A Implementation Details

For all algorithms, all Q functions and policies are approximated using deep neural networks with 2 hidden layers of size 256. They are all updated using the Adam optimizer from Kingma and Ba [2015].

A.1 Behavioral Cloning

We used a straightforward implementation of behavioral cloning, regressing with a mean square error loss. For all experiments learners were provided with the same number of demonstrators as the other algorithms and optimized for 10000 gradient steps using a learning rate of 1×10^{-4}.

A.2 Twin Delayed Deep Deterministic Policy Gradients

We use the author implementation of TD3 from Fujimoto et al. [2018], modifying it to implement MCAC. In order to maintain the assumption about complete trajectories described in the main text, we modify the algorithm to only add to the replay buffer at the end of sampled trajectories, but continue to update after each timestep. We found the default hyperparameters from the repository to be sufficient in all environments.

A.3 Soft Actor Critic

For all SAC experiments we used a modified version of the SAC implementation from Tandon which implements SAC with a double-Q critic update function to combat Q overestimation. Additionally, we modify the algorithm to satisfy the trajectory assumption as in Section A.2. We mostly use the default hyperparameters from Haarnoja et al. [2018], but tuned α and τ. Parameter choices are shown in Table 1.

Table 1: Hyperparameters for SAC

Parameter	Navigation	MuJoCo	Robosuite
Learning Rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Automatic Entropy Tuning	False	False	False
Batch Size	256	256	256
Hidden Layer Size	256	256	256
# Hidden Layers	2	2	2
# Updates Per Timestep	1	1	1
α	0.2	0.1	0.05
γ	0.99	0.99	0.99
τ	5×10^{-2}	5×10^{-3}	5×10^{-2}

A.4 Generalized Q Estimation

Similarly to the advantage estimation method in Schulman et al. [2016], we estimate Q values by computing a weighted average over a number of Q estimates estimated with k-step look-aheads. Concretely, if a $Q_t^{(k)}$ is a Q estimate with a k-step look-ahead given by

$$Q_t^{(k)} = \sum_{i=0}^{k-1} r_{t+i} + \gamma^k Q_\theta(s_{t+k}, a_{t+k}), \tag{A.1}$$

we compute the n-step GQE estimate Q_t^{GQE} as

$$Q_t^{\mathrm{GQE}} = \frac{1-\lambda}{1-\lambda^n} \sum_{k=1}^{n} \lambda^{k-1} Q_t^{(k)}. \tag{A.2}$$

We built GQE on top of SAC, using the SAC Q estimates for the values of Q_θ. However, in principle this method can be applied to other actor-critic methods.

Where applicable we used the hyperparameters from SAC, and tuned the values of λ and n as hyperparameters, trying values in the sets $\lambda \in \{0.8, 0.9, 0.95, 0.99, 0.999\}$ and n values in the set $n \in \{8, 16, 32, 64\}$. The chosen parameters for each environment are given in Table 2.

Table 2: Hyperparameters for GQE

Parameter	Navigation	Extraction	Push	Door	Lift
λ	0.9	0.95	0.95	0.9	0.9
n	32	8	16	16	16

A.5 Overcoming Exploration with Demonstrations

We implement the algorithm from Nair et al. [2018] on top of the implementation of TD3 described in Section A.2. Because it would provide an unfair advantage over comparisons, the agent is not given the ability to reset to arbitrary states in the replay buffer. Since our setting is not goal-conditioned, our implementation does not include hindsight experience replay. For the value λ balancing the actor critic policy update loss and behavioral cloning loss, we use $\lambda = 1.0$. In all experiments the agent is pretrained on offline data for 10000 gradient steps.

A.6 Conservative Q-learning

Offline reinforcement learning algorithm that produces a lower bound on the value of the current policy. We used the implementation from [Dittert, 2021], which implements CQL on top of SAC as is done in the original paper, modified for additional online-finetuning. We used the default hyperparameters from [Kumar et al., 2020] in all environments and pretrained the agent on offline data for 10000 gradient steps.

A.7 Advantage Weighted Actor Critic

For AWAC experiments we use the implementation from Sikchi and Wilcox, once again modifying it to maintain the assumption about complete trajectories and to implement MCAC. We found the default hyperparameter values to be sufficient in all settings. In all experiments the agent is pretrained on offline data for 10000 gradient steps.

