

Appendix

A Background

Reinforcement Learning (RL) We study RL as a discounted infinite-horizon Markov Decision Process (MDP) [56, 57]. For pixel observations, the agent’s state is approximated as a stack of consecutive RGB frames [58]. The MDP is of the form $(\mathcal{O}, \mathcal{A}, P, R, \gamma, d_0)$ where \mathcal{O} is the observation space, \mathcal{A} is the action space, $P : \mathcal{O} \times \mathcal{A} \rightarrow \Delta(\mathcal{O})$ is the transition function that defines the probability distribution over the next state given the current state and action, $R : \mathcal{O} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, γ is the discount factor and d_0 is the initial state distribution. The goal is to find a policy $\pi : \mathcal{O} \rightarrow \Delta(\mathcal{A})$ that maximizes the expected discount sum of rewards $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R(o_t, a_t)]$, where $o_0 \sim d_0$, $a_t \sim \pi(o_t)$ and $o_{t+1} \sim P(\cdot | o_t, a_t)$.

Imitation Learning (IL) The goal of imitation learning is to learn a behavior policy π^b given access to either the expert policy π^e or trajectories derived from the expert policy \mathcal{T}^e . While there are a multitude of settings with differing levels of access to the expert [21], this work operates in the setting where the agent only has access to observation-based trajectories, i.e. $\mathcal{T}^e \equiv \{(o_t, a_t)_{t=0}^T\}_{n=0}^N$. Here N and T denotes the number of trajectory rollouts and episode timesteps respectively. We choose this specific setting since obtaining observations and actions from expert or near-expert demonstrators is feasible in real-world settings [59, 60] and falls in line with recent work in this area [13, 6, 7].

Inverse Reinforcement Learning (IRL) IRL [4, 22] tackles the IL problem by inferring the reward function r^e based on expert trajectories \mathcal{T}^e . Then given the inferred reward r^e , policy optimization is used to derive the behavior policy π^b . Prominent algorithms in IRL [7, 6] requires alternating the inference of reward and optimization of policy in an iterative manner, which is practical for restricted model classes [22]. For compatibility with more expressive deep networks, techniques such as adversarial learning [6, 7] or optimal-transport [12, 13, 11] are needed. Adversarial learning based approaches tackle this problem by learning a discriminator that models the gap between the expert trajectories \mathcal{T}^e and behavior trajectories \mathcal{T}^b . The behavior policy π^b is then optimized to minimize this gap through gap-minimizing rewards r^e . Such a training procedure is prone to instabilities since r^e is updated at every iteration and is hence non-stationary for the optimization of π^b .

Optimal Transport for Imitation Learning (OT) To alleviate the non-stationary reward problem with adversarial IRL frameworks, a new line of OT-based approaches have been recently proposed [12, 13, 11]. Intuitively, the closeness between expert trajectories \mathcal{T}^e and behavior trajectories \mathcal{T}^b can be computed by measuring the optimal transport of probability mass from $\mathcal{T}^b \rightarrow \mathcal{T}^e$. During policy learning, the policy π_ϕ encompasses a feature preprocessor f_ϕ which transforms observations into informative state representations. Some examples of a preprocessor function f_ϕ are an identity function, a mean-variance scaling function and a parametric neural network. In this work, we use a parametric neural network as f_ϕ . Given a cost function $c : \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}$ defined in the preprocessor’s output space and an OT objective g , the optimal alignment between an expert trajectory \mathbf{o}^e and a behavior trajectory \mathbf{o}^b can be computed as

$$\mu^* \in \arg \min_{\mu \in \mathcal{M}} g(\mu, f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e), c) \quad (5)$$

where $\mathcal{M} = \{\mu \in \mathbb{R}^{T \times T} : \mu \mathbf{1} = \mu^T \mathbf{1} = \frac{1}{T} \mathbf{1}\}$ is the set of coupling matrices and the cost c can be the Euclidean or Cosine distance. In this work, inspired by [11], we use the entropic Wasserstein distance with cosine cost as our OT metric, which is given by the equation

$$\begin{aligned} g(\mu, f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e), c) &= \mathcal{W}^2(f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e)) \\ &= \sum_{t,t'=1}^T C_{t,t'} \mu_{t,t'} \end{aligned} \quad (6)$$

where the cost matrix $C_{t,t'} = c(f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e))$. Using Eq. 6 and the optimal alignment μ^* obtained by optimizing Eq. 5, a reward signal can be computed for each observation using the equation

$$r^{OT}(o_t^b) = - \sum_{t'=1}^T C_{t,t'} \mu_{t,t'}^* \quad (7)$$

Intuitively, maximizing this reward encourages the imitating agent to produce trajectories that closely match demonstrated trajectories. Since solving Eq. 5 is computationally expensive, approximate solutions such as the Sinkhorn algorithm [61, 12] are used instead.

B Challenges in Online Finetuning from a Pretrained Policy

In this section, we study the challenges with finetuning a pretrained policy with online interactions in the environment. Fig. 2 illustrates a task where an agent is supposed to navigate the environment from the top left to the bottom right, while dodging obstacles in between. The agent has access to a single expert demonstration, which is used to learn a BC policy for the task. Fig. 2 (a) shows that this BC policy, though close to the expert demonstration, performs suboptimally due to accumulating errors on out-of-distribution states during online rollouts [5]. Further, Fig. 2 (b) uses this BC policy as an initialization and naively finetunes it with OT rewards (described in Section 2). Such naive finetuning of a pretrained policy (or actor) with an untrained critic in an actor-critic framework exhibits a forgetting behavior in the actor, resulting in performance degradation as compared to the pretrained policy. This phenomenon has also been reported by Nair et al. [9] and we provide a detailed discussion in Appendix B.1. In this paper, we propose ROT which addresses this issue by adaptively keeping the policy close to the behavior data during the initial phase of finetuning and reduces this dependence over time. Fig. 2 (c) demonstrates the performance of our approach on such finetuning. It can be clearly seen that even though the BC policy is suboptimal, our proposed adaptive regularization scheme quickly improves and solves the task by driving it closer to the expert demonstration. In Fig. 2 (d), we demonstrate that even if the agent was initialized at points outside the expert trajectory, the agent is still able to learn quickly and complete the task. This generalization to starting states would not be possible with regular BC.

