 → **96**	13 → 78	- → 88	17 → **98**	37 → **99**	**74** → **99**
	avg. (5 tasks)	- → **99**	- → 96	- → 67	0 → 0	3 → 30	11 → 59	29 → 76	**68** → 80	5 → 29	11 → 77	11 → 65	- → 96	12 → 79	39 → **100**	**67** → **98**
cube-triple	task1	- → **100**	- → 51	- → 0	0 → 1	1 → 1	2 → 16	5 → 87	**17** → 97	1 → 0	0 → 3	2 → 1	- → 45	0 → 0	**19** → **100**	**13** → **100**
	task2	- → **61**	- → 1	- → 0	0 → 0	0 → 0	0 → 0	**1** → 0	**1** → 8	0 → 0	0 → 0	0 → 0	- → 8	0 → 0	**1** → **89**	**1** → **89**
	task3	- → 1	- → 2	- → 0	0 → 0	0 → 0	0 → 0	0 → 5	**1** → 11	0 → 0	0 → 0	0 → 0	- → 1	0 → 0	0 → **51**	**7** → **73**
	task4	- → **43**	- → 0	- → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 1	0 → 0	0 → 0	0 → 0	- → 0	0 → 0	0 → **26**	0 → **54**
	task5	- → 0	- → 0	- → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	- → 0	0 → 0	0 → 0	0 → 0
	avg. (5 tasks)	- → 41	- → 11	- → 0	0 → 0	0 → 0	0 → 3	1 → 18	**4** → 23	0 → 0	0 → 1	0 → 0	- → 11	0 → 0	**4** → **53**	**6** → **64**
cube-quadruple-100M	task1	- → 2	- → 7	- → 0	0 → 0	7 → **98**	0 → 0	1 → 14	7 → 56	1 → 0	**28** → **97**	0 → 0	- → 37	4 → 96	6 → **98**	**16** → **99**
	task2	- → 0	- → 26	- → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 3	0 → 0	**2** → 49	0 → 0	- → 17	0 → 0	0 → **97**	0 → **98**
	task3	- → 0	- → 3	- → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	**5** → 4	0 → 0	- → 1	0 → 0	0 → **94**	**3** → 78
	task4	- → 0	- → 4	- → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	**1** → 32	0 → 0	- → 2	0 → 0	0 → **94**	**1** → **88**
	task5	- → 0	- → 0	- → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	0 → 0	- → 0	0 → 0	0 → 0	0 → 0
	avg. (5 tasks)	- → 0	- → 7	- → 0	0 → 0	1 → 20	0 → 0	0 → 3	1 → 12	0 → 0	**7** → 36	0 → 0	- → 11	1 → 19	1 → **77**	**4** → 74
overall	avg. (25 tasks)	- → 67	- → 61	- → 52	0 → 12	14 → 50	36 → 47	37 → 58	**51** → 63	34 → 44	27 → 57	25 → 50	- → 63	39 → 59	38 → **86**	**52** → **86**

Table 6: **Complete OGBench offline-to-online RL results.** 4 seeds. 95% confidence interval.

