Monte Carlo Augmented Actor-Critic for Sparse Reward Deep Reinforcement Learning from Suboptimal Demonstrations

Albert Wilcox
UC Berkeley
albertwilcox@berkeley.edu

Ashwin Balakrishna
UC Berkeley
ashwin_balakrishna@berkeley.edu

Jules Dedieu
UC Berkeley
jules_dedieu@berkeley.edu

Wyame Benslimane
UC Berkeley
wyame.benslimane@berkeley.edu

Daniel Brown
University of Utah
dsbrown@cs.utah.edu

Ken Goldberg
UC Berkeley
goldberg@berkeley.edu

Abstract

Providing densely shaped reward functions for RL algorithms is often exceedingly challenging, motivating the development of RL algorithms that can learn from easier-to-specify sparse reward functions. This sparsity poses new exploration challenges; one common response is to use demonstrations to provide initial signal about regions of the state space with high rewards. However, prior RL from demonstrations algorithms introduce significant complexity and many hyperparameters, making them hard to implement and tune. We introduce Monte Carlo augmented Actor-Critic (MCAC), a parameter free modification to standard actor-critic algorithms which initializes the replay buffer with demonstrations and computes a modified Q-value by taking the maximum of the standard temporal distance (TD) target and a Monte Carlo estimate of the reward-to-go. This encourages exploration in the neighborhood of high-performing trajectories by encouraging high Q-values in corresponding regions of the state space. Experiments across 5 continuous control domains suggest that MCAC can be used to significantly increase learning efficiency for a number of prior RL and RL-from-demonstrations algorithms.

1 Introduction

Reinforcement learning has been successful in learning complex skills in many environments [Mnih et al., 2015, Silver et al., 2016, Schulman et al., 2017], but providing dense, informative reward functions for RL agents is often very challenging [Xie et al., 2019, Krakovna et al., 2020, Wu et al., 2020]. This is particularly challenging for high-dimensional control tasks, in which there may be a large number of factors that influence the agent's objective. In many settings, it may be much easier to provide sparse reward signals that simply convey high-level information about task progress, such as whether an agent has completed a task or has violated a constraint. However, optimizing RL policies given such reward signals can be exceedingly challenging, as sparse reward functions may not be able to meaningfully distinguish between a wide range of different policies.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

This issue can be mitigated by leveraging demonstrations, which provide initial signal about desired behaviors. Though demonstrations may often be suboptimal in practice, they should still serve to encourage exploration in promising regions of the state space, while allowing the agent to explore behaviors which may outperform the demonstrations. Prior work has considered a number of ways to leverage demonstrations to improve learning efficiency for reinforcement learning, including by initializing the policy to match the behavior of the demonstrator [Rajeswaran et al., 2018, Peng et al., 2019], using demonstrations to explicitly constrain agent exploration [Thananjeyan et al., 2020, 2021, Wilcox et al., 2021], and introducing auxiliary losses to incorporate demonstration data into policy updates [Nair et al., 2018, Hester et al., 2018, Gao et al., 2018]. While these algorithms have shown impressive performance in improving the sample efficiency of RL algorithms, they add significant complexity and hyperparameters, making them difficult to implement and tune for different tasks.

We present Monte Carlo augmented Actor-Critic (MCAC), which introduces an easy-to-implement, but highly effective, change that can be readily applied to existing actor-critic algorithms without the introduction of any additional hyperparameters and only minimal additional complexity. The idea is to encourage initial optimism in the neighborhood of successful trajectories, and progressively reduce this optimism during learning so that it can continue to explore new behaviors. To operationalize this idea, MCAC introduces two modifications to existing actor-critic algorithms. First, MCAC initializes the replay buffer with task demonstrations. Second, MCAC computes a modified target Q-value for critic updates by taking the maximum of the standard temporal distance targets used in existing actor critic algorithms and a Monte Carlo estimate of the reward-to-go. The intuition is that Monte Carlo value estimates can more effectively capture longer-term reward information, making it possible to rapidly propagate reward information from demonstrations through the learned Q-function. This makes it possible to prevent underestimation of values in high-performing trajectories early in learning, as high rewards obtained later in a trajectory may be difficult to initially propagate back to earlier states with purely temporal distance targets [Wright et al., 2013]. Experiments on five continuous control domains suggest that MCAC is able to substantially accelerate exploration for both standard RL algorithms and recent RL from demonstrations algorithms in sparse reward tasks.