B.1 Issue with Fine-tuning Actor-Critic Frameworks

In this paper, we use n -step DDPG proposed by Yarats et al. [8] as our RL optimizer for actor-critic based reward maximization. DDPG [23] concurrently learns a deterministic policy π_ϕ using deterministic policy gradients (DPG) [15] and a Q-function Q_θ by minimizing a n-step Bellman residual (for n -step DDPG). For a parameterized actor network $\pi_\phi(s)$ and a critic function $Q_\theta(s, a)$, the deterministic policy gradients (DPG) for updating the actor weights is given by

$$\begin{aligned} \nabla_\phi J &\approx \mathbb{E}_{s_t \sim \rho_\beta} \left[\nabla_\phi Q_\theta(s, a) \Big|_{s=s_t, a=\pi_\phi(s_t)} \right] \\ &= \mathbb{E}_{s_t \sim \rho_\beta} \left[\nabla_a Q_\theta(s, a) \Big|_{s=s_t, a=\pi_\phi(s_t)} \nabla_\phi \pi_\phi(s) \Big|_{s=s_t} \right] \end{aligned} \quad (8)$$

Here, ρ_β refers to the state visitation distribution of the data present in the replay buffer at time t . From Eq. 8, it is clear that the policy gradients in this framework depend on the gradients with respect to the critic value. Hence, as mentioned in [9, 10], naively initializing the actor with a pretrained policy while using a randomly initialized critic results in the untrained critic providing an exceedingly poor signal to the actor network during training. As a result, the actor performance drops immediately and the good behavior of the informed initialization of the policy gets forgotten. In this paper, we propose an adaptive regularization scheme that permits finetuning a pretrained actor policy in an actor-critic framework. As opposed to Rajeswaran et al. [16], Jena et al. [17] which employ on-policy learning, our method is off-policy and aims to leverage the sample efficient characteristic of off-policy learning as compared to on-policy learning [7].

Algorithm 1 ROT: Regularized Optimal Transport

Require:

Expert Demonstrations $\mathcal{T}^e \equiv \{(o_t, a_t)_{t=0}^T\}_{n=0}^N$
 Pretrained policy π^{BC}
 Replay buffer \mathcal{D} , Training steps T , Episode Length L
 Task environment env
 Parametric networks for RL backbone (e.g., the encoder, policy and critic function for DrQ-v2)
 A discriminator D for adversarial baselines

Algorithm:

```

 $\pi^{ROT} \leftarrow \pi^{BC}$                                  $\triangleright$  Initialize with pretrained policy
for each timestep  $t = 1 \dots T$  do
    if done then
         $r_{1:L} = \text{rewarder}_{OT}(\text{episode})$            $\triangleright$  OT-based reward computation
        Update episode with  $r_{1:L}$  and add  $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}, r_t)$  to  $\mathcal{D}$ 
         $\mathbf{o}_t = env.reset()$ , done = False, episode = []
    end if
     $\mathbf{a}_t = \pi^{ROT}(\mathbf{o}_t)$ 
     $\mathbf{o}_{t+1}$ , done =  $env.step(\mathbf{a}_t)$ 
    episode.append( $[\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}]$ )
    Update backbone-specific networks and reward-specific networks using  $\mathcal{D}$ 
end for

```

C Algorithmic Details

C.1 Implementation

Algorithm 1 describes our proposed algorithm, Regularized Optimal Transport (ROT), for sample efficient imitation learning for continuous control tasks. Further implementation details are as follows:

Algorithm and training procedure Our model consists of 3 primary neural networks - the encoder, the actor and the critic. During the BC pretraining phase, the encoder and the actor are trained using a mean squared error (MSE) on the expert demonstrations. Next, for finetuning, weights of the pretrained encoder and actor are loaded from memory and the critic is initialized randomly. We observed that the performance of the algorithm is not very sensitive to the value of α and we set it to 0.03 for all experiments in this paper. A copy of the pretrained encoder and actor are stored with fixed weights to be used for computing $\lambda(\pi)$ for soft Q-filtering.

Actor-critic based reward maximization We use a recent n-step DDPG proposed by Yarats et al. [8] as our RL backbone. The deterministic actor is trained using deterministic policy gradients (DPG) [15] given by Eq. 8. The critic is trained using clipped double Q-learning similar to Yarats et al. [8] in order to reduce the overestimation bias in the target value. This is done using two Q-functions, Q_{θ_1} and Q_{θ_2} . The critic loss for each critic is given by the equation

$$\mathcal{L}_{\theta_k} = \mathbb{E}_{(s,a) \sim \mathcal{D}_\beta} [(Q_{\theta_k}(s, a) - y)^2] \quad \forall k \in \{1, 2\} \quad (9)$$

where \mathcal{D}_β is the replay buffer for online rollouts and y is the target value for n-step DDPG given by

$$y = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \min_{k=1,2} Q_{\bar{\theta}_k}(s_{t+n}, a_{t+n}) \quad (10)$$

Here, γ is the discount factor, r is the reward obtained using OT-based reward computation and $\bar{\theta}_1$, $\bar{\theta}_2$ are the slow moving weights of target Q-networks.

Target feature processor to stabilize OT rewards The OT rewards are computed on the output of the feature processor f_ϕ which is initialized with a parametric neural network. Hence, as the

weights of f_ϕ change during training, the rewards become non-stationary resulting in unstable training. In order to increase the stability of training, the OT rewards are computed using a target feature processor $f_{\phi'}$ [11] which is updated with the weights of f_ϕ every T_{update} environment steps. For state-based observations, f_ϕ corresponds to a 'trunk' network which is a single layer neural network. For pixel-based observations, f_ϕ includes DrQ-v2's encoder followed by the 'trunk' network.

C.2 Hyperparameters

The complete list of hyperparameters is provided in Table 1. Similar to Yarats et al. [8], there is a slight deviation from the given setting for the Walker Stand/Walk/Run task from the DeepMind Control suite where we use a mini-batch size of 512 and a n -step return of 1.

Method	Parameter	Value
Common	Replay buffer size	150000
	Learning rate	$1e^{-4}$
	Discount γ	0.99
	n -step returns	3
	Action repeat	2
	Seed frames	12000
	Mini-batch size	256
	Agent update frequency	2
	Critic soft-update rate	0.01
	Feature dim	50
	Hidden dim	1024
	Optimizer	Adam
ROT	Exploration steps	0
	DDPG exploration schedule	0.1
	Target feature processor update frequency(steps)	20000
	Reward scale factor	10
	Fixed weight α	0.03
	Linear decay schedule for $\lambda(\pi)$	linear(1,0.1,20000)
OT	Exploration steps	2000
	DDPG exploration schedule	linear(1,0.1,500000)
	Target feature processor update frequency(steps)	20000
	Reward scale factor	10
DAC	Exploration steps	2000
	DDPG exploration schedule	linear(1,0.1,500000)
	Gradient penalty coefficient	10

Table 1: List of hyperparameters.