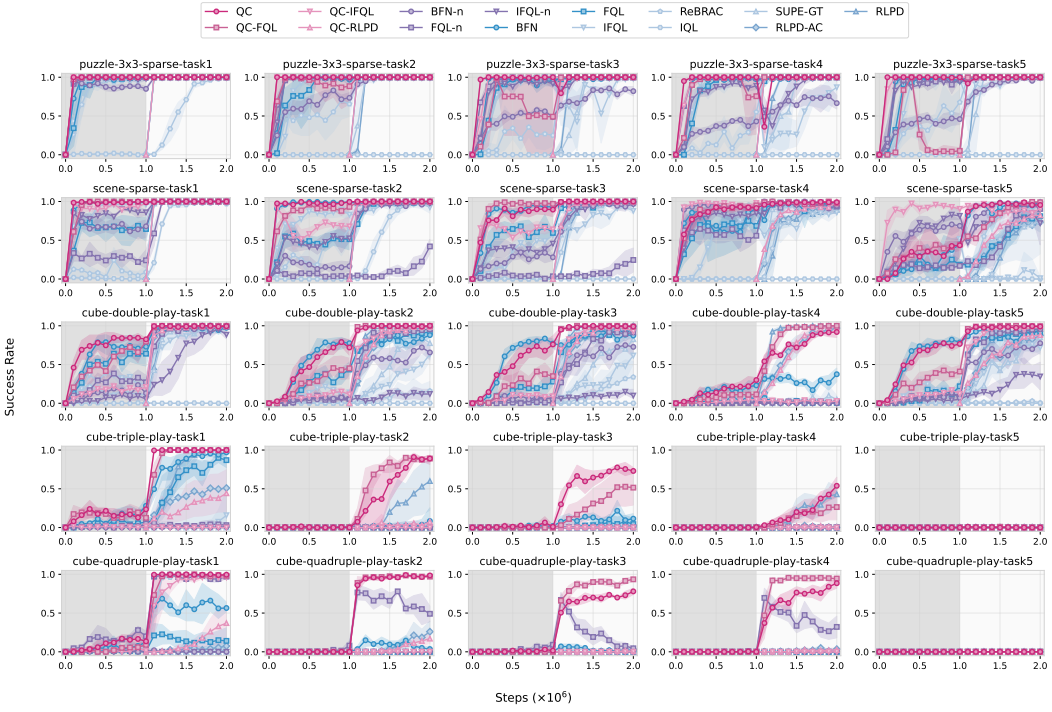


Figure 10: **Complete OGBench offline-to-online RL results by task. 4 seeds. 95% confidence interval.**