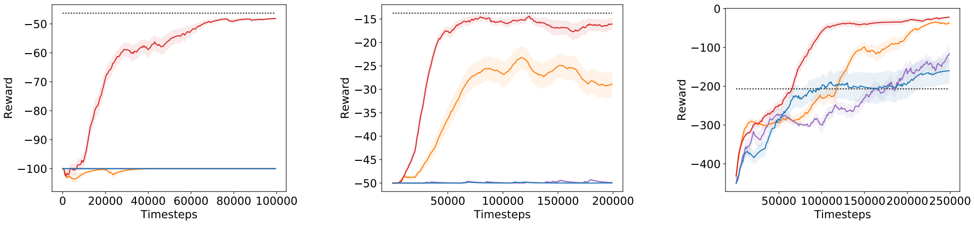
B Additional Experiments

B.1 No Demonstrations

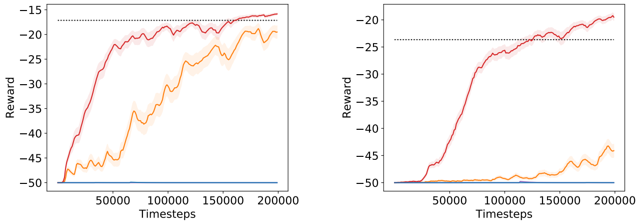
To study the effects that MCAC has without demonstration data in the replay buffer we compare performance with and without MCAC and demonstrations in all environments with the SAC learner, shown in Figure 5. Overall we see that, as desired, the agent is unable to make progress in most environments. The only exception is the sequential pushing environment results (Figure 5(c)), where the intermediate reward for pushing each block helps the agent learn to make some progress. Overall, this experiment does not conclusively answer whether MCAC is helpful without demonstrations, but this is an exciting direction for future work.

B.2 MCAC Sensitivity Experiments

In Figure 6, we first study the impact of demonstration quality (Figure 6(a)) and quantity (Figure 6(b)) on MCAC when applied to SAC (SAC + MCAC) on the Pointmass Navigation domain. We evaluate sensitivity to demonstration quality by injecting ϵ-greedy noise into the demonstrator for the Pointmass Navigation domain. Results suggest that MCAC is somewhat sensitive to demonstration quality, since MCAC's performance does drop significantly for most values of ϵ, although it still typically makes some task progress. In Figure 6(b), results suggest that MCAC is relatively robust to the number of demonstration.



(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing



(d) Door Opening (e) Block Lifting

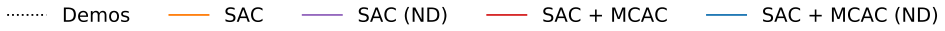
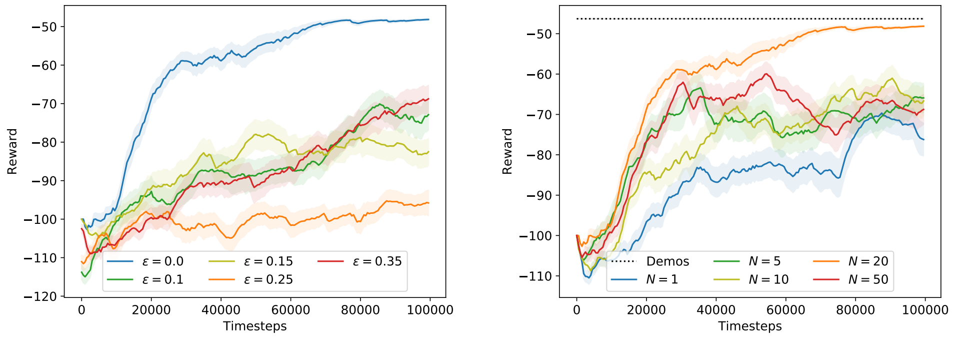


Figure 5: **MCAC without Demonstrations:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds for experiments with demonstrations and 3 seeds for experiments without them. We see that in all environments the demonstrations are critical for learning an optimal policy. The only place the variants without demonstrations make progress is in the push environment, because of the intermediate reward for pushing each block.