D Environments

Table 2 lists the different tasks that we experiment with from the DeepMind Control suite [18, 25], OpenAI Robotics suite [26] and the Meta-world suite [27] along with the number of training steps and the number of demonstrations used. For the tasks in the OpenAI Robotics suite, we fix the goal while keeping the initial state randomized. No modifications are made in case of the DeepMind Control suite and the Meta-world suite. The episode length for all tasks in DeepMind Control is 1000 steps, for OpenAI Robotics is 50 steps and Meta-world is 125 steps (except bin picking which runs for 175 steps).

E Demonstrations

For DeepMind Control tasks, we train expert policies using pixel-based DrQ-v2 [8] and collect 10 demonstrations for each task using this expert policy. The expert policy is trained using a stack of 3 consecutive RGB frames of size 84×84 with random crop augmentation. Each action in the environment is repeated 2 times. For OpenAI Robotics tasks, we train a state-based DrQ-v2 with hindsight experience replay [28] and collect 50 demonstrations for each task. The state representation comprises the observation from the environment appended with the desired goal location. For this, we did not do frame stacking and action repeat was set to 2. For Meta-World tasks, we use a single expert demonstration obtained using the task-specific hard-coded policies provided in their open-source implementation [27].

F Robot Tasks

In this section, we describe the suite of manipulation experiments carried out on a xArm robot in this paper.

- (a) **Door Close:** Here, the robot arm is supposed to close an open door by pushing it to the target.
- (b) **Hang Hanger:** While holding a hanger between the grippers, the robot arm is initialized at a random position and is tasked with putting the hanger at a goal region on a closet rod.
- (c) **Erase Board:** While holding a board duster between the grippers, the robot arm is tasked with erasing markings drawn on the board while being initialized at a random position.
- (d) **Reach:** The robot arm is required to reach a specific goal after being initialized at a random position.
- (e) **Hang Mug:** While holding a mug between the grippers, the robot arm is initialized at a random position and is tasked with hanging the mug on a specific hook.
- (f) **Hang Bag:** While holding a tote between the grippers, the robot arm is initialized at a random position and is tasked with hanging the tote bag on a specific hook.
- (g) **Turn Knob:** The robot arm is tasked with rotating a knob placed on the table by a certain angle after being initialized at a random position. We consider a 90 degree rotation as success.
- (h) **Stack Cups:** While holding a cup between the gripper, the robot arm is required with stacking it on another cup placed on the table.
- (i) **Press Switch:** With the gripper kept closed, the robot arm is required to press a switch (with an LED light) placed on the table.
- (j) **Peg (Easy, Medium, Hard):** The robot arm is tasked with inserting a peg, hanging by a wire, into a bucket placed on the table. This task has 3 variants - Easy, Medium, Hard - with the size of the bucket decreasing from Easy to Hard.
- (k) **Box Open:** In this task, the robot arm is supposed to open the lid of a box placed on the table by lifting a handle provided in the front of the box.
- (l) **Pour:** While holding a cup containing some item (in our case, almonds), the robot arm is supposed to move towards another cup placed on the table and pour the item into this cup.

Suite	Tasks	Allowed Steps	# Demonstrations
DeepMind Control	Acrobot Swingup	2×10^6	10
	Cartpole Swingup		
	Cheetah Run		
	Finger Spin		
	Hopper Stand		
	Hopper Hop		
	Quadruped Run		
	Walker Stand		
	Walker Walk		
	Walker Run		
OpenAI Robotics	Fetch Reach	1.5×10^6	50
	Fetch Push		
	Fetch Pick and Place		
Meta-World	Hammer	1×10^6	1
	Drawer Close		
	Door Open		
	Bin Picking		
	Button Press Topdown		
	Door Unlock.		
xArm Robot	Close Door	6×10^3	1
	Hang Hanger		
	Erase Board		
	Reach		
	Hang Mug		
	Hang Bag		
	Turn Knob		
	Stack Cups		
	Press Switch		
	Peg (Easy)		
	Peg (Medium)		
	Peg (Hard)		
	Open Box		
	Pour		

Table 2: List of tasks used for evaluation.

Evaluation procedure For each task, we obtained a set of 20 random initializations and evaluate all of the methods (BC, RDAC and ROT) over 20 trajectories from the same set of initializations. These initializations are different for each task based on the limits of the observation space for the task.



Figure 7: Examples of randomized initializations for the real robot tasks.

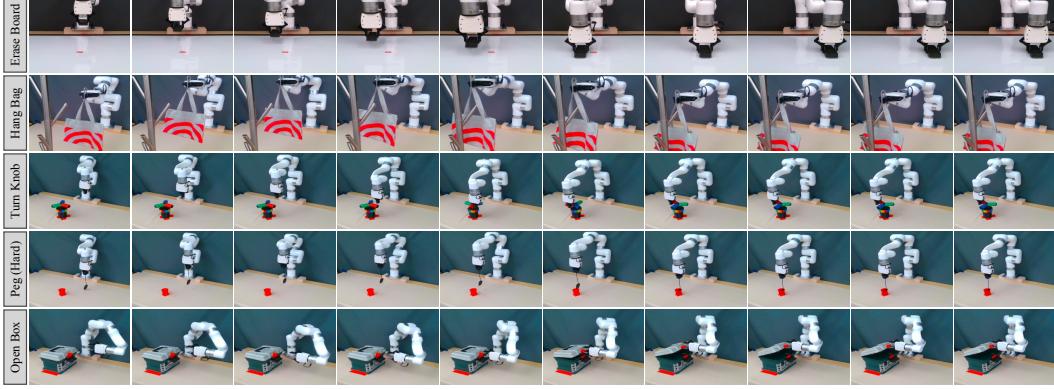


Figure 8: An example of trajectories for selected real robot tasks.

G Baselines

Throughout the paper, we compare ROT with several prominent imitation learning and reinforcement learning methods. Here, we give a brief description of each of the baseline models that have been used.