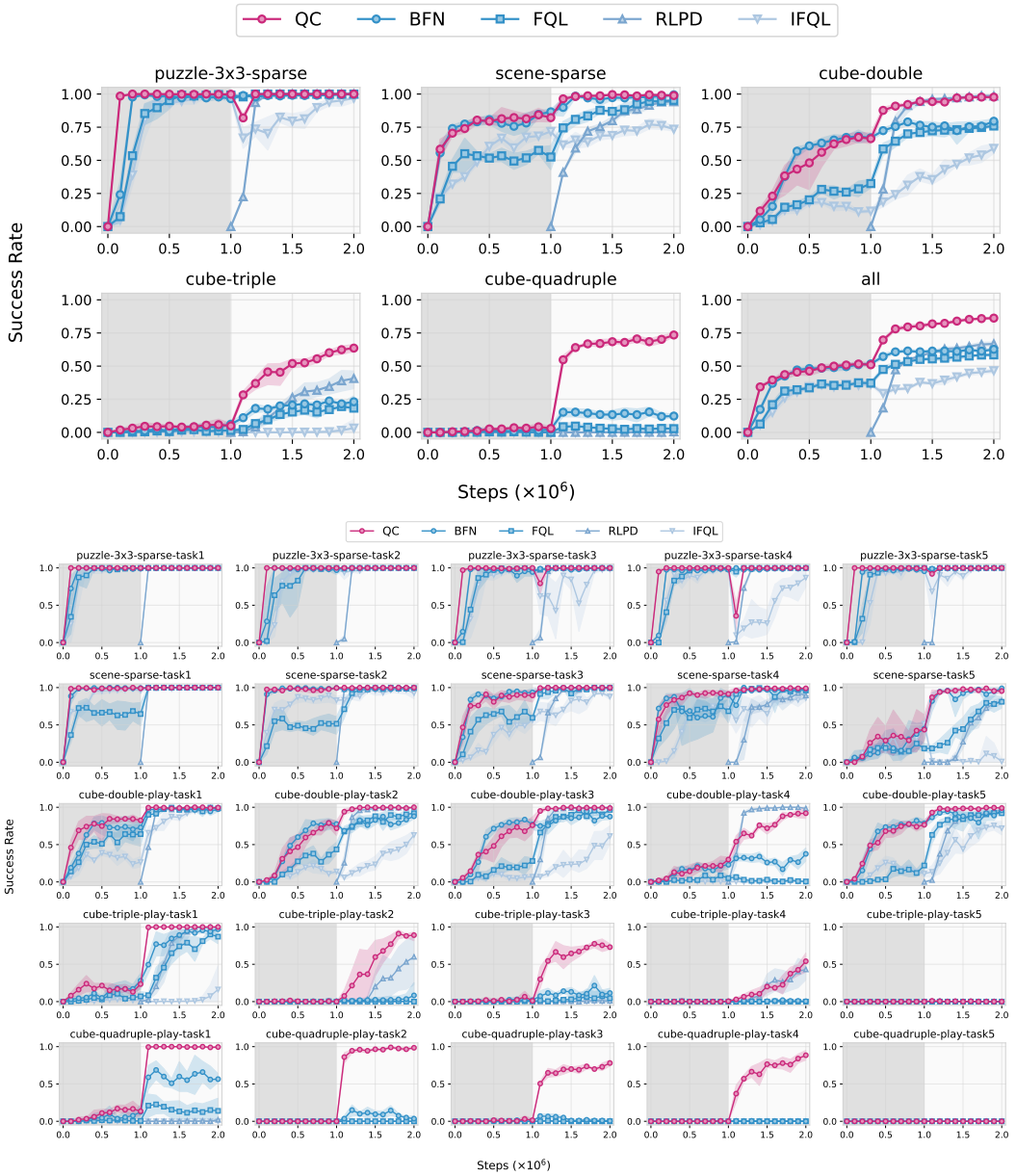


Figure 11: **Full OGBench results by task (selected baselines).** *Top:* summary plots by domain; *Bottom:* individual plots by task.

Selected baselines. Figure 11 shows the results for selected baselines (*e.g.*, BFN, FQL, RLPD and RLPD-AC).

n-step return ablation results by task. Figure 12 shows the performance breakdown for Figure 4.

Q-chunking with Gaussian 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).

