When Life Gives You BC, Make Q-functions: Extracting Q-values from Behavior Cloning for On-Robot Reinforcement Learning

Lakshita Dodeja[1,2], Ondrej Biza[1], Shivam Vats[2], Stephen Hart[1], Stefanie Tellex[1,2], Robin Walters[3], Karl Schmeckpeper[1], and Thomas Weng[1]

[1]Robotics and AI Institute, Cambridge, MA, USA
[2]Brown University, Providence, RI, USA
[3]Northeastern University, Boston, MA, USA

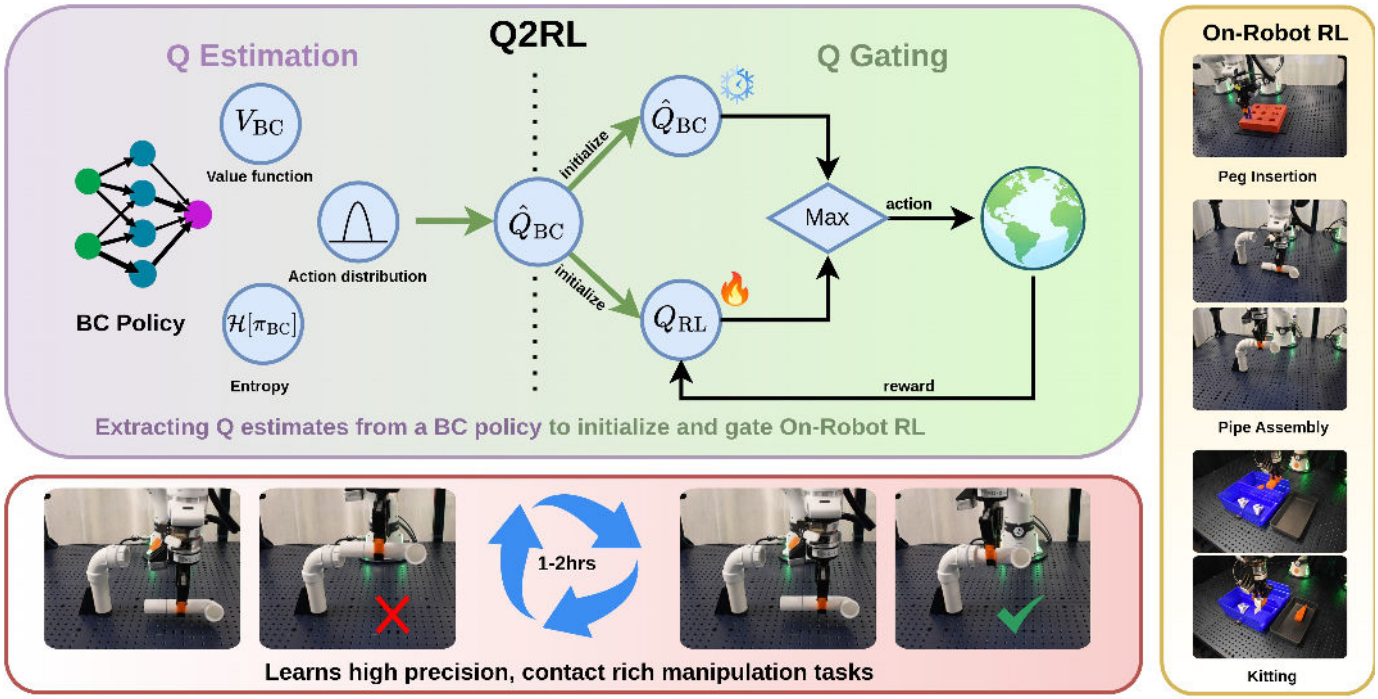


Fig. 1: *Q2RL* consists of Q-Estimation and Q-Gating. Q-Estimation extracts a Q-function from a BC Policy. Then, during RL policy training, Q-Gating selects and executes the BC or RL action with highest respective Q-value, updating the RL policy on the collected interactions. *Q2RL* learns contact-rich manipulation tasks in 1-2 hours of online interaction.

***Abstract*—Behavior Cloning (BC) has emerged as a highly effective paradigm for robot learning. However, BC lacks a self-guided mechanism for online improvement after demonstrations have been collected. Existing offline-to-online learning methods often cause policies to replace previously learned good actions due to a distribution mismatch between offline data and online learning. In this work, we propose *Q2RL*, Q-Estimation and Q-Gating from BC for Reinforcement Learning, an algorithm for efficient offline-to-online learning. Our method consists of two parts: (1) *Q-Estimation* extracts a Q-function from a BC policy using a few interaction steps with the environment, followed by online RL with (2) *Q-Gating*, which switches between BC and RL policy actions based on their respective Q-values to collect samples for RL policy training. Across manipulation tasks from D4RL and robomimic benchmarks, *Q2RL* outperforms SOTA offline-to-online learning baselines on success rate and time to convergence. *Q2RL* is efficient enough to be applied in an on-robot RL setting, learning robust policies for contact-rich and high precision manipulation tasks such as pipe assembly and kitting, in 1-2 hours of online interaction, achieving success rates of up to 100% and up to 3.75x improvement against the original BC policy. Code and video are available at https://q2rl.rai-inst.com/.**

I. Introduction

Behavior Cloning (BC) has been highly successful in robot learning due to its simple supervised training objective, which directly models an action distribution conditioned on observed states. However, BC does not have a mechanism for self-guided online improvement. When the training data for a BC policy is insufficient for the scope of the task, and/or changes in the environment induce even subtle distribution shifts, BC policy performance can collapse due to compounding errors from covariate shift [30].

Several research directions have been proposed to address the shortcomings of behavior cloning. Interactive imitation

learning approaches finetune BC policies online, but require expert annotators [15]. Offline reinforcement learning (RL) learns a value function offline [1, 16] to guide online learning, but require pretraining on large datasets of positive and negative offline samples [18] for good performance. Offline-to-online learning approaches have also been proposed, but they often result in unlearning good BC actions [27, 39]. Another challenge faced by offline-to-online methods is the tendency to produce unsafe behaviors due to training an RL policy from scratch. In the on-robot RL setting in which online RL is trained directly on the robot, executing random exploratory actions is unsafe, causing significant wear and tear on the robot, workspace, and manipulated objects.

In this paper, our objective is to combine the benefits of pre-training a BC policy with the iterative improvement capability of online RL. We aim to keep good BC policy behavior, and when BC performs poorly, post-train with RL to find better actions. We propose Q-Estimation and Q-Gating from BC for Reinforcement Learning (*Q2RL*), an algorithm for efficient offline-to-online learning. In the *Q-Estimation* phase, we estimate the Q-function of the pretrained BC policy from initial online rollouts, then use it to initialize off-policy reinforcement learning. We are able to derive these estimated Q-values with just the action selection probability and entropy of the BC policy, *without any additional information of the underlying policy class or training distribution*. To ensure a stable transition from offline pretraining to online RL, we propose *Q-Gating*, which takes the Q-function from the Q-Estimation phase and uses it to guide online reinforcement learning. Two copies of the Q-function are used: the initially extracted Q function is kept frozen to preserve good BC actions, and a learnable RL Q-function is optimized to search for potentially better actions. At each state, *Q2RL* compares these Q-estimates to selectively execute BC or RL actions, ensuring that the RL policy is able to initialize learning from good BC actions and explore online to improve beyond the BC policy.

We provide extensive experiments in simulation and real-world on-robot RL to validate the method. *Q2RL* outperforms SOTA offline RL and offline-to-online methods across D4RL and robomimic benchmark tasks, improving the success rates from 50-60% for the original BC policy up to 80-100% for most environments. Ablation experiments show that both Q-Estimation and Q-Gating are valuable components of the algorithm. Our real-world experiments show that *Q2RL* learns contact-rich, high-precision manipulation tasks in 1-2 hours of online interaction, outperforming baselines especially as tasks increase in difficulty, and improving the success rate compared to the original BC policies by up to 100% and up to 3.75x improvement. Furthermore, our real-world experiments demonstrate that *Q2RL* can recover from distribution shifts in the environment.

Our contributions are as follows:

- "Q-Estimation", an algorithm for estimating a Q-function from a behavior cloning policy and a few online rollouts. Q-Estimation can be applied to any policy class that provides action likelihoods and entropy.
- "Q-Gating", an algorithm that guides online reinforcement learning via a gating mechanism to select between BC and RL actions.
- Experiments demonstrating that *Q2RL*, which combines Q-Estimation and Q-Gating, outperforms baselines on D4RL and robomimic benchmarks.
- Real-world, on-robot reinforcement learning experiments with *Q2RL* that surpass the original BC policy's performance on contact-rich manipulation tasks and adapt to task distribution shifts unseen by the BC policy, in 1-2 hours of environment interaction.

II. Related Work

We review existing approaches to offline learning such as Behavior Cloning and offline RL, as well as offline-to-online approaches that combine offline learning with online RL. Then, we review on-robot reinforcement learning, which require sample-efficient techniques for real-world learning.

A. Offline Learning

Offline learning in robotics is broadly divided into Imitation Learning based approaches [38, 25] and Offline Reinforcement Learning approaches [16, 18]. Imitation learning approaches that train policies via supervised learning (i.e., Behavior Cloning) learn a direct mapping from states to an action distribution using demonstration data. Offline RL methods learn value functions that estimate expected returns from data generated by a behavior policy. Offline RL techniques like CQL [16] penalize out-of-distribution actions to address dataset shift, but this can introduce pessimistic bias in value estimates. IQ-Learn [10] learns soft Q-functions directly from demonstrations, enabling policy extraction through RL objectives. In contrast, our approach does not learn a new Q-function from demonstrations, but instead estimates the Q-values of an existing black-box BC policy to guide online improvement. Furthermore, offline RL methods are more suited to learn from large and potentially suboptimal datasets [9], and not necessarily from robotics datasets where successful demonstrations are easier to curate [28, 5]. BC approaches can further be divided into policy classes that explicitly parameterize the action distribution, such as Gaussian or Gaussian mixture heads paired with any network backbone [26, 32], and policy classes that implicitly represent the action distribution using generative denoising processes such as diffusion [6, 35] or flow-based models [19, 13]. *Q2RL* only requires access to the action selection probabilities and the entropy, which is easier to obtain for policy classes that explicitly parameterize the action distribution.