- (a) **Expert:** For each task, the expert refers to the expert policy used to generate the demonstrations for the task (described in Appendix E).
- (b) **Behavior Cloning (BC):** This refers to the behavior cloned policy trained on expert demonstrations.
- (c) **Adversarial IRL (DAC):** Discriminator Actor Critic [7] is a state-of-the-art adversarial imitation learning method [6, 29, 7]. Since DAC outperforms prior work such as GAIL[6] and AIRL[30], it serves as our primary adversarial imitation baseline.
- (d) **State-matching IRL (OT):** Sinkhorn Imitation Learning [12, 13] is a state-of-the-art state-matching imitation learning method [31] that approximates OT matching through the Sinkhorn Knopp algorithm. Since ROT is derived from similar OT-based foundations, we use SIL as our primary state-matching imitation baseline.
- (e) **RDAC:** This is the same as ROT, but instead of using state-matching IRL (OT), adversarial IRL (DAC) is used.
- (f) **Finetune with fixed weight:** This is similar to ROT where instead of using a time-varying adaptive weight $\lambda(i)$, only the fixed weight λ_0 is used. λ_0 is set to a fixed value of 0.03.
- (g) **Finetune with fixed schedule:** This is similar to ROT that uses both the fixed weight λ_0 and the time-varying adaptive weight $\lambda_1(i)$. However, instead of using Soft Q-filtering to compute $\lambda_1(i)$, a hand-coded linear decay schedule is used.
- (h) **DrQ-v2 (RL):** DrQ-v2 [8] is a state-of-the-art algorithm for pixel-based RL. DrQ-v2 is assumed to have access to environment rewards as opposed to ROT which computes the reward using OT-based techniques.
- (i) **Demo-DrQ-v2:** This refers to DrQ-v2 but with access to both environment rewards and expert demonstrations. The model is initialized with a pretrained BC policy followed by RL finetuning with an adaptive regularization scheme like ROT. During RL finetuning, this baseline has access to environment rewards.
- (j) **BC+OT:** This is the same as the OT baseline but the policy is initialized with a pretrained BC policy. No adaptive regularization scheme is used while finetuning the pretrained policy.
- (k) **OT+BC Reg.:** This is the same as the OT baseline with randomly initialized networks but during training, the adaptive regularization scheme is added to the objective function.

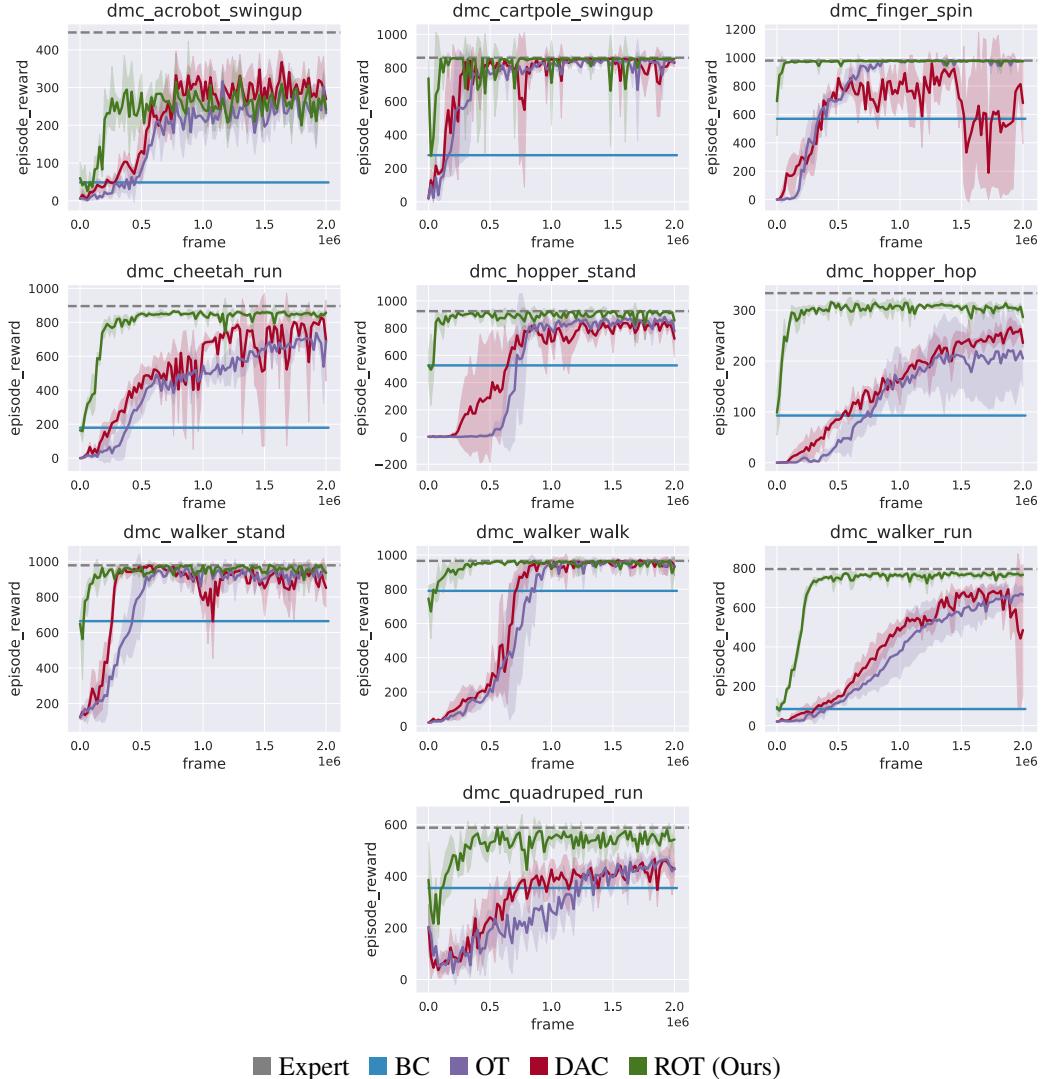


Figure 9: Pixel-based continuous control learning on 10 DMC environments. Shaded region represents ± 1 standard deviation across 5 seeds. We notice that ROT is significantly more sample efficient compared to prior work.

H Additional Experimental Results

H.1 How efficient is ROT for imitation learning?

In addition to the results provided in Sec. 4.1, Fig. 9 and Fig. 10 shows the performance of ROT for pixel-based imitation on 10 tasks from the DeepMind Control suite, 3 tasks from the OpenAI Robotics suite and 7 tasks from the Meta-world suite. On all but one task, ROT is significantly more sample efficient than prior work. Finally, the improvements from ROT hold on state-based observations as well(see Fig. 11). Table 3 provides a comparison between the factor of speedup of ROT to reach 90% of expert performance compared to prior state-of-the-art [7, 11] methods.

H.2 Does soft Q-filtering improve imitation?