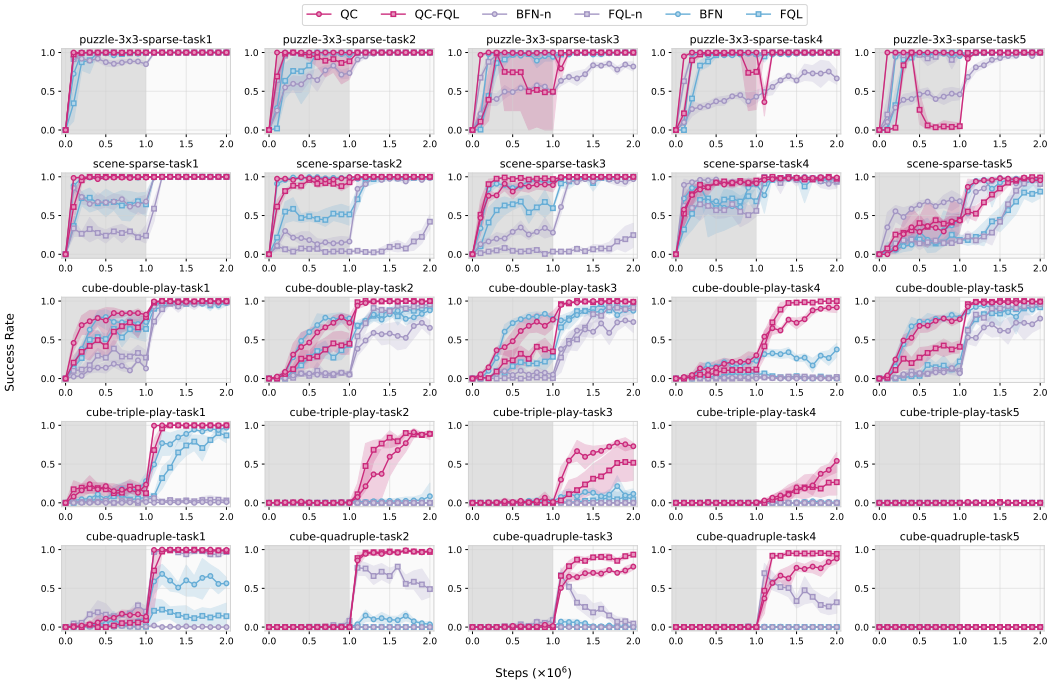


Figure 12: **Full OGBench results by task (selected baselines).**

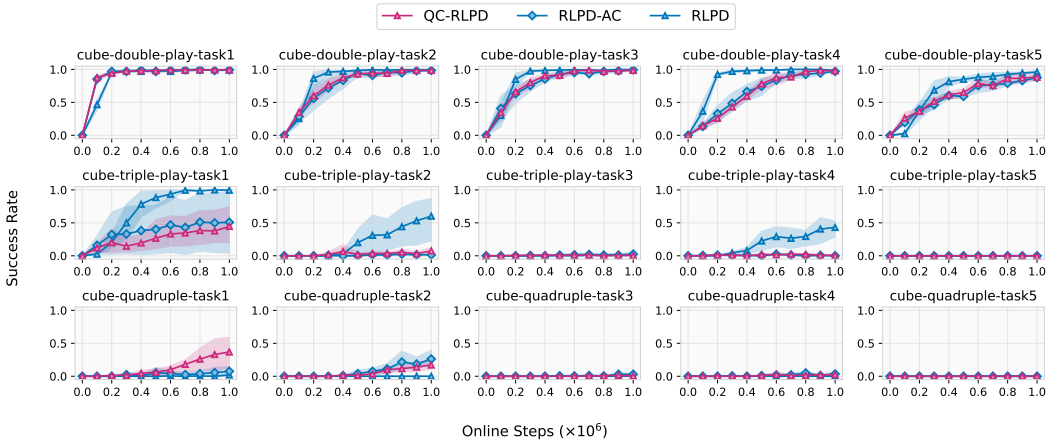


Figure 13: **Full RLPD results by task.** QC-RLPD is RLPD-AC (RLPD on the temporally extended action space) where we additionally add a fixed behavior cloning coefficient of 0.01.

Action chunking size ablations. Figure 14 includes individual task results for the action chunking size ablation study. We also include results on the hardest `cube-quadruple` task in Figure 15.

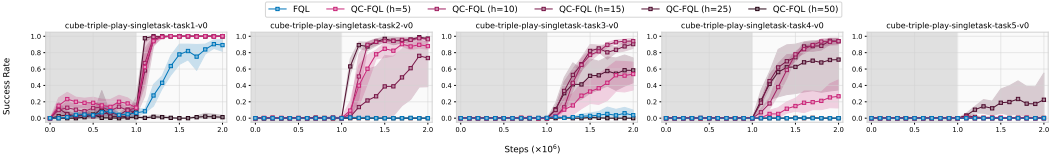


Figure 14: **Action chunking size ablation on `cube-triple-*`.** Increasing the action chunking size generally helps until $h = 25$. Surprisingly, $h = 25$ is the only chunk size where QC-FQL achieves non-trivial success on the hardest `cube-triple-play-task5` (none of other methods can).