B. Offline-to-Online Learning

Offline learning can provide strong initial performance from readily available datasets, but policies trained purely offline cannot improve further without additional supervision or online interaction [15, 31]. Calibrated Q-Learning (CalQL) [27] aims to enable a smooth transition from offline to online RL

by calibrating offline Q-values against a reference policy, but can struggle with limited offline data. WSRL [39] mitigates this by adding a warm-up phase that seeds the online replay buffer with rollouts from the offline policy. In contrast, other methods bypass offline RL by first pretraining an IL policy, then updating the behavior of the agent by adding corrective residual terms to the output action [33, 14, 37, 7]. These approaches require task-specific tuning of the residual action scale to ensure that the residual policy does not deviate from the BC policy, and can fail to optimize large action parameterizations. *Q2RL* does not use a residual formulation and allows the RL policy to learn completely different actions from the original BC policy, while still relying on the BC policy for guidance. Imitation Bootstrapped RL (IBRL) [12], most closely related to our approach, also combines a pretrained IL/BC policy with an RL policy and selects between their actions using a randomly initialized critic. A key benefit of *Q2RL* is that it *estimates the Q-function of the BC policy* from early online interaction, and uses it in conjunction with an RL Q-function to evaluate actions during online, off-policy reinforcement learning. In our evaluations, we show that *Q2RL* outperforms IBRL in simulated and real-world experiments, and that *Q2RL* does not require seeding the online replay buffer with high quality demonstrations, unlike IBRL.

C. On-Robot Reinforcement Learning

Online reinforcement learning on a real robot is a challenging but promising post-training strategy for achieving robust robotic policies. Since random exploration is costly on hardware, practical methods must begin with reasonable initial performance and adapt sample-efficiently. SERL [21] introduces a system designed specifically for on-robot RL. It provides a software suite to run online RL algorithms [11, 36, 2] using async actor and learner processes [34]. Extensions like HIL-SERL [23] incorporate human interventions to further improve efficiency, while other approaches like GCR [3] use reward shaping to guide learning. *Q2RL* uses the async framework for on-robot RL from SERL, but leverages readily available BC policies to provide strong initial behavior, without additional reward engineering or human intervention.

III. Problem Formulation

Assume we are given a behavior cloning policy π_{BC}, pretrained on a dataset $\mathcal{D}$ of successful human demonstrations for a task $\mathcal{M}_{\mathrm{train}} \in \mathfrak{M}$. Here $\mathfrak{M}$ denotes a class of tasks that share the same success condition; in the case of task distribution shift, $\mathcal{M}_{\mathrm{test}}$ and $\mathcal{M}_{\mathrm{train}}$ are drawn from task class $\mathfrak{M}$. The dataset consists of M trajectories of observation-action tuples:

$$\mathcal{D} = \{\tau_j\}_{j=1}^{M}, \qquad \tau_j = \{(o_{j,t}, a_{j,t})\}_{t=1}^{T_j}, \tag{1}$$

where T_j is the length of trajectory τ_j, and $o_{j,t}, a_{j,t}$ are the observation and action, respectively.

Assume the BC policy π_{BC} performs sub-optimally at test time (i.e. failing to achieve the success condition), because the amount of training data or number of training epochs were insufficient, or due to distribution shifts induced by changes in environment conditions such as lighting variations, the presence of visual distractors, modified robot hardware or controllers, or other changes to environment dynamics.

Our objective is to improve performance on task $\mathcal{M}_{\mathrm{test}}$ beyond the original BC policy's performance. To achieve this, we assume access to the test environment for online interactions and a sparse reward signal. For this online phase, we formulate the task as a Markov Decision Process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \rho_0, \gamma\}$, where $\mathcal{S} \in \mathbb{R}^n$ and $\mathcal{A} \in \mathbb{R}^m$ represent states and actions, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, ρ_0 is the probability distribution over initial states, and $\gamma \in [0, 1)$ is a discount factor.

A. Background

Our proposed approach will involve estimating a Q-function, or the expected return from taking an action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$, then following a policy π thereafter:

$$Q_\pi(s, a) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a\right]. \tag{2}$$

The value function for policies is defined as the expectation over Q-values:

$$V(s) = \mathbb{E}_{a \sim \pi}\left[Q(s, a)\right]. \tag{3}$$

Although $Q_\pi(s, a)$ is a standard action-value function, it can also be interpreted as defining an energy-based model over actions. In particular, treating $E(s, a) = -Q(s, a)$ as an energy yields a Boltzmann policy

$$\pi(a \mid s) \propto \exp\left(\frac{1}{\alpha} Q(s, a)\right), \tag{4}$$

where α is a temperature parameter.

IV. Q2RL: Q-Estimation and Q-Gating from Behavior Cloning for Reinforcement Learning

In this section, we present *Q2RL*, an algorithm for estimating Q-values from a BC policy to guide online reinforcement learning. *Q2RL* consists of two phases: Q-Estimation (Sec. IV-A) and Q-Gating (Sec. IV-B). See Alg. 1 for a high-level description and Fig. 1 for a visual overview.

Algorithm 1 Q2RL

Given: behavior cloning policy π_{BC}

Roll out π_{BC} for N environment steps
Estimate $\hat{Q}_{\mathrm{BC}}$ using Eq. 6 ▷ Q-Estimation
Initialize Q_{RL} with $\hat{Q}_{\mathrm{BC}}$, Initialize π_{RL}

for $t = 0 \dots T$ environment steps **do**
  $a_{\mathrm{BC}} \sim \pi_{\mathrm{BC}}(s_t)$
  $a_{\mathrm{RL}} \sim \pi_{\mathrm{RL}}(s_t)$
  Select a_t using Eq. 10 ▷ Q Gating

  Observe next state s_{t+1} and reward r_t
  Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{B}$
  Sample a minibatch $\{(s, a, r, s')\} \in \mathcal{B}$
  Update Q_{RL}, π_{RL} using off-policy RL
end for

A. Q-Estimation from a Behavior Cloning Policy

In the first phase of our method, we estimate the Q-function of a pretrained BC policy π_{BC}. We first derive an analytic formula for the Q-function in terms of the value function, policy likelihoods, and entropy. We assume the action distribution of the human demonstrations $\mathcal{D}$ used to train π_{BC} is approximately a Boltzmann distribution, which has been widely used to model human actions [17, 20, 24, 4]. Assuming the BC policy's action distribution $\pi_{\mathrm{BC}}(a \mid s)$ can be expressed as a Boltzmann distribution with respect to Q_{BC} gives:

$$\pi_{\mathrm{BC}}(a \mid s) \approx \frac{\exp\left(Q_{\mathrm{BC}}(s,a)/\alpha\right)}{\int \exp\left(Q_{\mathrm{BC}}(s,a')/\alpha\right) da'}. \tag{5}$$

Using Eq. 3 and Eq. 5, we derive the equation for the Q-function Q_{BC} using the value function, log probability of the action, and entropy:

$$\hat{Q}_{\mathrm{BC}} = V_{\mathrm{BC}}(s) + \alpha \log \pi_{\mathrm{BC}}(a \mid s) + \alpha \mathcal{H}[\pi_{\mathrm{BC}}(\cdot \mid s)]. \tag{6}$$

Appendix A provides a complete derivation for Eq. 6.

Next, we detail how each term in Eq. 6 is computed or approximated. First, we must estimate the value function V_{BC}. We collect rollouts of the BC policy for a few initial online interaction episodes $\{\tau_i\}_{i=1}^{N}$ and calculate the Monte Carlo returns under the online reward function:

$$\hat{V}_{BC}(s_t^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=t}^{\tau_i} \gamma^k r_{t+k}^{(i)}, \text{ for } t \in \{1, \ldots, \tau_i\}. \tag{7}$$

These Monte Carlo returns are used to train a value estimator.

The remaining terms of Eq. 6 are derived from the BC policy π_{BC}. Our method is agnostic to the BC policy class, as long as it provides the log probabilities of actions and entropy. We provide the remaining components of Eq. 6 using Gaussian policies as an illustrative case. The log probability of the actions $\log \pi(a \mid s)$ for Gaussian policies has a closed-form expression:

$$\log \pi(a \mid s) = \frac{1}{2} \sum_{i=1}^{d} \left[\frac{(a_i - \mu_i(s))^2}{\sigma_i^2(s)} + \log(2\pi\sigma_i^2(s)) \right], \tag{8}$$

where (μ, σ) is Gaussian distribution output by the policy. Finally, the entropy $\mathcal{H}[\pi(\cdot \mid s)]$ for Gaussian policies also has a closed-form expression:

$$\mathcal{H}[\pi(\cdot \mid s)] = \frac{1}{2} \log\left(2\pi e\, \sigma^2(s)\right). \tag{9}$$

See Appendix A for a similar treatment for Gaussian Mixture Models (GMMs).

B. Q-Gating for Online Reinforcement Learning

We now describe Q-Gating, our approach to using $\hat{Q}_{\mathrm{BC}}$ estimated from Eq. 6 to guide online reinforcement learning. We first initialize two separate Q-functions, both from $\hat{Q}_{\mathrm{BC}}$: $\hat{Q}_{\mathrm{BC}}$ and Q_{RL}. $\hat{Q}_{\mathrm{BC}}$ is kept frozen to serve as a stable reference for the BC policy. Q_{RL} initialization involves an initial supervised training period to match $\hat{Q}_{\mathrm{BC}}$ on the same BC rollouts used for Q-Estimation. Q_{RL} then serves as the critic for the RL policy, and is updated through online interaction. During online training, we sample both the BC policy and RL policy for actions a_{BC} and a_{RL}, respectively. We evaluate a_{BC} using $\hat{Q}_{\mathrm{BC}}$ and a_{RL} using Q_{RL} to get their estimated Q-values. The action corresponding to the greater of these two values is executed:

$$a = \begin{cases} a_{\mathrm{BC}} \sim \pi_{\mathrm{BC}}(s), & \hat{Q}_{BC}(s, a_{\mathrm{BC}}) > Q_{RL}(s, a_{\mathrm{RL}}) \\ a_{\mathrm{RL}} \sim \pi_{\mathrm{RL}}(s_t), & \text{otherwise.} \end{cases} \tag{10}$$

We propose this Q-Gating mechanism to preserve good BC action proposals while still enabling policy improvement through RL. $\hat{Q}_{\mathrm{BC}}$ represents the (estimated) value of taking an action in the current state, then following the BC policy thereafter. We evaluate BC actions through a frozen $\hat{Q}_{\mathrm{BC}}$, so that BC actions with high value under $\hat{Q}_{\mathrm{BC}}$ are likely to be executed. On the other hand, Q_{RL} is only initialized as $\hat{Q}_{\mathrm{BC}}$, and trains on samples with either BC or RL actions. Q_{RL} therefore improves its estimate of both BC and RL actions, enabling the reinforcement learning policy to surpass BC performance through online exploration and interaction. Note that our approach does not require access to the training data for π_{BC}; we rely on our Q-estimate $\hat{Q}_{\mathrm{BC}}$ to bootstrap online reinforcement learning.