Extending the results shown in Fig. 6, we provide training curves from representative tasks in each suite in Fig. 12. We observe that our adaptive soft-Q filtering regularization is more stable compared

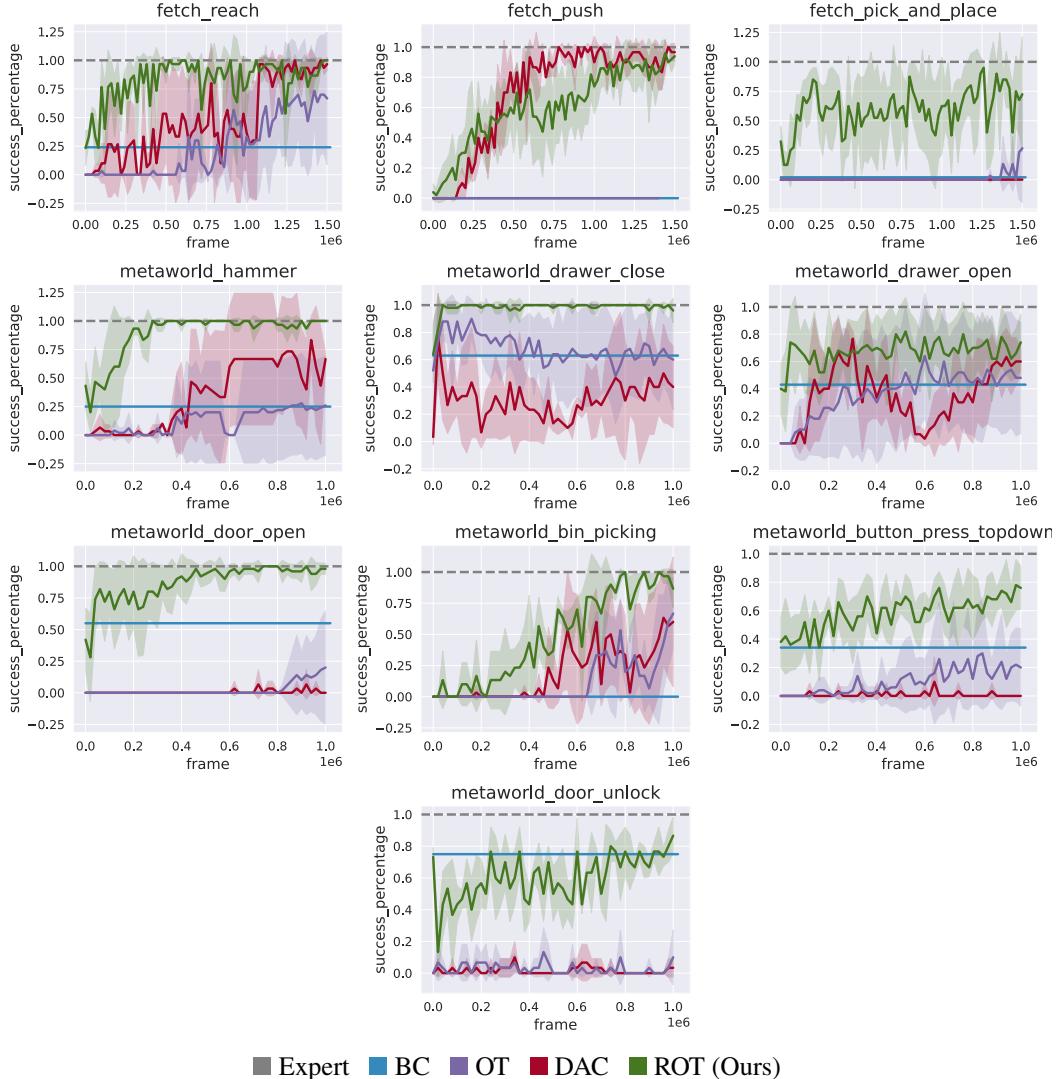


Figure 10: Pixel-based continuous control learning on 3 OpenAI Gym Robotics and 7 Meta-World tasks. Shaded region represents ± 1 standard deviation across 5 seeds. We notice that ROT is significantly more sample efficient compared to prior work.

to prior hand-tuned regularization schemes. ROT is on par and in some cases exceeds the efficiency of a hand-tuned decay schedule, while not having to hand-tune its regularization weights.

H.3 How does ROT compare to standard reward-based RL?

Extending the results shown in Fig. 5, we provide training curves from representative tasks in each suite in Fig. 13, thus showing that ROT can outperform standard RL that requires explicit task-reward. We also show that this RL method combined with our regularization scheme (represented by Demo-DrQ-v2 in Fig. 13 provides strong results.

H.4 How important are the design choices in ROT?

Importance of pretraining and regularizing the IRL policy Fig. 14 compares the following variants of ROT on set of pixel-based tasks: (a) Training the IRL policy from scratch (OT); (b) Finetuning a pretrained BC policy without BC regularization (BC+OT); (c) Training the IRL policy from scratch with BC regularization (OT+BC Reg.). We observe that pretraining the IRL policy