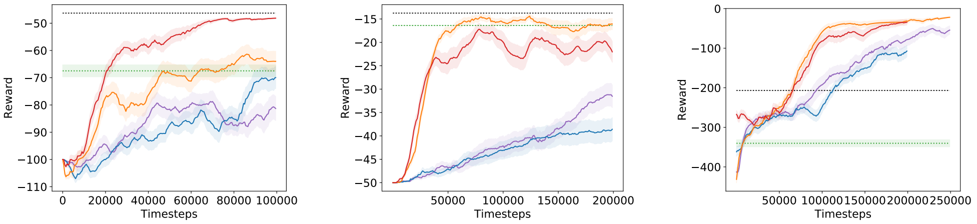


(a) Demonstration Quality (b) Demonstration Quantity

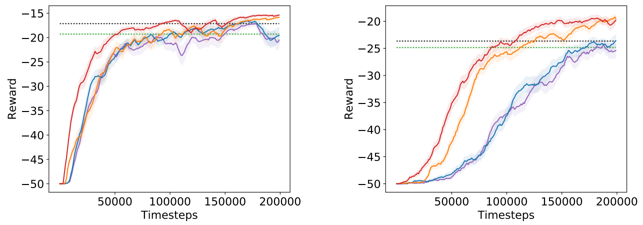
Figure 6: **MCAC Sensitivity Experiments:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds for varying demonstration qualities (a) and quantities (b) for SAC + MCAC. (a): Results suggest that MCAC is somewhat sensitive to demonstration quality, as when ϵ-greedy noise is injected into the demonstrator, MCAC's performance does drop significantly, although it eventually make some task progress for most values of ϵ. (b): MCAC appears to be much less sensitive to demonstration quantity, and is able to achieve relatively high performance even with a single task demonstration.

B.3 Pretraining

In Figure 7 we ablate on the pretraining step of the algorithm to determine whether it is useful to include. We find that it was helpful for the Navigation environment but unhelpful and sometimes limiting in other settings. Thus, we leave pretraining as a hyperparameter to be tuned.



(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing



(d) Door Opening (e) Block Lifting

Figure 7: **MCAC with and without Pretraining Results:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds. We find that other than in the navigation environment pretraining does not provide a significant benefit.

B.4 Lambda Weighting

As an additional comparison we consider computing a weighted combination of the Monte Carlo and Bellman updates, using a Q target $Q^{\text{target}}_{\lambda}$ given by

$$Q^{\text{target}}_{\lambda} = (1-\lambda)Q^{\text{target}}(s^i_j, a^i_j) + \lambda Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j). \tag{B.1}$$

Results of this experiment are shown in Figure 8 alongside results from our GQE experiments. We find that this method is comparable to GQE in most environments.

B.5 Critic Tail

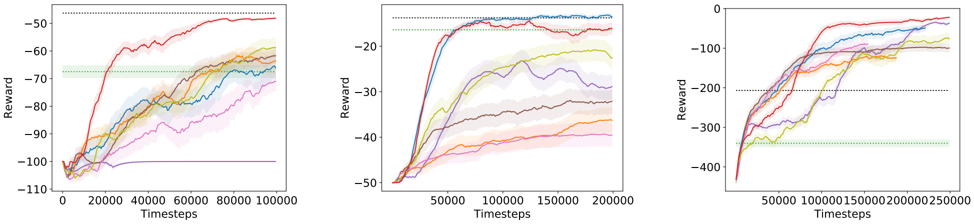
In this experiment, we consider replacing $Q^{\text{target}}_{\text{MC-}\infty}$ defined in the main text with a new target, which replaces the infinite geometric sum with the critic's Q value estimate. Formally, for a trajectory given by $[(s_i, a_t, r_i), (s_{i+1}, a_{i+1}, r_{i+1}), \ldots, (s_T, a_T, r_T), s_{T+1}]$, a delayed target network $Q^{\pi}_{\theta'}$ and a policy π, the new target is

$$\hat{Q}^{\text{target}}_{\text{MC-}\infty}(s_i, a_i) = \gamma^{T-j+1} Q^{\pi}_{\theta'}(s_{T+1}, \pi(s_{T+1})) + \sum_{k=j}^{T} \gamma^{k-j} r_k. \tag{B.2}$$

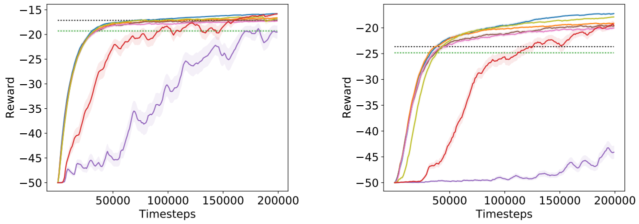
We compare this version to the standard version of MCAC presented in the paper in Figure 10. Results suggest that the version presented in the main paper performs at least as well if not better than this version. Since this version also adds implementation difficulty and computation load, we choose to use the original $Q^{\text{target}}_{\text{MC-}\infty}$ estimate. However, it would be interesting in future work to study whether there are environments where this estimate has better performance.