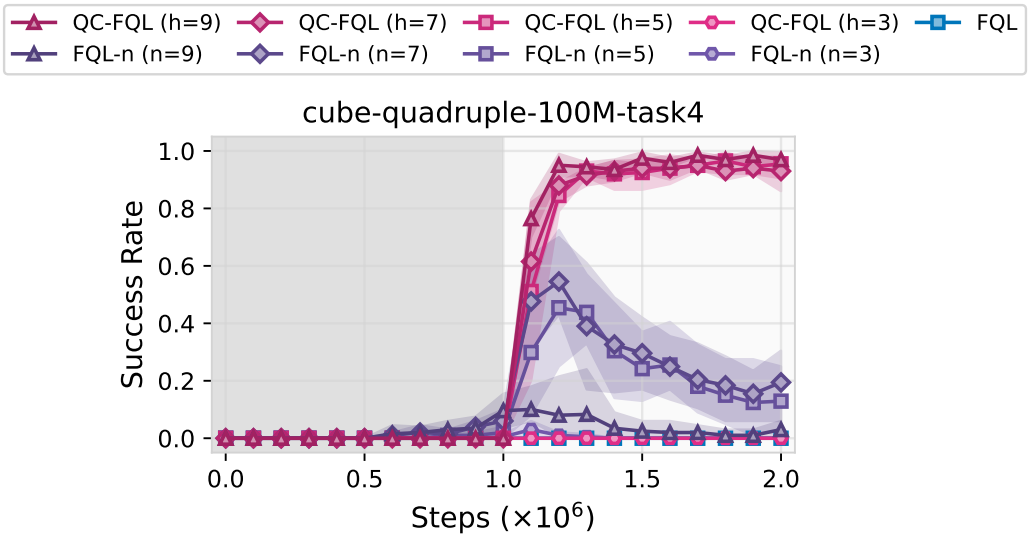


Figure 15: **Action chunking size ablation on `cube-quadruple-play-task4`.** The n-step return baselines (*i.e.*, **FQL-n**) can achieve good initial success during online learning but quickly collapse. In contrast, our method (*i.e.*, **QC-FQL**) does not suffer from such collapse and solves the task consistently across reasonably large chunk sizes (*e.g.*, $h \in \{5, 7, 9\}$). The results are over 5 seeds.

E.3 Robomimic ablation results

Figure 16 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).

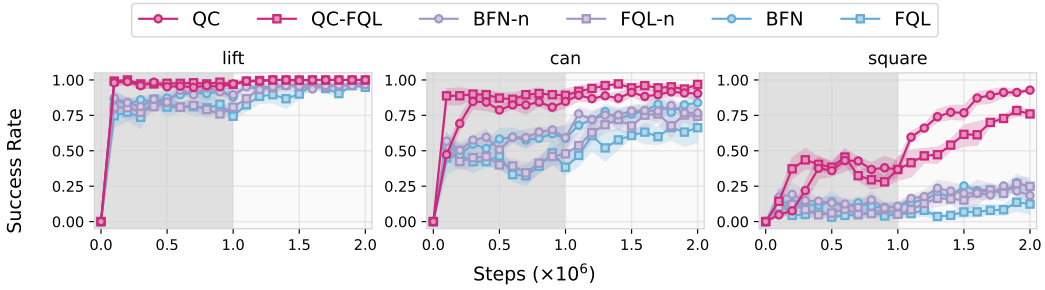


Figure 16: **Full robomimic ablation by task.** For each method on each task, we use 5 seeds.

E.4 How resource efficient is Q-chunking?

In Figure 17, 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.

Finally, we include the parameter count of each of these method on the representative domain `cube-triple-*` in Table 7 below (assuming $h = 5$ for all Q-chunking methods).

Methods	Parameter Count (in millions)
QC	≈ 4.2
BFN	≈ 4.1
QC-FQL	≈ 5.0
FQL	≈ 4.9
RLPD	≈ 17.2

Table 7: **Parameter count for each method.** RLPD has a much larger parameter count because it uses $K = 10$ critic networks whereas all the other methods and baselines use only $K = 2$.

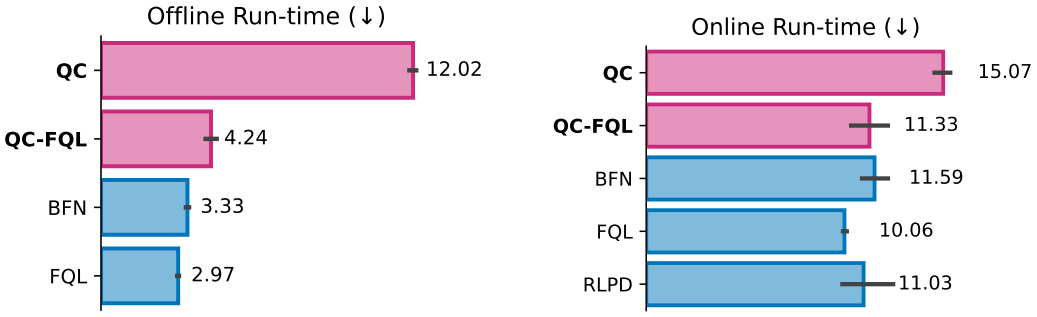


Figure 17: **How long does each method take for one step in milliseconds.** Left: offline. Right: online (one agent training step and an environment step). The runtime is measured using the default hyperparameters in our paper on `cube-triple-task1` on a single RTX-A5000.