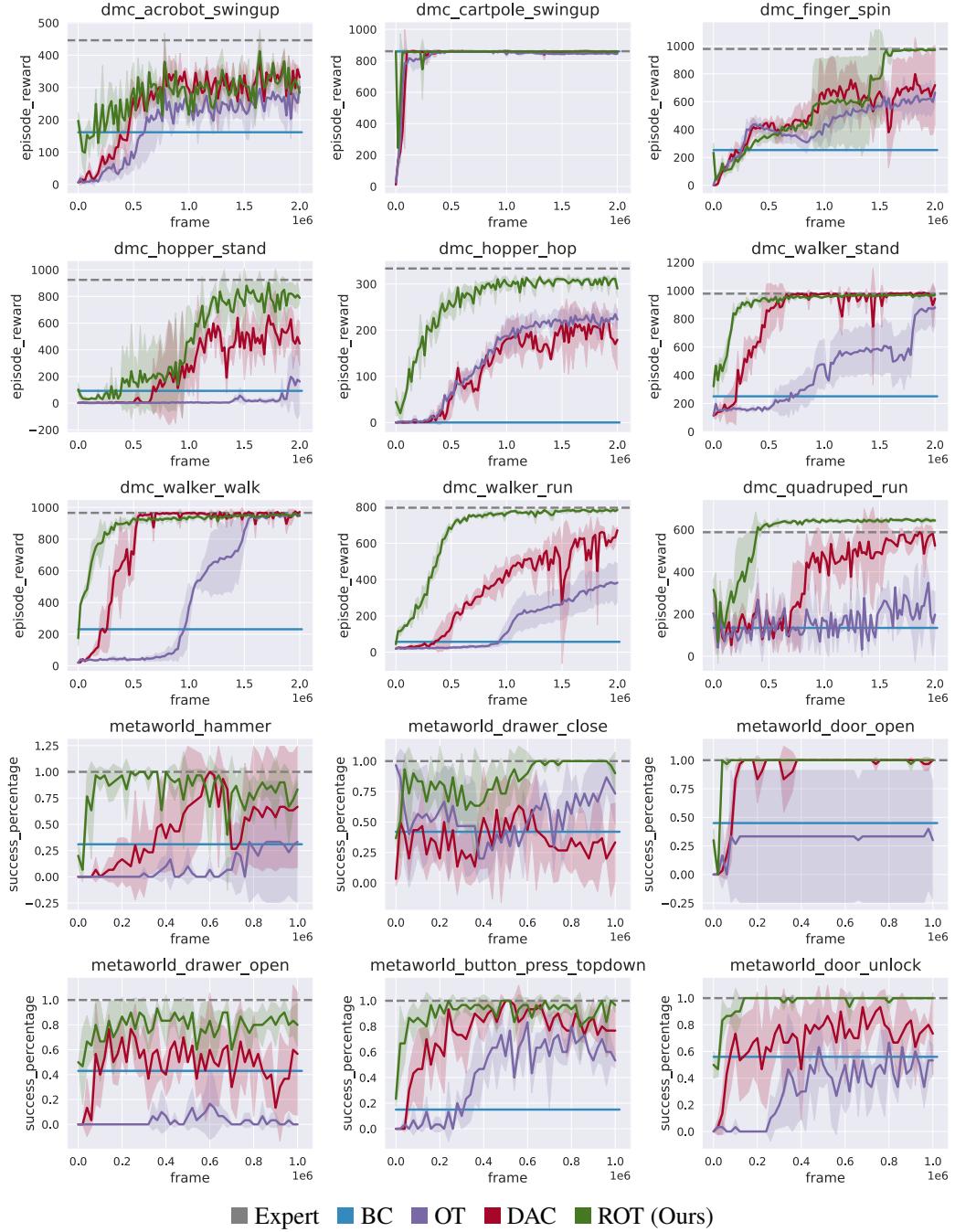


Figure 11: State-based continuous control learning on DMC and Meta-World tasks. We notice that ROT is significantly more sample efficient compared to prior work.

(BC+OT) does not provide a significant difference without regularization. This can be attributed to the ‘forgetting behavior’ of pre-trained policies, studied in Nair et al. [9]. Interestingly, we see that even without BC pretraining, keeping the policy close to a behavior distribution (OT+BC Reg.) can yield improvements in efficiency over vanilla training from scratch. Our key takeaway from these experiments is that both pretraining and BC regularization are required to obtain sample-efficient imitation learning.

Suite	Tasks	ROT	2nd Best Model	Speedup Factor
DeepMind Control	Acrobot Swingup	200k	600k (OT)	3
	Cartpole Swingup	100k	350k (OT)	3.5
	Finger Spin	20k	700k (OT)	35
	Cheetah Run	400k	2M (DAC)	5
	Hopper Stand	60k.	750k (OT)	12.5
	Hopper Hop	200k	>2M (DAC)	10
	Walker Stand	80k	400k (DAC)	5
	Walker Walk	200k	750k (DAC)	3.75
	Walker Run	320k	>2M (OT)	6.25
	Quadruped Run	400k	>2M (DAC)	5
OpenAI Robotics	Fetch Reach	300k	1.1M (DAC)	3.67
	Fetch Push	1.1M	600k (DAC)	0.54
	Fetch Pick and Place	750k	>1.5M (OT)	2
Meta-World	Hammer	200k	>1M (DAC)	5
	Drawer Close	20k	>1M (OT)	50
	Drawer Open	>1M	>1M (OT)	1
	Door Open	400k	>1M (OT)	2.5
	Bin Picking	700k	>1M (OT)	1.43
	Button Press Topdown	>1M	>1M (OT)	1
	Door Unlock	1M	>1M (OT)	1

Table 3: Task-wise comparison between environment steps required to reach 90% of expert performance for pixel-based ROT compared to the strongest baseline for each task.

Choice of IRL method In ROT, we build on OT-based IRL instead of adversarial IRL. This is because adversarial IRL methods require iterative reward learning, which produces a highly non-stationary reward function for policy optimization. In Fig. 15, we compare ROT with adversarial IRL methods that use our pretraining and adaptive BC regularization technique (RDAC). We find that our soft Q-filtering method does improve prior state-of-the-art adversarial IRL (RDAC vs. DAC in Fig. 15). However, our OT-based approach (ROT) is more stable and on average leads to more efficient learning.

Choice of Q-filtering method In ROT, we adopt a soft Q-filtering method as opposed to the hard assignment strategy proposed by Nair et al. [24]. Fig. 16 shows a comparison between the performance of soft Q-filtering and hard Q-filtering. We observe that though the two strategies have comparable performance in most cases, soft Q-filtering exhibits better sample efficiency and more stable training in some tasks. This justifies our choice of opting for soft Q-filtering as opposed to hard assignment.