C. Implementation Details

We assume a temperature of $\alpha = 1$ for the BC policy's Boltzmann parameterization in Eq. 6. *Q2RL* can be applied to any behavior cloning policy that provides log probabilities of actions and entropy; in this work, we demonstrate the approach using Gaussian policies and GMMs. Similarly, though *Q2RL* is agnostic to the choice of off-policy RL algorithm, we use SAC optimized for real-world learning [11, 39].

To prevent the RL policy from deviating too far from the BC state-action distribution, we also weight the RL actor loss with an auxiliary BC loss [29]. While this regularization term can occasionally prevent the policy from reaching perfect success rates, our experiments in the following section show that training with BC regularization still outperforms baselines and plays a crucial role in producing safe, smooth behaviors on real robotic systems.

V. Experiments

Our experiments are designed to answer two key questions: (i) does *Q2RL* improve performance beyond the BC policy faster and better than baseline approaches? (ii) Does *Q2RL* use high-value actions from the BC policy during online RL? To provide a comprehensive evaluation, we conduct experiments across multiple simulation environments using BC policies parameterized by Gaussian and GMM-RNN architectures, and compare against offline-to-online RL and BC-to-RL baselines. Our evaluation spans both state-based and image-based observation settings. Finally, we validate the practicality of our approach through real-world RL experiments, assessing whether *Q2RL* can support online learning on physical hardware.

Fig. 2: Tasks from D4RL (`Kitchen-Complete`, `Adroit-Pen`, `Adroit-Door`) and robomimic (`Lift`, `Can`, `Square`).

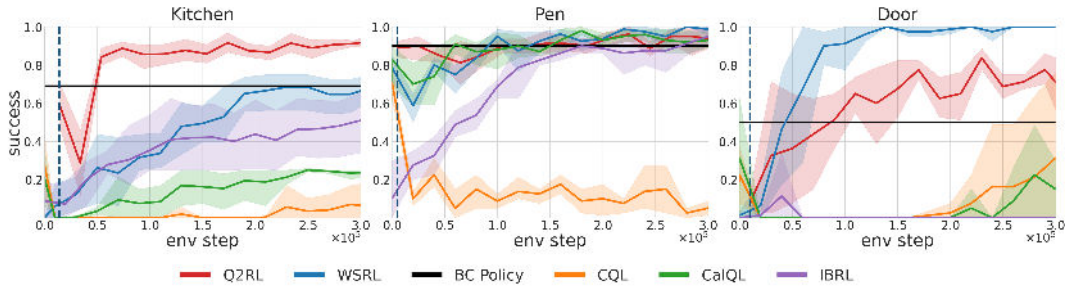


Fig. 3: Results on D4RL. Plots begin when a method starts online interaction. The dotted blue line signifies the end of the Q-Estimation phase and the start of online RL with Q-Gating for our method. Results are reported over 20 trajectories and 5 seeds, with 95% confidence intervals.

A. Simulation Environments

We test *Q2RL* on tasks from the D4RL [8] and robomimic [26] benchmarks. All tasks are visualized in Fig 2.

1) ***D4RL****:* We run experiments on two Adroit manipulation tasks: *Pen* and *Door*. The Pen task involves dexterous in-hand manipulation, requiring the robotic hand to reorient a pen to a target configuration, while Door consists of a standard door-opening task. In addition, we evaluate performance on the *Kitchen* environment, which features a Franka robot interacting with multiple objects to achieve a multi-stage goal configuration. For these experiments, we use the offline Kitchen-Complete dataset, which contains 19 successful demonstrations which are representative of standard robot learning setups where we only have access to a few successful trajectories. We conduct all D4RL experiments without access to offline training data during the online RL phase.

2) ***robomimic****:* We run experiments on three widely used robomimic tasks, *Lift*, *Can*, and *Square*, all featuring a Franka robotic arm. In the Lift task, the robot must lift a cube from a randomly sampled position on the table. The Can task requires the robot to grasp a can from one side of the table and place it into the correct bin among four target bins located on the opposite side. The Square task involves picking up a square nut from the table and inserting it onto a square peg. For robomimic tasks, we test both with and without access to offline training data during the online RL phase.

B. Baselines

We compare *Q2RL* with the following baselines:

1) ***CQL****:* Conservative Q-Learning [16] is an offline RL method that trains exclusively on an offline dataset by prioritizing in-distribution actions while penalizing out-of-distribution actions. This induces a pessimistic bias in the learned Q-function, which often requires a recalibration phase during online interaction.

2) ***CalQL****:* Calibrated Q-Learning [27] addresses the pessimism of Q-value estimates in CQL by calibrating the critic loss with respect to the value of a reference policy. However, despite this calibration, transitioning to online training can still lead to unlearning of its offline pretraining.

3) ***WSRL****:* Warm-Start RL [39] performs for offline-to-online RL without explicit data retention. To mitigate unlearning during the transition from offline to online training, WSRL introduces a warm-up phase in which the online replay buffer is seeded with rollouts generated by the offline pretrained policy. However, a key limitation of WSRL is its dependence on the quality of the offline pretraining data.

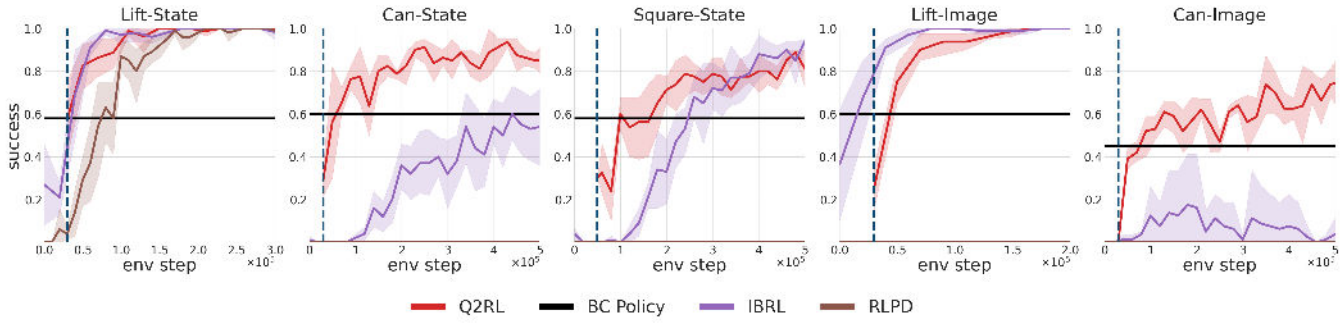
4) ***RLPD****:* RL with Prior Demonstrations [2] incorporates offline data directly into online learning by uniformly sampling from both offline and online replay buffers during actor and critic updates.

5) ***IBRL****:* Imitation Bootstrapped RL [12] uses a pre-trained BC policy to bootstrap Q-value estimates and action proposals for RL. IBRL randomly initializes the Q-function and uses demonstration data to seed the online replay buffer.

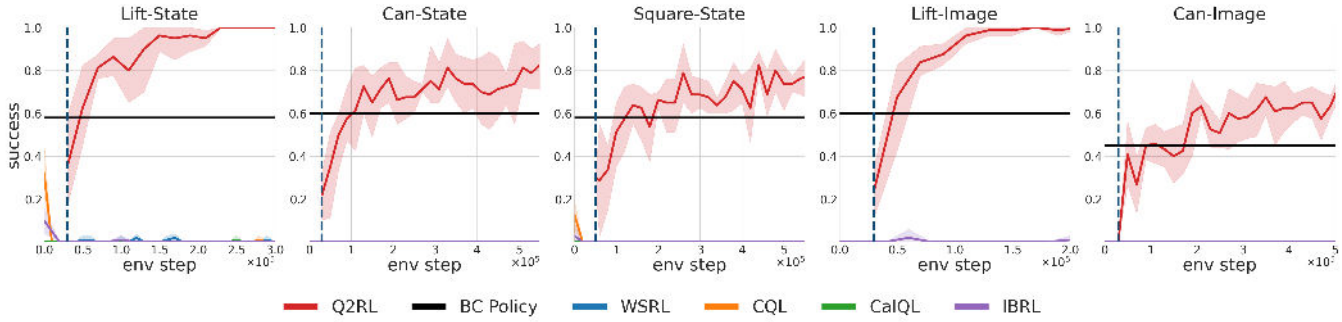
Additional details on simulation experiments can be found in Appendix B.

C. Results on D4RL

We present the results for Kitchen, Door and Pen tasks from D4RL in Fig 3. Offline RL methods perform poorly on the **Kitchen** task because the offline Kitchen-Complete dataset only contains successful demonstrations. Even with online RL after offline pre-training, these methods fail to improve over BC for the Kitchen task. *Q2RL*, on the other hand, starts



(a) Results with data seeded in the online replay buffer.



(b) Results without data seeded in the online replay buffer.

Fig. 4: Robomimic Results. The plot for each method begins at the start of online interaction. The dotted blue line signifies the end of the Q-Estimation phase and the start of online RL with Q-Gating for our method. Results are reported over 20 trajectories and 5 seeds, with 95% confidence intervals.

with better initial performance than the baselines and improves beyond the BC policy, which we attribute to Q-Estimation and online RL with Q-Gating.

In the **Pen** task, the offline data is able to provide sufficient pretraining for both BC and offline RL methods. We observe that *Q2RL* starts with near-optimal performance and maintains it throughout online learning. WSRL and CalQL are also able to reach high performance with online interaction on Pen, whereas CQL experiences a drop in performance, likely due to its conservative Q-value estimates. IBRL is also eventually able to reach high performance.

For the **Door** task, WSRL converges to a higher success rate than the baselines and *Q2RL*. However, from the rollout videos we observe that the behavior of the WSRL policy diverged from the original human demonstrations, and that it has learned to exploit the simulator through online RL, resulting in actions that would not be feasible in the real world. (Fig. 15, Appendix C). *Q2RL* uses Q-Estimation and Q-Gating to remain close to the BC policy, resulting in actions that can be realistically executed in the real world.

D. Robomimic Results

Next, we evaluate *Q2RL* on both state-based and image-based robomimic tasks. We use two cameras for image-based tasks, the agentview of the workspace and wrist camera. We implement *Q2RL* and all baselines with a learnable CNN encoder for the images to keep the implementation standard and comparisons fair. As these tasks are more challenging than the D4RL benchmarks, we consider two experimental settings: one in which training data is available in the online replay buffer, and another in which the policy does not have access to any offline training data during online learning.