B.6 Q Estimate Scale Analysis

In this experiment we investigate how the scales of the various Q value estimates we consider vary as the agent trains. In Figure 10(a) we present average values for Q estimates based on samples from the replay buffer for successful trajectories, while in Figure 10(b) we present the same data for failed trajectories. The results suggest that early on, Monte Carlo estimates are much higher than Bellman

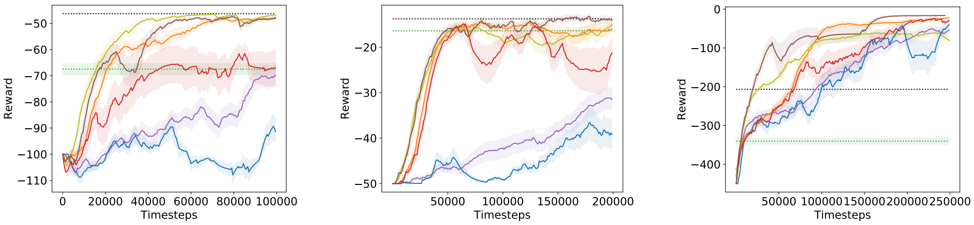


(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing

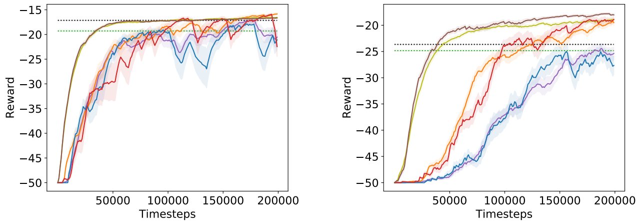


(d) Door Opening (e) Block Lifting

Figure 8: **MCAC with Various λ Weighting Results:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds. Results suggest that this method performs similarly to GQE in most environments.



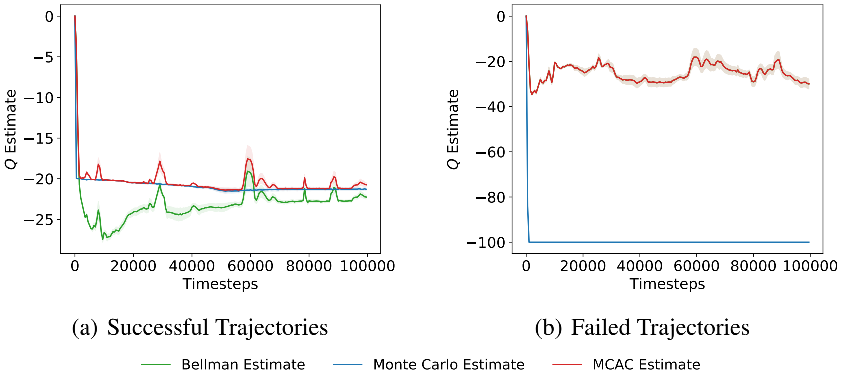
(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing



(d) Door Opening (e) Block Lifting

Figure 9: **MCAC with Critic Tail:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds for experiments with the original $Q^{\text{target}}_{\text{MC-}\infty}$ estimate and 3 seeds for experiments with the new version described in Equation B.2. We see that in all environments the original version is at least as good as the the new version, and occasionally it performs better.

estimates for successful trajectories, but in the limit these two estimates converge to similar values. This confirms our claims that MCAC helps to drive up Q values along demonstrator trajectories early in training. For failed trajectories, Monte Carlo estimates are uniformly as low as possible but the Bellman estimate and MCAC are identical because the low Monte Carlo estimates are always cancelled out by the $\max$ term in the MCAC estimate.



(a) Successful Trajectories

(b) Failed Trajectories

Figure 10: **Estimator Scale Experiment:** Plots show mean and standard error across three random seeds of the various Q estimates discussed in the paper based on uniform samples from the replay buffer for (a) successful and (b) failed trajectories. (b) appears to only have two lines because the data for Bellman estimates and MCAC estimates are identical, as expected for failed trajectories.

B.7 AWAC Q Estimates

In this section we briefly study the results of the AWAC+MCAC experiment in the navigation environment, where AWAC's policy fell apart during online training while AWAC+MCAC was more stable. Although there are too many factors at play to arrive at a concrete conclusion, a factor that likely differentiated the algorithms is the Q value scales, presented in Figure 11. In these results, which show the mean Q estimates computed during gradient updates for AWAC with and without MCAC, we see that MCAC helps to prevent the Q values from crashing with the addition of online data, which they do without it. This phenomenon likely played a role in the difference between the two algorithms.

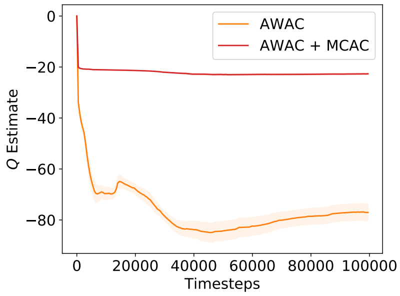


Figure 11: **AWAC Q Estimates:** Plots show mean and standard error across five random seeds of the Q estimates from the AWAC and AWAC+MCAC learners. At the 0th timestep, each algorithm has undergone a pretraining procedure as described in Section A.7. We see that while the original Q estimate plummets after online data is added to the buffer, the version with MCAC stays relatively constant.