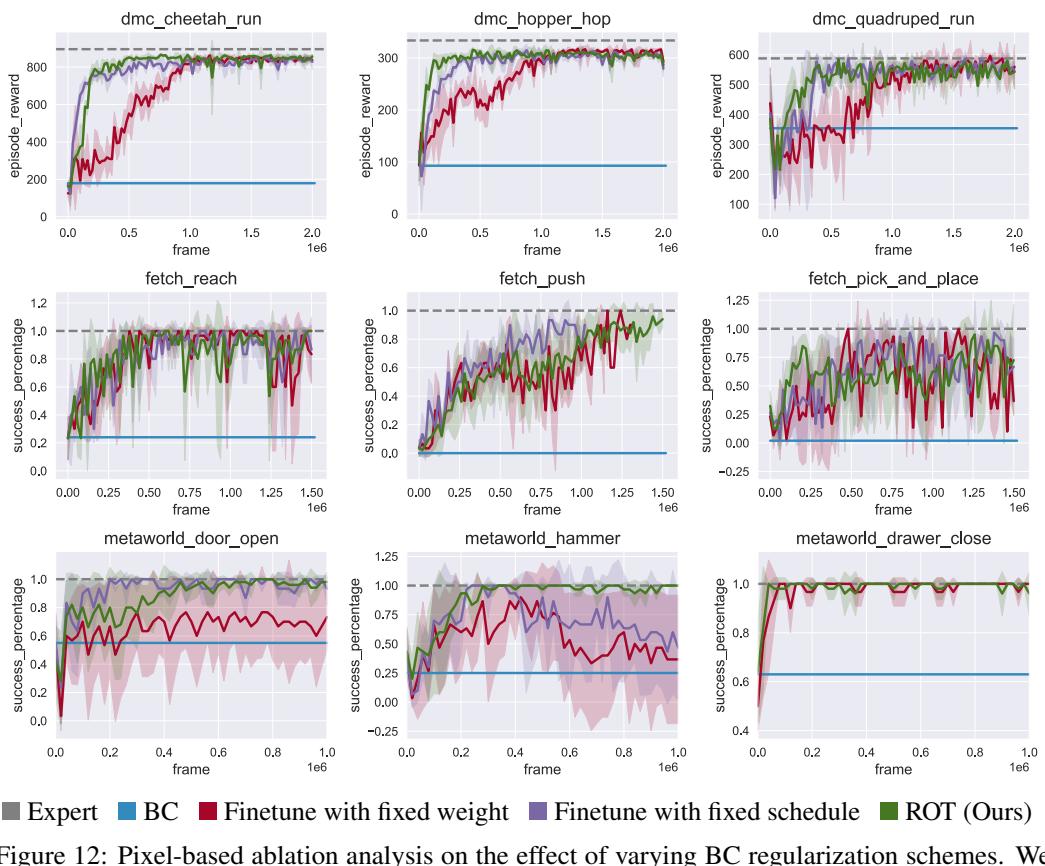


Figure 12: Pixel-based ablation analysis on the effect of varying BC regularization schemes. We observe that our adaptive soft-Q filtering regularization is more stable compared to prior hand-tuned regularization schemes.

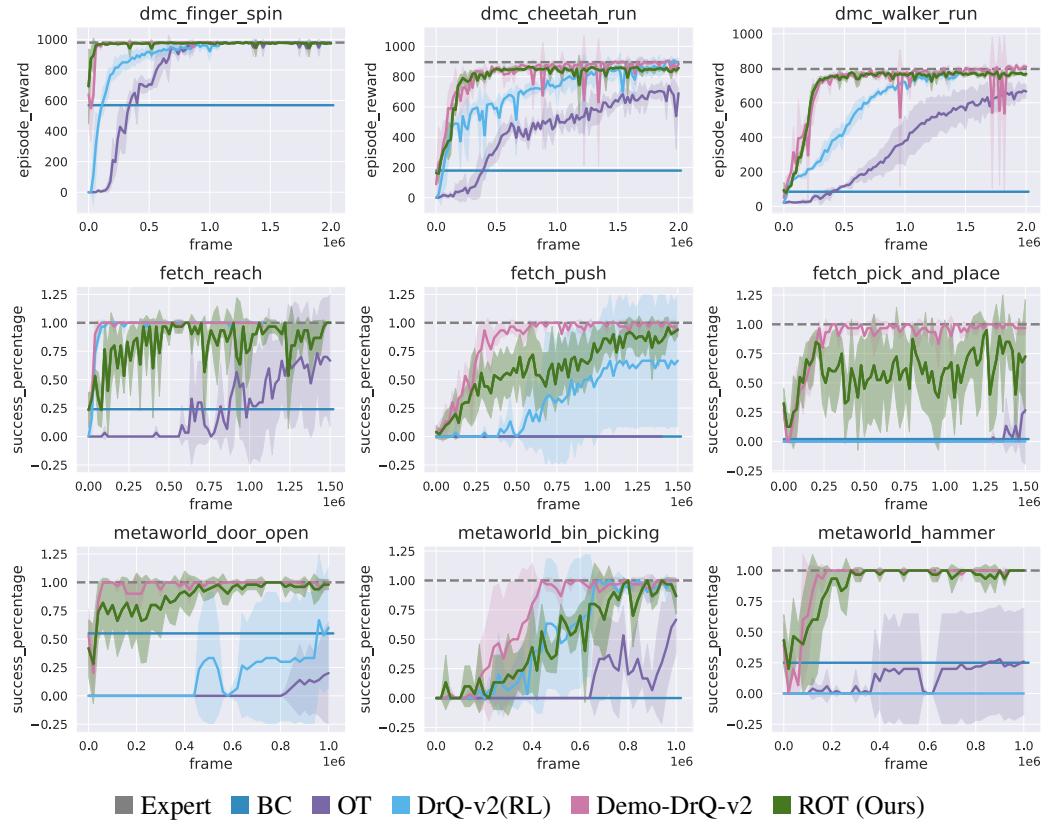


Figure 13: Pixel-based ablation analysis on the performance comparison of ROT against DrQ-v2, a reward-based RL method. Here we see that ROT can outperform plain RL that requires explicit task-reward. However, we also observe that this RL method combined with our regularization scheme provides strong results.

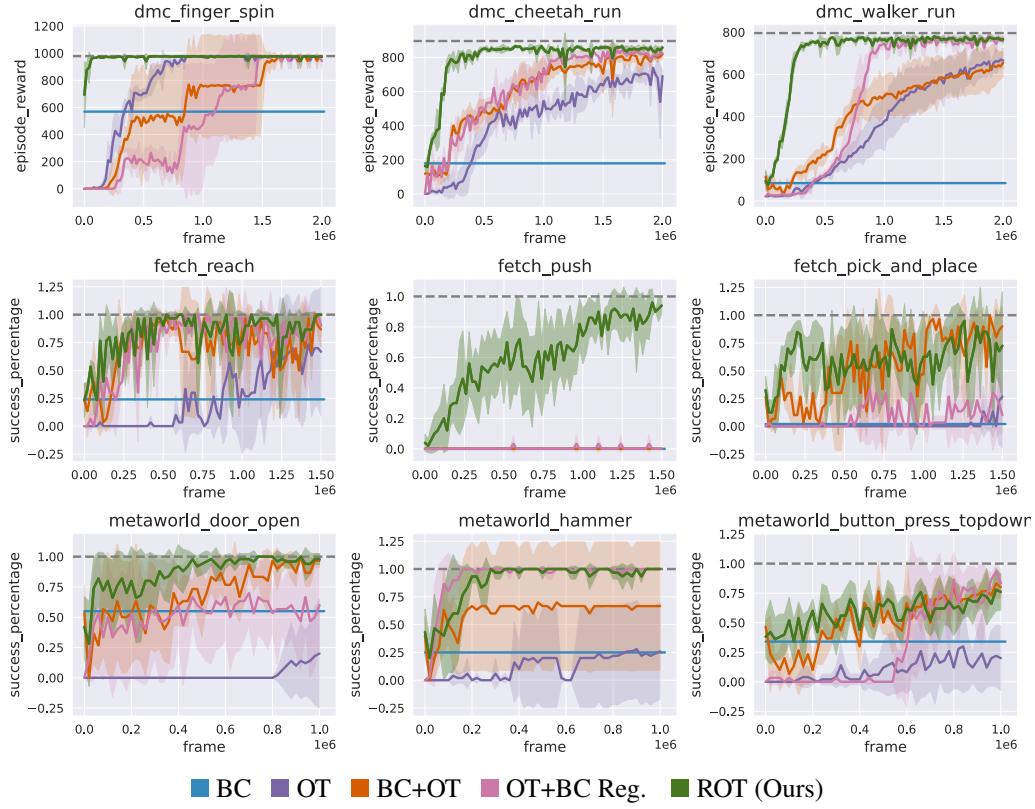


Figure 14: Pixel-based ablation analysis on the importance of pretraining and regularizing the IRL policy. The key takeaway from these experiments is that both pretraining and BC regularization are required to obtain sample-efficient imitation learning.