1) Access to BC Training Data: Results for the robomimic tasks with access to training data during online learning are shown in Fig. 4a. When training data is available in the online replay buffer, IBRL performs competitively. IBRL randomly initializes the critic for off-policy RL and has no mechanism to preserve the initial performance of the BC policy, causing online interaction to begin with limited or no meaningful performance. In contrast, *Q2RL* begins online interaction with non-trivial initial performance rather than starting from near-zero success, which is particularly important for real-world deployment where unsafe or unproductive exploration is costly. Moreover, by restricting learning to targeted policy improvements, *Q2RL* converges to high-performing policies more rapidly. We also observe that for Can-Image, a harder high-dimensional task, IBRL is unable to reach base policy performance. We note that in the original IBRL paper, IBRL is able to converge for the Can-Image task, but we hypothesize that this is due to other optimizations used, such as specialized ViT encoders. *Q2RL* on the other hand is able to outperform the base policy with only the standard set of encoders. RLPD succeeds only on simpler tasks such as Lift and struggles to converge on the more challenging Can and Square tasks.

2) No Access to BC Training Data: BC policies are often provided by third parties without the data or infrastructure used to train them. Due to such occurrences, we evaluated our approach in cases where pre-training data was not accessible. Results for the robomimic tasks without access to training data during online learning are shown in Fig. 4b. Offline RL and offline-to-online RL methods are unable to learn for harder robotic tasks with limited offline datasets. Without access to data, IBRL is also unable to learn any meaningful performance. Early in training, the randomly initialized RL policy can propose arbitrary actions that may receive spuriously high Q-values from an untrained critic. To mitigate this issue, IBRL relies on seeding the replay buffer with training data, so that the critic learns to assign higher value to BC action proposals than to random RL actions. In the absence of a seeded replay buffer, IBRL loses this mechanism to stabilize initial Q-values, leading to unreliable action selection. *Q2RL* is able to continually improve performance even in the absence of training data. We attribute *Q2RL*'s strong performance without seeding to the effectiveness of the proposed Q-Estimation step. Q-Estimation computes Q_{BC} using BC policy rollouts *only*; it does *not* require expert demonstrations, nor a diverse offline dataset. *Q2RL* therefore has broad applicability in both limited access, memory-constrained settings where expert data is unavailable, as well as in unconstrained settings where expert data can seed online RL. We also conduct a thorough evaluation for seeding the replay buffer with different amounts of data in Appendix C.

E. Analysis of BC Actions Used During Online Training

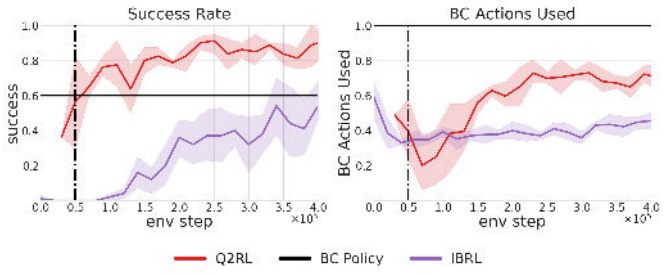
Next, we investigate the relationship between success rate and the ratio of BC vs. RL actions used during online training. Fig. 5 shows the success rate and the fraction of BC actions selected by *Q2RL* vs. IBRL on Can-State and Square-State tasks. The vertical dotted line serves as a visual reference indicating when *Q2RL*'s online performance approximately matches the original BC policy performance.

We observe that *Q2RL* rapidly recovers the original BC performance within only a few online interaction steps, demonstrating that it identified and exploited high-value actions of the BC policy. This performance is achieved while using BC actions less than half of the time, indicating that the RL policy also learned to produce effective actions. With continued training, as Q_{RL} estimates improve, Q-Gating more reliably identifies states where each policy has strong performance, so that Q2RL relies on BC for simple motions and reserves RL for difficult, contact-rich segments.

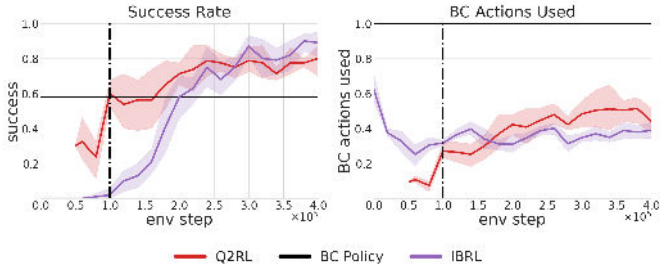
In contrast, IBRL selects a similar fraction of BC actions, but requires substantially more interaction to reach the same performance. Further, the ratio of BC actions used by IBRL during training stays relatively flat, indicating that it trades off between BC and RL actions less efficiently than *Q2RL*.

F. Ablations

We ablate Q-Initialization and Q-Gating in Fig. 6. Note that Q-Initialization, or initializing Q_{RL} with $\hat{Q}_{BC}$, is distinct



(a) Can-State



(b) Square-State

Fig. 5: BC vs. RL Actions Used During Online Training. *Q2RL* optimizes BC vs. RL action selection via Q-Gating, and achieves higher success rate earlier than IBRL.

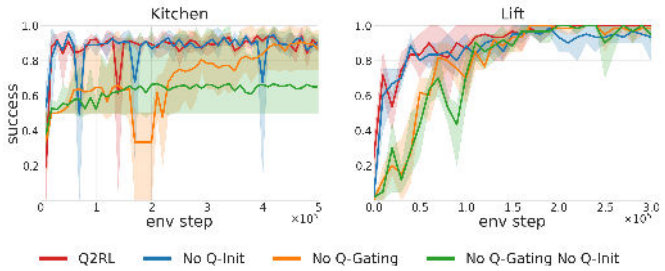


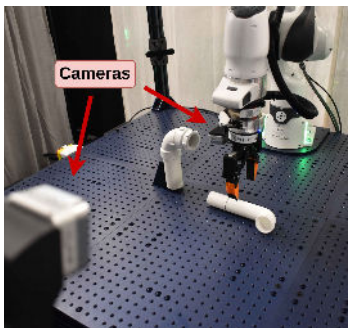
Fig. 6: Q-Initialization and Q-Gating Ablations.

from Q-Estimation, which is the algorithm for estimating $\hat{Q}_{BC}$ (Sec. IV-A). In the No Q-Gating condition, BC and RL action proposals are evaluated using the same Q-function during online RL. We see that Q-Gating is critical for achieving high performance. When Q-Gating is enabled, Q-Initialization yields performance comparable to a randomly initialized critic; however, we retain this initialization because it provides a safer starting distribution that remains closer to the BC policy, which is preferable for on-robot RL. See Appendix C for additional ablation studies.

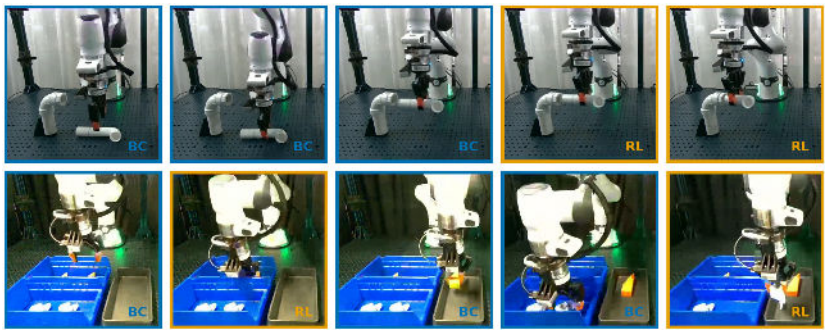
G. Real World RL Experiments

1) Experiment Setup: We run all experiments on a 7-DOF Franka Panda Robot with a Robotiq 2F-85 gripper. For all tasks, the robot receives the end effector pose, gripper width, and 84×84 RGB images from a workspace camera and wrist camera (see Fig. 7a). The robot outputs delta pose actions executed by a Cartesian impedance controller. We evaluate on the following tasks (see Fig. 1):

Peg Insertion. This insertion task from FMB [22] has been used in recent on-robot RL methods [21, 39]. Success occurs when the peg is fully inserted into the board.



(a) On-Robot RL Setup.



(b) Trained *Q2RL* policies for pipe assembly and kitting tasks.

Fig. 7: Real World Experiments. (a) A Franka Panda arm with a Robotiq 2F-85 gripper, with RGB images from wrist and workspace Intel RealSense D405s. (b) We analyze usage of BC (**blue**) vs. RL (**yellow**) actions in representative trajectories executed by trained *Q2RL* policies. For pipe assembly, *Q2RL* relies on BC for grasping and initial alignment, then switches to RL for high-precision, contact-rich insertion. For kitting, the BC policy was trained in a task setting with only one object present in each bin; *Q2RL* learns to complete the task with multiple objects in each bin. In the modified setting, *Q2RL* uses BC actions for behaviors common to both task settings (e.g. moving between bins), and uses RL for part grasping and placement.

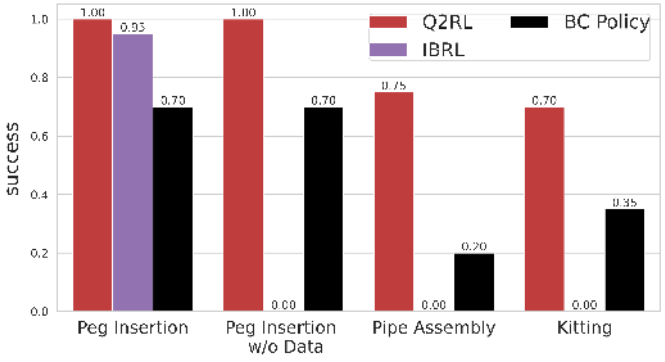


Fig. 8: On-Robot RL Results. Task success is reported over 20 trials per method. *Q2RL* significantly improves BC policy performance on every task. IBRL fails when (1) demonstrations are not seeded in the replay buffer (w/o Data), and (2) when tasks involve large action spaces and long horizon manipulation (even with seeding).

Plumbing Pipe Assembly. This task involves both grasping and low tolerance insertion. Insertion requires rotating the gripper to align the pipe with the fixture. Success occurs when the pipe is fully inserted into the fixture.

Kitting with Task Distribution Shift. In this task, two parts must be retrieved from their respective bins and placed in a kitting tray. As the longest horizon task, this task requires multiple picks and places. Success occurs when the robot has placed both parts in the kitting tray. The BC policy for this task was only trained on demonstrations with a single object per bin. To evaluate robustness to task distribution shift, the test task setting has two objects per bin, placed in new locations.