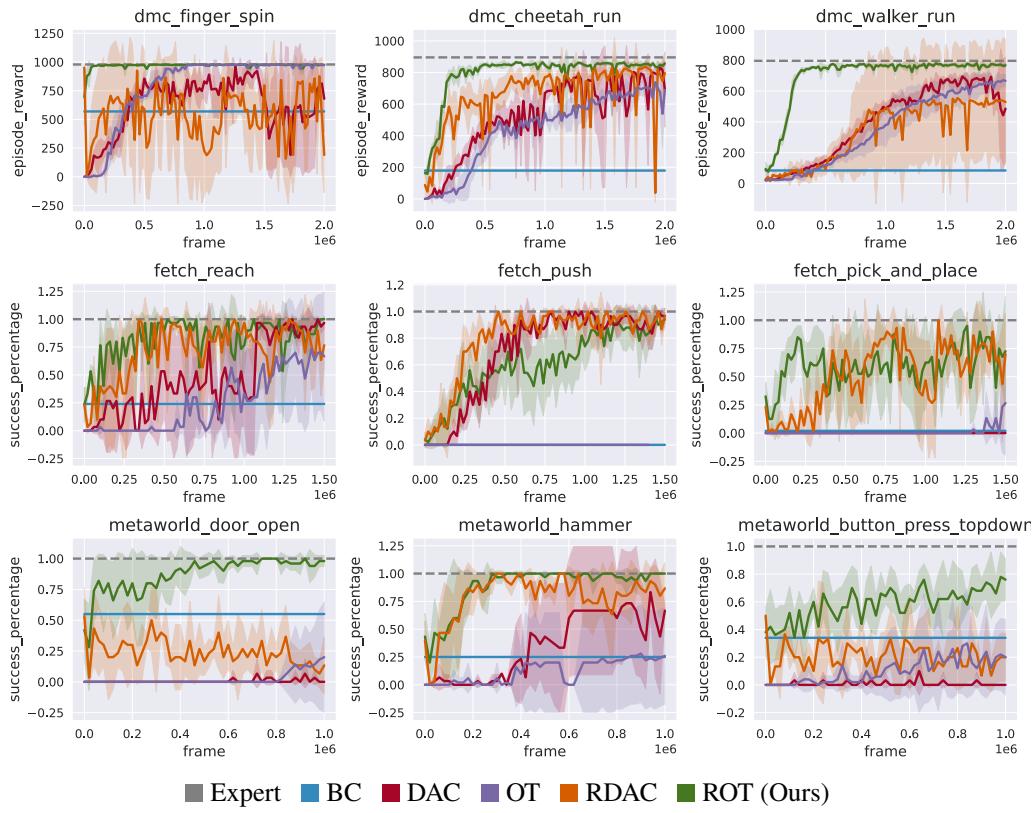


Figure 15: Pixel-based ablation analysis on the choice of base IRL method. We find that although adversarial methods benefit from regularized BC, the gains seen are smaller compared to ROT.

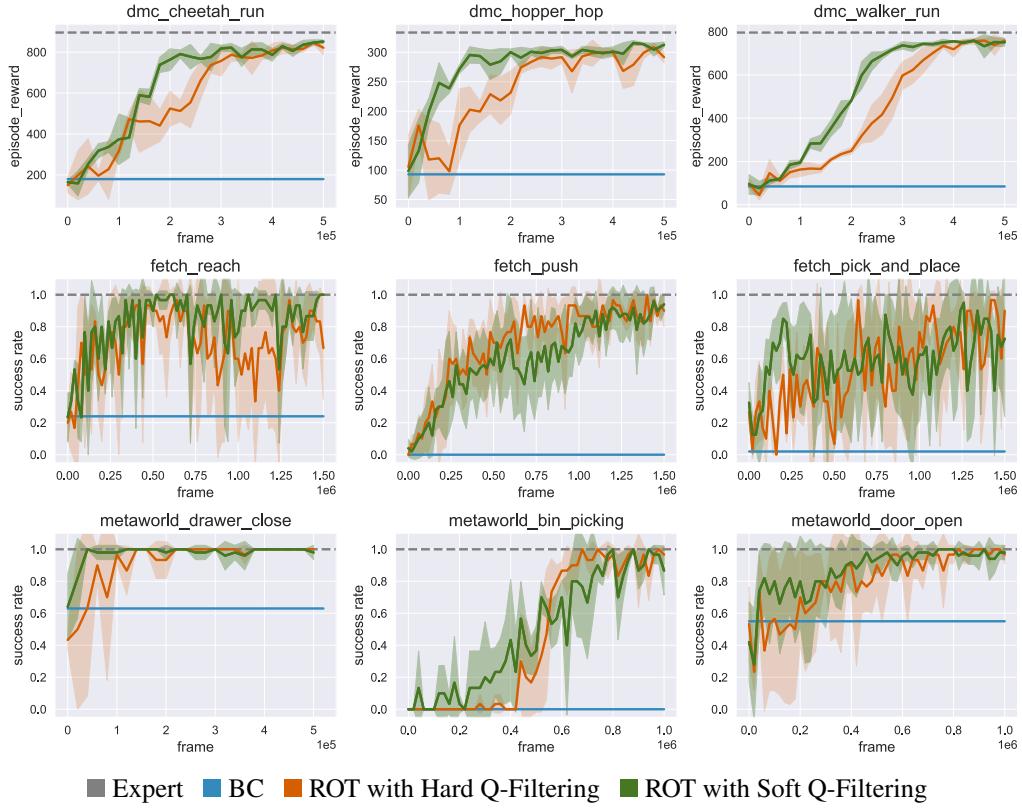


Figure 16: Pixel-based ablation analysis on the choice of Q-filtering method. We find that although the two strategies have comparable performance in most cases, soft Q-filtering exhibits better sample efficiency and more stable training in some tasks.