GMM-RNN BC policies were trained for each task on 50 to 100 demos collected by a human operator using a 3D Connexion Spacemouse. We compare *Q2RL* with IBRL for offline-to-online improvement with the pre-trained BC policies. During online learning, a termination signal and sparse reward signal were provided by a human operator when the task's success conditions were met. Environment resets were also handled by a human operator. Additional details on the real experiment setup can be found in Appendix D.

2) Real World Results: For the Kitting task, all results are for the modified setting (two parts in each bin instead of one). The best checkpoint after up to 2.5 hours of online training is used for each method, representing no more than 170k gradient steps and 80k environment steps (much fewer than in simulation). From Fig. 8, ***Q2RL* significantly improves performance compared to the original BC policy, achieving a 1.4x improvement and 100% success on the Peg Insertion task, 3.75x improvement on Pipe Assembly, and 2x improvement on Modified Kitting.** While IBRL performed on par with *Q2RL* for the Peg Insertion task, it achieved zero success when demonstration samples were not seeded in the online replay buffer. Even with data seeding, when tasks involved large action spaces or were long horizon, IBRL had zero success, failing to recover BC performance. We attribute IBRL's poor performance to its use of a single, randomly initialized Q-function to select BC vs. RL actions, which hinders it from accurately rating BC actions, especially for long horizon, sparse reward tasks. In contrast, we initialize Q_{RL} from $\hat{Q}_{\mathrm{BC}}$, and our proposed Q-Gating procedure uses both a frozen $\hat{Q}_{\mathrm{BC}}$ and trainable Q_{RL} to score respective actions more accurately.

Safety. During the course of training IBRL for Peg Insertion, we noted two safety violations, in which the Franka exerted excessive force against the board and caused the robot to fault. These safety violations occurred despite careful tuning of controller gains during preliminary testing that was subsequently kept fixed for all reported runs. *Q2RL* experienced no such safety violations, and we qualitatively observed that it produced relatively smoother and safer actions early in training: we attribute this behavior to a combination of the auxiliary BC loss, Q-Gating, and Q_{RL} initialization with $\hat{Q}_{\mathrm{BC}}$.

Qualitative Results. Fig. 7b shows frames from representative rollouts of trained *Q2RL* policies, annotated to indicate BC vs. RL action selection. We observe that *Q2RL* selects BC actions to perform behaviors supported by the BC policy's training data, e.g. initial pipe grasping and alignment for the Pipe Assembly task, before switching to RL actions for high-precision, contact-rich insertion. With Modified Kitting, BC actions were selected for moving between bins, with RL actions handling parts of the task that had shifted from their original setting, e.g. grasping parts from new locations. Other cases of switching between BC and RL actions corresponded to recovery behaviors, such as insertion re-alignment and re-grasping. Not all rollouts or trained *Q2RL* policies exhibit these behaviors exactly as described here, but we generally find meaningful switching between BC and RL actions. Please refer to the website for videos of annotated rollouts, and Appendix D for additional details on results.

VI. Conclusion

We presented *Q2RL*, an algorithm that improves the performance of a behavior cloning policy through online reinforcement learning. *Q2RL* estimates a Q-function from the BC policy, then uses it to initialize and guide online RL. Our approach does not require access to the BC policy's original training data, and is compatible with any BC policy class that provides action likelihoods and entropy. In on-robot reinforcement learning experiments, *Q2RL* successfully improves performance on high precision, contact-rich manipulation tasks within a few hours of online interaction.

One limitation of our work is that *Q2RL* requires the BC policy to provide action likelihoods and entropy; as future work, we aim to extend *Q2RL* to other policy classes that do not provide these by default, such as diffusion and flow matching policies.

Acknowledgments

This work was supported in part by ONR grant N00014-22-1-2592 and ONR grant "Long-Term Autonomy for Ground and Aquatic Robotics" N00014-24-1-2784. The authors thank Justin Rojas, Will Heitman, and Steve Proulx for assistance with real robot experiments and hardware, Victoria Coleman and Colin Kohler for real robot infrastructure support, and Eric Rosen for helpful paper feedback.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International conference on machine learning*, pages 104–114. PMLR, 2020.

[2] 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.

[3] Ondrej Biza, Thomas Weng, Lingfeng Sun, Karl Schmeckpeper, Tarik Kelestemur, Yecheng Jason Ma, Robert Platt, Jan-Willem van de Meent, and Lawson L.S. Wong. On-robot reinforcement learning with goal-contrastive rewards. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4797–4805, 2025. doi: 10.1109/ICRA55743.2025.11128466.

[4] Andreea Bobu, Andrea Bajcsy, Jaime F Fisac, and Anca D Dragan. Learning under misspecified objective spaces. In *Conference on Robot Learning*, pages 796–805. PMLR, 2018.

[5] Lawrence Yunliang Chen, Simeon Adebola, and Ken Goldberg. Berkeley UR5 demonstration dataset. https://sites.google.com/view/berkeley-ur5/home.

[6] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[7] Lakshita Dodeja, Karl Schmeckpeper, Shivam Vats, Thomas Weng, Mingxi Jia, George Konidaris, and Stefanie Tellex. Accelerating residual reinforcement learning with uncertainty estimation. *arXiv preprint arXiv:2506.17564*, 2025.

[8] 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.

[9] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

[10] Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. *Advances in Neural Information Processing Systems*, 34:4028–4039, 2021.

[11] 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.

[12] Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation bootstrapped reinforcement learning. *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi: 10.15607/RSS.2024.XX.056.

[13] Sunshine Jiang, Xiaolin Fang, Nicholas Roy, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Siddharth Ancha. Streaming flow policy: Simplifying diffusion / flow-matching policies by treating action trajectories as flow trajectories. *arXiv preprint arXiv:2505.21851*, 2025.

[14] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[15] Michael Kelly, Chelsea Sidrane, Katherine Rose Driggs-Campbell, and Mykel J. Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In

International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019, pages 8077–8083. IEEE, 2019.

[16] 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.

[17] Cassidy Laidlaw and Anca Dragan. The boltzmann policy distribution: Accounting for systematic suboptimality in human models. In *ICLR*, 2022.

[18] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

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

[20] R Duncan Luce et al. *Individual choice behavior*, volume 4. Wiley New York, 1959.

[21] Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16961–16969. IEEE, 2024.

[22] Jianlan Luo, Charles Xu, Fangchen Liu, Liam Tan, Zipeng Lin, Jeffrey Wu, Pieter Abbeel, and Sergey Levine. Fmb: a functional manipulation benchmark for generalizable robotic learning. *The International Journal of Robotics Research*, 44(4):592–606, 2025.

[23] Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10 (105):eads5033, 2025.

[24] Jim Mainprice, Rafi Hayne, and Dmitry Berenson. Predicting human reaching motion in collaborative tasks using inverse optimal control and iterative re-planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 885–892, 2015. doi: 10.1109/ICRA.2015.7139282.

[25] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. *arXiv preprint arXiv:2003.06085*, 2020.

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

[27] 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*, 36:62244–62269, 2023.

[28] Soroush Nasiriany, Tian Gao, Ajay Mandlekar, and Yuke Zhu. Learning and retrieval from prior data for skill-based imitation learning. *arXiv preprint arXiv:2210.11435*, 2022.

[29] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[30] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

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

[32] Ziyad Sheebaelhamd, Michael Tschannen, Michael Muehlebach, and Claire Vernade. Quantization-free autoregressive action transformer. *arXiv preprint arXiv:2503.14259*, 2025.

[33] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[34] You Liang Tan. Agentlace, framework for distributed agent policy, May 2024. URL https://github.com/youliangtan/agentlace.

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

[36] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International conference on learning representations*, 2021.

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

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

[39] Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HN0CYZbAPw.

APPENDIX A
DERIVATIONS

A. Derivation for Estimating Q-values for BC: Q_{BC}

Assuming π_{BC} is a Boltzmann policy that is defined by the value function Q as follows,

$$\pi_{BC}(a \mid s) = \exp \frac{1}{\alpha} Q(s,a)/Z(s), \tag{11}$$

$$\text{where } Z(s) = \int_{\mathcal{A}} \exp \frac{1}{\alpha} Q(s,a')da', \ a \in \mathcal{A} \tag{12}$$

where α is a temperature parameter, then we can re-arrange the above equations to estimate $Q(s,a)$ using only the state-value and the policy distribution.

Lemma A.1. *Given a Boltzmann policy π_{BC} and assuming $Z(s)$ is finite for all $s \in \mathcal{S}$, we can express the Boltzmann value function $Q(s,a)$ of π_{BC} as the following:*

$$Q(s,a) = V(s) + \alpha \log \pi_{BC}(a \mid s) + \alpha \mathcal{H}[\pi_{BC}(\cdot \mid s)], \tag{13}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

Proof: Firstly, we derive an equation for the partition function $\log Z$ using the policy entropy $\mathcal{H}$ and Eq. 11:

$$\begin{aligned}
\mathcal{H}[\pi_{BC}(\cdot \mid s)] &= -\mathbb{E}_{a \sim \pi_{BC}(\cdot|s)}[\log \pi_{BC}(a \mid s)] \\
&= -\mathbb{E}_{a \sim \pi_{BC}(\cdot|s)} \left[\frac{1}{\alpha} Q(s,a) - \log Z(s) \right] \\
&= -\frac{1}{\alpha} V(s) + \log Z(s).
\end{aligned}$$

Rearranging,

$$\log Z(s) = \frac{1}{\alpha} V(s) + \mathcal{H}[\pi_{BC}(\cdot \mid s)]$$

Secondly, we substitute this equation into the definition of our Boltzmann policy,

$$\begin{aligned}
\log \pi_{BC}(a \mid s) &= \frac{1}{\alpha} Q(s,a) - \log Z(s) \\
&= \frac{1}{\alpha} Q(s,a) - \frac{1}{\alpha} V(s) - \mathcal{H}[\pi_{BC}(\cdot \mid s)].
\end{aligned}$$

So

$$Q(s,a) = V(s) + \alpha \log \pi_{BC}(a \mid s) + \alpha \mathcal{H}[\pi_{BC}(\cdot \mid s)]$$

as desired. ∎

B. Derivations for Gaussian Mixture Models

In this section, we provide additional details on using *Q2RL* with Gaussian Mixture Model (GMM) based BC policies.

1) Action Likelihoods: The probability density function of a GMM with C components is defined as the weighted sum of the probability density of each of the mixture component π_i [3]:

$$\pi(a \mid s) = \sum_{i=1}^{N} c_i \pi_i(a \mid s), \tag{14}$$

where c_i is the weight of each component i ($c_i \geq 0, \sum_i c_i = 1$).

2) Entropy: For a GMM, $\mathcal{H}[\pi(\cdot \mid s)]$ is upper-bounded by the sum of expected component entropy and discrete entropy of mixture weights [3]:

$$\mathcal{H}[\pi(\cdot \mid s)] \leq \mathcal{H}[\pi(\cdot \mid s) \mid C] + \mathcal{H}(C) \tag{15}$$

where the expected component entropy can be defined as:

$$\mathcal{H}[\pi(\cdot \mid s) \mid C] = \sum_i c_i \mathcal{H}[\pi_i(\cdot \mid s)] \tag{16}$$

We use this upper bound of entropy for our GMM policy experiments.

APPENDIX B
DETAILS ON SIMULATION EXPERIMENTS

We define the BC policy hyperparameters in Table I along with the BC loss weights. Table II contains hyperparameters of the RL agent for *Q2RL*. We use the same RL hyperparameters for IBRL and WSRL as used in their respective papers.

TABLE I: BC Epochs and Auxiliary Loss Weights for RL

	BC Loss Weight	BC Epoch Used
Kitchen	0.3	200k
Pen	0.3	200k
Door	0.3	100k
Lift-State	0.3	350
Can-State	0.3	250
Square-State	0.2	1000
Lift-Image	0.2	20
Can-Image	0.2	15
Peg Insertion	0.1	600
Pipe Assembly	0.1	600
Kitting (Modified)	0.1	600

A. Simulation Hyperparameters

We use the same reward shaping from WSRL, namely a reward bias b at each timestep and scale factor a for nonzero rewards. The transformed reward is:

$$\tilde{r} = ar + b \tag{17}$$

In our experiments, the only nonzero environment reward r is the sparse reward given upon success ($r = 1$). For D4RL tasks, we use the reward scale and bias values from WSRL. For all other tasks (including Kitchen), bias $b = -1$, a negative per-step penalty that encourages the agent to complete the task faster. Scale factor a helps with TD error propagation and its value in our experiments differs between simulation and the real world (see Table II).

We use the same Soft Actor-Critic implementation as WSRL [11] as our reinforcement learning algorithm. We use DrQ image augmentation for image-based environments as implemented in SERL [9, 4].

TABLE II: Reinforcement Learning Agent Hyperparameters

	Simulation (State)	Simulation (Image)	Real-world
Batch Size	256 (Kitchen: 1024)	256	256
Hidden Dimensions (Actor and Critic)	[512, 512, 512]	[1024, 1024]	[1024, 1024]
Learning Rate (Actor and Critic)	3e-4	1e-4	1e-4
Reward Scale	5 (Adroit: 10, Kitchen: 4)	5	10
Num. Rollouts for Q-Estimation Phase	50 (D4RL), 100 (Robomimic)	100	See Table V
Reward Bias	-1 (Adroit: 5)		
Num. Q-Estimation Training Steps	20k (50k for Square)		
UTD Ratio	4		
Critic Ensemble Size	10		
Critic Subsample Size	2		
Discount Factor	0.99		
Soft Target Update Rate	0.005		
Layer Normalization	Yes		
Replay Buffer Size	2e6		

TABLE III: Simulation Results - State

Without Data	Kitchen			Pen			Door		
Method	100k	200k	300k	0	150k	300k	100k	200k	300k
BC Policy	0.69	0.69	0.69	**0.9**	**0.9**	0.9	0.5	0.5	0.5
WSRL	0.31 ± 0.09	0.65 ± 0.09	0.64 ± 0.08	0.78 ± 0.06	$\mathbf{0.92 \pm 0.05}$	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.91 \pm 0.11}$	$\mathbf{0.98 \pm 0.01}$	**1.0**
CQL	0.0	0.0	$.06 \pm 0.08$	0.7 ± 0.08	0.12 ± 0.03	0.05 ± 0.03	0.0	0.02 ± 0.03	0.31 ± 0.33
CalQL	0.07 ± 0.07	0.19 ± 0.07	0.23 ± 0.02	0.71 ± 0.11	$\mathbf{0.87 \pm 0.02}$	$\mathbf{0.93 \pm 0.03}$	0.0	0.0	0.15 ± 0.15
IBRL	0.35 ± 0.14	0.43 ± 0.18	0.50 ± 0.15	0.1 ± 0.15	0.82 ± 0.02	$\mathbf{0.95 \pm 0.05}$	0.0	0.0	0.0
Q2RL (Ours)	$\mathbf{0.85 \pm 0.06}$	$\mathbf{0.87 \pm 0.03}$	$\mathbf{0.91 \pm 0.01}$	$\mathbf{0.88 \pm 0.04}$	$\mathbf{0.91 \pm 0.05}$	$\mathbf{0.93 \pm 0.03}$	0.55 ± 0.15	0.73 ± 0.12	0.87 ± 0.08
With Data	Lift-State			Can-State			Square-State		
Method	100k	200k	300k	100k	300k	500k	100k	300k	500k
BC Policy	0.58	0.58	0.58	0.6	0.6	0.6	**0.58**	0.58	0.58
RLPD	0.87 ± 0.07	$\mathbf{0.98 \pm 0.02}$	$\mathbf{0.99 \pm 0.01}$	0.0	0.0	0.0	0.0	0.0	0.0
IBRL	$\mathbf{0.97 \pm 0.03}$	**1.0**	$\mathbf{0.98 \pm 0.02}$	0.02 ± 0.02	0.32 ± 0.16	0.54 ± 0.18	0.0	$\mathbf{0.72 \pm 0.11}$	$\mathbf{0.94 \pm 0.01}$
Q2RL (Ours)	0.88 ± 0.06	**1.0**	**1.0**	$\mathbf{0.76 \pm 0.05}$	$\mathbf{0.86 \pm 0.03}$	$\mathbf{0.85 \pm 0.03}$	$\mathbf{0.60 \pm 0.07}$	$\mathbf{0.78 \pm 0.06}$	0.81 ± 0.08
Without Data	Lift-State			Can-State			Square-State		
Method	100k	200k	300k	100k	300k	550k	100k	300k	550k
BC Policy	0.58	0.58	0.58	**0.6**	0.6	0.6	**0.58**	0.58	0.58
WSRL	0.01 ± 0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CQL	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CalQL	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
IBRL	0.01 ± 0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Q2RL (Ours)	$\mathbf{0.86 \pm 0.08}$	$\mathbf{0.96 \pm 0.05}$	**1.0**	$\mathbf{0.57 \pm 0.12}$	$\mathbf{0.75 \pm 0.13}$	$\mathbf{0.82 \pm 0.10}$	$\mathbf{0.51 \pm 0.15}$	$\mathbf{0.68 \pm 0.06}$	$\mathbf{0.76 \pm 0.06}$

B. Additional Details

We provide some additional details on our simulation experiments:

1) IBRL implementation: IBRL uses observation history $(t-2, t-1, t)$ for their RL policy. To keep comparisons fair, we implement our RL policy conditioned only on the current observation. Our results demonstrate that *Q2RL* performs well using a simple RL policy architecture that takes a single observation as input and outputs a single action.

2) Sampling Actions: The Q-gating step in *Q2RL* selects the action with the highest Q-value. Accordingly, in simulation experiments we evaluate RL actions using the mode of the policy distribution. In real-world settings, which are inherently more stochastic, we instead sample actions using the policy's distributional standard deviation. *Q2RL* demonstrates good performance under both action sampling strategies.

C. Details on Simulation Results

To support the plots in the main paper, we provide success rate numbers and confidence intervals for state-based simulation tasks in Table III and image-based tasks in Table IV. All the evaluations are done for 20 rollouts and 5 seeds. We do not evaluate offline RL-based methods in image-based environments, as they failed to successfully pretrain and already exhibited poor performance on the more challenging state-based tasks.

APPENDIX C
ADDITIONAL EXPERIMENTS AND ABLATIONS

A. Testing the Soft-Optimality Assumption

To test *Q2RL*'s sensitivity to the soft-optimality assumption, we provide experiments simulating other forms of sub-optimality. We use a *deterministic* π_{BC} plus ϵ noise sampled from Gaussian and uniform distributions. This non soft-

TABLE IV: Simulation Results - Image

With Data	Lift-Image		Can-Image		
Method	100k	200k	100k	300k	500k
BC Policy	0.6	0.6	0.45	0.45	0.45
RLPD	0.0	0.0	0.0	0.0	0.0
IBRL	**1.0**	**1.0**	0.12 ± 0.13	0.01 ± 0.01	0.03 ± 0.05
Q2RL (Ours)	0.93 ± 0.05	**1.0**	$\mathbf{0.53 \pm 0.07}$	$\mathbf{0.63 \pm 0.02}$	$\mathbf{0.73 \pm 0.06}$
Without Data	**Lift-Image**		**Can-Image**		
Method	100k	200k	100k	300k	550k
BC Policy	0.6	0.6	**0.45**	0.45	0.45
IBRL	0	0.01 ± 0.01	0.0	0.0	0.0
Q2RL (Ours)	$\mathbf{0.87 \pm 0.05}$	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.45 \pm 0.07}$	$\mathbf{0.57 \pm 0.10}$	$\mathbf{0.63 \pm 0.05}$

optimal π_{BC} is used for both Q-Estimation and online RL. Fig. 9 shows that such policies suffer an initial drop in success (performance at dotted line), but they recover during online RL, demonstrating the robustness of *Q2RL*.

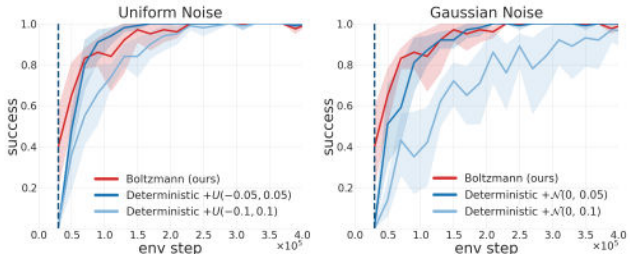


Fig. 9: Results with non soft-optimal BC policies for Lift-State on 20 trials.

B. Baseline: Residual Reinforcement Learning

We compare *Q2RL* with Policy Decorator [10], a state-of-the-art residual learning method. Residual policy learning is another approach to improving a pre-trained BC policy by providing corrective adjustments to the BC actions. Policy Decorator introduces two key enhancements to the Residual RL framework: (1) scaling the residual action added to the base policy, and (2) employing a uniform progressive exploration schedule for the residual policy. The results are shown in Fig. 10. We find that while Policy Decorator is able to recover and modestly improve upon the BC performance for the Can task, it requires significantly more interaction to recover the BC performance on more challenging Square task.

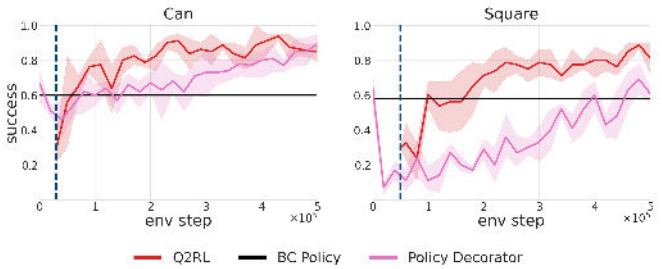


Fig. 10: Policy Decorator takes more online interaction to recover BC performance for challenging tasks like Square.

C. Ablation: Seeding the Online Replay Buffer

We analyze the effect of seeding the online replay buffer with different fractions of data for both *Q2RL* and IBRL (Fig. 11). *Q2RL* maintains strong performance regardless of whether offline training data is available in the replay buffer, corroborating our real-world experiments. In contrast, IBRL shows a strong dependence on replay buffer seeding.

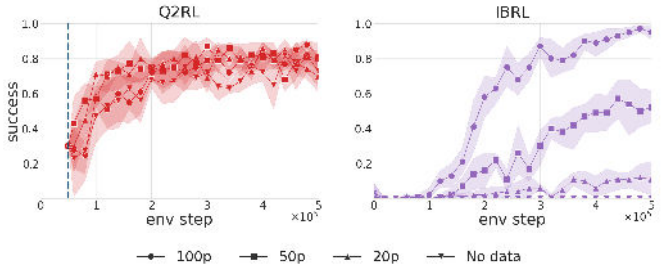


Fig. 11: Seeded Online Replay Buffer Ablation. *Q2RL* does not require seeding the online replay buffer with data.

D. Ablation: BC Auxiliary Loss

Q2RL incorporates an auxiliary behavior cloning (BC) loss to stabilize training, whereas IBRL relies on access to demonstration data. To ensure a fair comparison in settings without data access, we also evaluate IBRL with the same BC loss in Fig. 12. While the BC loss improves IBRL relative to the no-BC loss variant, it does not lead to meaningful performance gains compared to *Q2RL*.

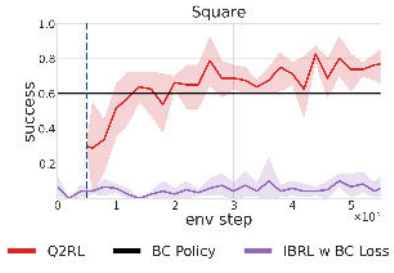


Fig. 12: IBRL is not able to match the performance of *Q2RL* even with the BC auxiliary loss in the absence of training data.

E. Ablation: Number of Rollouts used for Q-Estimation

We ablate the number of rollouts used during the Q-estimation phase in Fig. 13, using the Can-State task in the with data setting. We observe that *Q2RL* achieves competitive performance with as few as 25 rollouts. In our main simulation experiments, we use 100 rollouts to obtain more reliable Q-estimates and ensure stable performance.

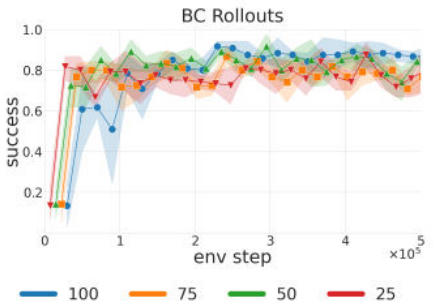


Fig. 13: *Q2RL* achieves competitive performance with as few as 25 rollouts.

F. Ablation: Initial BC Performance

Next, we evaluate whether *Q2RL* can improve BC policies with varying levels of initial performance. We use the same experimental setup as in Sec. C-E and test *Q2RL* with different BC checkpoints, with initial success rates ranging from 10% to 75%, as shown in Fig. 14. We observe that *Q2RL* consistently improves performance across all initializations.

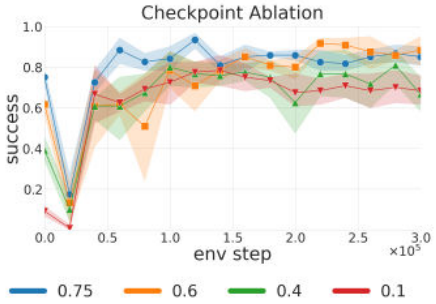


Fig. 14: *Q2RL* is able to improve a wide range of initial BC performance.

G. Qualitative Result: Door

While Fig. 3 shows that WSRL achieves higher success rates than *Q2RL* for the Adroit-Door task, qualitative rollouts (Fig. 15) reveal that *Q2RL* learns more realistic policies where the hand first grabs the door handle and then opens the door. On the other hand, the policies learned by WSRL are less feasible in the real world.

APPENDIX D
DETAILS ON REAL WORLD EXPERIMENTS

A. Hardware

We conducted real world, on-robot experiments on a table-top Franka FR3 arm. The arms is equipped with Robotiq 2F-85 gripper with 3D-printed, compliant "finray" fingertips. Two Realsense D405s provide RGB input, one providing the workspace view, and the other the wrist view. See Fig. 7a for an image of the robot and workspace. A force-torque sensor is mounted on the wrist but is not used. A 3D Connexion Spacemouse is used to collect teleoperated human demonstrations and to assist with environment resets.

B. Software

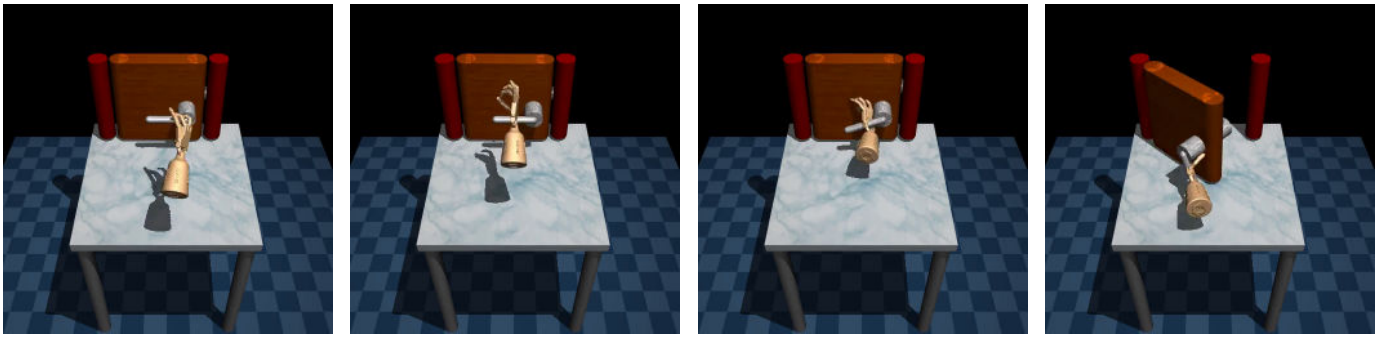
The inputs for all policies are the end effector pose in the robot base frame, and 84 x 84 RGB images (cropped and resized from 480 x 848) from each of the two cameras. For the FMB [5] peg insertion task, the gripper is disabled. The policy outputs delta end effector pose actions and gripper commands (if relevant). The delta end effector actions are scaled to metric range, clipped to be within safe limits, converted to the robot base frame, and executed by a Cartesian impedance controller. The stiffness and damping gains for the controller were set prior to starting experiments and the values were kept the same for all tasks. The policy maintains a 10 Hz action frequency.

Our on-robot RL system consists of an actor process and a learner process. The actor process collects transition samples by executing actions on the robot. The learner process performs gradient descent based on collected data. The actor and learner run as independent processes and communicate asynchronously via ZeroMQ [7]. This asynchronous approach allows both processes to run simultaneously, speeding up time to convergence [4, 8]. For Q2RL, the actor process loads the frozen BC policy and maintains a copy of the RL policy's weights. The learner process also maintains a copy of the RL weights for updating via learning. The actor asynchronously updates the learner's replay buffer every 30 actor samples, and the learner asynchronously updates the actor's network every 30 learner training steps. On average, our system executes around 13k RL actions and 44k RL learner steps per hour.

C. Task Descriptions

This section provides detailed descriptions for real world tasks. For all tasks, we reset the environment via motion planning to move to a fixed reset pose or a randomized reset pose. During reset, spacemouse teleoperation is sometimes used to move the arm into free-space before executing motion planning (e.g., when a pipe is partially inserted at the end of an episode). A human assists with resets and also provides the success signal based on the success conditions below. The success signal terminates the episode; otherwise, the episode is truncated upon reaching the max episode length. See Fig. 16 for key-frame images for each task and Tab. V for task configuration parameters.

Peg Insertion. In this task, the robot inserts a 3D-printed peg into a board. The peg tolerance for mating is 1-2 mm



WSRL rollout on Adroit-Door.

Q2RL rollout on Adroit-Door.

Fig. 15: *Q2RL* learns more realistic policies than WSRL, even though WSRL achieves higher success rate for Adroit-Door.

TABLE V: Real World Experiment Configurations

	Peg Insertion	Pipe Assembly	Kitting (Modified)
Initial Pose Distribution (m, rad)	±0.045, ±0.07875	fixed	fixed
Delta Action Range (m, rad)	±0.015, ±0.02625	±0.015, ±0.02625	±0.015, ±0.02625
Safety Limit XYZ (m)	[-0.02, 0.05], [-0.1, 0.02], [-0.13, 0.02]	[-0.2, 0.1], [-0.2, 0.2], [-0.4, 0.05]	[-0.2, 0.3], [-0.2, 0.2], [-0.18, 0.05]
Safety Limit RPY (rad)	[-5, 5], [-5, 5], [-20, 20]	[-5, 5], [-5, 5], [-20, 20]	[-5, 5], [-5, 5], [-20, 20]
Num. Demos for BC Training	50	100	50
Num. Rollouts for Q-Estimation Phase	100	100	30
Max Episode Length	200	200	500
Cartesian Stiffness (XYZRPY)	[1024, 1024, 1024, 100, 100, 100]		
Cartesian Damping (XYZRPY)	[64, 64, 64, 10, 10, 10]		
Policy Hz	10		

and the insertion length is 50 mm. We evaluate on this task using objects from the Functional Manipulation Assembly [5] benchmark to provide a comparison similar to recent work that report real-world RL experiments [4, 11]. However, note that differences in action space such as the size of the safety box, range of allowed end effector position and orientation, and other task parameters mean the results across works are not directly comparable. Hence we run baselines on our task configuration to provide a fair comparison.

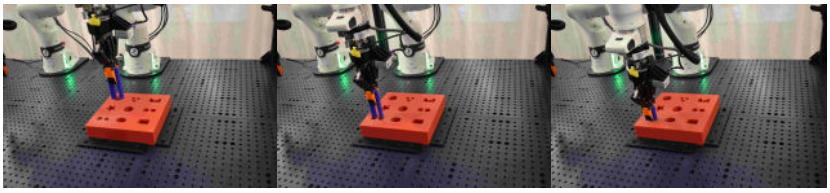
Starting Configuration: The board is fixed on the table for all episodes. The peg starts grasped by the gripper. Before the start of every episode, the gripper pose is randomized by sampling a random delta pose action and executing it three times.

Success Condition: The peg is fully inserted into the board.

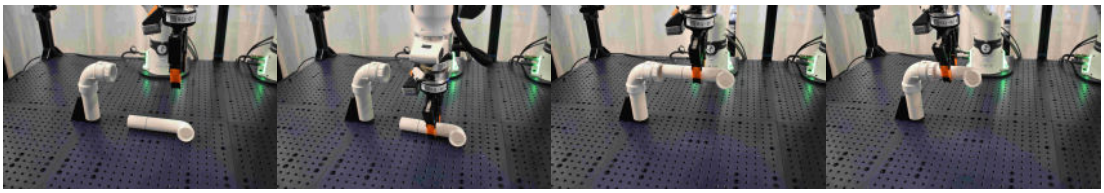
Pipe Assembly. In this task, the robot picks up a PVC pipe and inserts into a plumbing fixture. The tolerance for mating is 1-2 mm and the insertion length is 57 mm. Besides evaluating insertion with commercially available parts, this task differs from the prior Peg Insertion task in the following ways: (1) the PVC pipe starts on the table and must be grasped before attempting insertion; and (2) the fixture is not aligned with the initial pose of the pipe, so inserting the pipe requires gripper rotation, or else the pipe will jam.

Starting Configuration: The pipe, gripper, and fixture start in the same poses for this task. Despite having less initial variation in this task compared to Peg Insertion, we observe that this task has lower BC policy success rates as it requires grasping followed by insertion. The average number of steps required to achieve the task is double that of Peg Insertion.

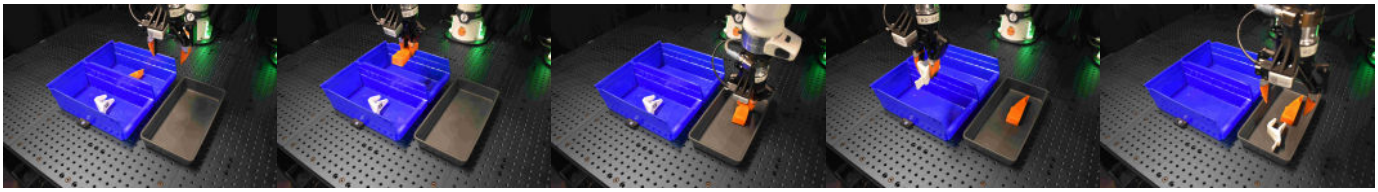
Success Condition: The pipe is fully inserted into the fixture. A black line is drawn on the pipe to help the human



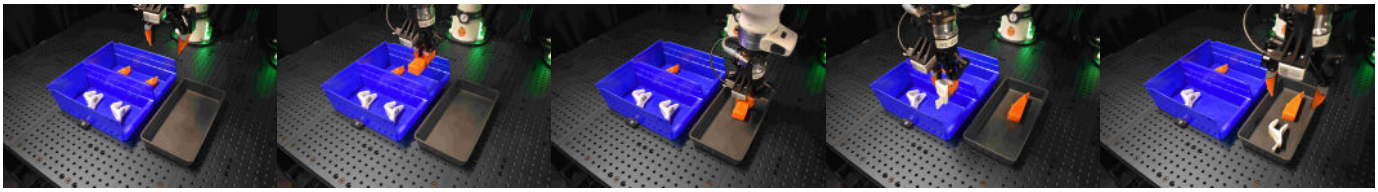
(a) Peg Insert (start) (b) Align (c) Insert (done)



(d) Pipe Assembly (start) (e) Grasp (f) Align (g) Insert (done)



(h) Kitting-Original (start) (i) Pick #1 (j) Place #1 (k) Pick #2 (l) Place #2 (done)



(m) Kitting-Modified (start) (n) Pick #1 (o) Place #1 (p) Pick #2 (q) Place #2 (done)

Fig. 16: Key-frame images for each task, showing important events in successful trajectories. In Kitting-Original (third row), one part is placed in the center of each bin. In Kitting-Modified (last row), two parts are placed in each bin, and neither are at the center location. Kitting-Modified tests adaptation to task distribution shift, where the goal of picking two pieces into the kitting tray is the same, but the task conditions have changed.

operator identify when the pipe is fully inserted.

Kitting. In this task, the robot takes a robot part from each bin and places it into a kitting tray. Performing this task requires two pick and place actions. This task is the longest of the three tasks, requiring on average nearly double the number of steps as Pipe Assembly.

There are two versions of this task: Kitting-Original and Kitting-Modified. In Kitting-Original, only one object part is in each bin at the start of an episode; in Kitting-Modified, each bin starts with two identical parts. See Fig. 7 for key-frame images of Kitting-Original and Kitting-Modified.

The BC policy for this task is trained on demos from the Kitting-Original task. When evaluated on Kitting-Original, the BC policy achieves 0.95 success rate, but in the modified setting, the success rate drops to 0.35. We evaluate on Kitting-Modified to determine how well each method adapts to task distribution shift. For *Q2RL*, the Q-estimation phase is provided with 50 episodes from the Kitting-Original setting, and the online replay buffer is seeded with 30 episodes from the Kitting-Modified setting. For IBRL, which does not have an initial Q-estimation phase, the online replay buffer is seeded with 30 episodes from the Kitting-Modified setting.

Starting Configuration: The bins and kitting tray start in the same location for both Kitting-Original and Kitting-Modified. In Kitting-Original, the sole object part in each bin is placed approximately in the center of the bin with minimal rotation. In Kitting-Modified, the two parts in each bin are placed so that the space in the bin is divided into thirds, so that neither of the parts in each bin are in the same location as in Kitting-Original. The robot gripper starts at the same initial pose for each episode.

Success Condition: One of each part is placed in the kitting tray. In Modified Kitting, where there are two identical parts in each bin, it does not matter which part is picked.

D. Behavior Cloning Policy Details

The BC policies for real world tasks are all Gaussian Mixture Models with ImageNet pre-trained ResNet-10 [1]

TABLE VI: Real-World Evaluation Results

	Peg Insertion		Pipe Assembly		Kitting-Modified	
Method	Success Rate	Online Learner Steps	Success Rate	Online Learner Steps	Success Rate	Online Learner Steps
BC Policy	0.70	–	0.20	–	0.35	–
IBRL [2]	**0.95**	40k	0.0	90k	0.0	165k
Q2RL (Ours)	**1.0**	60k	**0.75**	90k	**0.70**	165k

Success Rate over 20 trials. Results are reported for the best evaluated checkpoint.

vision encoders. For training the BC policies, we use hyperparameters provided by robomimic for the Can-Image task. All BC policies were trained on a single v100 GPU. Table I contains hyperparameters for the BC policy and BC loss weights.

E. Reinforcement Learning Agent Details

All RL agents for real world tasks use a small convolutional neural network as the vision encoder followed by a Gaussian MLP head. The agents were trained and evaluated using stochastic actions, i.e. the actions were sampled from the Gaussian distribution output by the agent. We found that stochastic actions performed better for our real-world, contact-rich manipulation tasks. Table II contains hyperparameters for RL agents. On-robot RL training was run on a single A6000 GPU.

F. Additional Real World Results

Tab. VI contains additional details for the results in Fig. 8. We report the best success rate per task for the evaluated checkpoints. Online learner steps reported in our tables correspond to a single high-UTD learner update. For example, with UTD = 4, one learner update consists of four critic updates and one actor update. The learner steps for *Q2RL* contains the Q-estimation steps as well. Refer to the supplementary video for more qualitative results.

TABLE VII: Peg Insertion without Seeded Replay Buffer

	Peg Insertion, No Data	
Method	Success Rate	Online Learner Steps
BC Policy	0.70	–
CalQL [6] Deterministic	0.10	–
CalQL [6] Stochastic	0.20	–
IBRL [2]	0.0	20k
Q2RL (Ours)	**1.00**	20k

All methods either do not do online learning or start with no data in the online replay buffer. Success rate is over 20 trials. Results are reported for the best evaluated checkpoint.

G. Baseline: Real World CalQL

We report results on the Peg Insertion task for methods that don't use prior data to seed the online replay buffer in Table VII. In this table, we include results for CalQL [6], an offline RL baseline that estimates a conservative Q-function from offline data. We train CalQL on the dataset of successful human demonstrations that we used to train the BC policy, as that is the data that would be available in our setting. CalQL was trained with 250k offline training steps. We evaluated both stochastic and deterministic variants of CalQL. The success rates for both variants was much lower than training a BC policy's 70%. We also observe that stochastic CalQL commanded high-jerk actions that were not in the original demonstration dataset. These high-jerk actions resulted in 4 safety violations during evaluation. These results suggest that offline RL is unable to learn a good Q-function from small datasets of successful demonstrations that are commonly used to train behavior cloning policies. *Q2RL* was able to improve upon the performance of the original BC policy to achieve 100% success over 20 trials. *Q2RL* leverages the fact that BC policies only require successful demonstrations for training, then uses the trained BC policy to estimate a Q-function $\hat{Q}_{\mathrm{BC}}$ (Sec. IV-A). The estimated $\hat{Q}_{\mathrm{BC}}$ is used to train an improved policy using online RL with Q-Gating (Sec. IV-B).

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation bootstrapped reinforcement learning. *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi: 10.15607/RSS.2024.XX.056.

[3] Artemy Kolchinsky and Brendan D Tracey. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7): 361, 2017.

[4] Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16961–16969. IEEE, 2024.

[5] Jianlan Luo, Charles Xu, Fangchen Liu, Liam Tan, Zipeng Lin, Jeffrey Wu, Pieter Abbeel, and Sergey Levine. Fmb: a functional manipulation benchmark for generalizable robotic learning. *The International Journal of Robotics Research*, 44(4):592–606, 2025.

[6] 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*, 36:62244–62269, 2023.

[7] You Liang Tan. Agentlace, framework for distributed agent policy, May 2024. URL https://github.com/youliangtan/agentlace.

[8] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. Daydreamer: World models for physical robot learning. *Conference on Robot Learning*, 2022.

[9] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International conference on learning representations*, 2021.

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

[11] Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HN0CYZbAPw.