2 Related Work

2.1 Reinforcement Learning from Demonstrations

One standard approach for using demonstrations for RL first uses imitation learning [Argall et al., 2009] to pre-train a policy, and then fine-tunes this policy with on-policy reinforcement learning algorithms [Schaal et al., 1997, Kober and Peters, 2014, Peng et al., 2019, Rajeswaran et al., 2018]. However, initializing with suboptimal demonstrations can hinder learning and using demonstrations to initialize only a policy is inefficient, since they can also be used for Q-value estimation.

Other approaches leverage demonstrations to explicitly constrain agent exploration. Thananjeyan et al. [2020] and Wilcox et al. [2021] propose model-based RL approaches that uses suboptimal demonstrations to iteratively improve performance by ensuring consistent task completion during learning. Similarly, Jing et al. [2020] also uses suboptimal demonstrations to formulate a soft-constraint on exploration. However, a challenge with these approaches is that they introduce substantial algorithm complexity, making it difficult to tune and utilize these algorithms in practice. For example, while Thananjeyan et al. [2020] does enable iterative improvement upon suboptimal demonstrations, it requires learning a model of system dynamics and a density estimator to capture the support of successful trajectories making it challenging to scale to high-dimensional observations.

Finally, many methods introduce auxiliary losses to incorporate demonstrations into policy updates [Kim et al., 2013, Gao et al., 2018, Kang et al., 2018]. Deep Deterministic Policy Gradients from Demonstrations (DDPGfD) [Vecerik et al., 2018] maintains all the demonstrations in a separate replay buffer and uses prioritized replay to allow reward information to propagate more efficiently. Nair et al. [2018], and Hester et al. [2018] use similar approaches, where demonstrations are maintained separately from the standard replay buffer and additional policy losses encourage imitating the behavior in the demonstrations. Meanwhile, RL algorithms such as AWAC [Nair et al., 2021] pretrain using demonstration data to constrain the distribution of actions selected during online exploration. While these methods often work well in practice, they often increase algorithmic complexity and introduce several additional hyperparameters that are difficult and time consuming to tune. By

contrast, MCAC does not increase algorithmic complexity, is parameter-free, and can easily be wrapped around any existing actor-critic algorithm.

2.2 Improving Q-Value Estimates

The core contribution of this work is an easy-to-implement, yet highly effective, method for stabilizing actor-critic methods for sparse reward tasks using demonstrations and an augmented Q-value target. There has been substantial literature investigating learning stability challenges in off-policy deep Q-learning and actor-critic algorithms. See Van Hasselt et al. [2018] for a more thorough treatment of the learning stability challenges introduced by combining function approximation, bootstrapping, and off-policy learning, as well as prior work focused on mitigating these issues.

One class of approaches focuses on developing new ways to compute target Q-values [Konidaris et al., 2011, Wright et al., 2013, Van Hasselt et al., 2016, Schulman et al., 2016]. Van Hasselt et al. [2016] computes target Q-values with two Q-networks, using one to select actions and the other to measure the value of selected actions, which helps to prevent the Q-value over-estimation commonly observed in practice in many practical applications of Q-learning. TD3 [Fujimoto et al., 2018] attempts to address overestimation errors by taking the minimum of two separate Q-value estimates, but this can result in underestimation of the true Q-value target. Kuznetsov et al. [2020] uses an ensemble of critics to adaptively address estimation errors in Q-value targets, but introduces a number of hyperparameters which must be tuned separately for different tasks. Bellemare et al. [2016] and Ostrovski et al. [2017] consider a linear combination of a TD-1 target and Monte Carlo target. Wright et al. [2015], Schulman et al. [2016] and Sharma et al. [2018] consider a number of different estimators of a policy's value via n-step returns, which compute Q-targets using trajectories with n contiguous transitions followed by a terminal evaluation of the Q-value after n steps. Each of these targets make different bias and variance tradeoffs that can affect learning dynamics.

Similar to our work, Wright et al. [2013] explore the idea of taking a maximum over a bootstrapped target Q-value (TD-1 target) and a Monte Carlo estimate of the return-to-go to improve fitted Q-iteration. However, Wright et al. [2013] focuses only on fully offline Q-learning and only considers simple low-dimensional control tasks using Q-learning with linear value approximation. There are numerous reasons to extend these ideas to online deep RL. First, deep RL algorithms are often unstable, and using the ideas in Wright et al. [2013] to improve Q estimates is a promising way to alleviate this, as we empirically verify. Second, while offline learning has many important applications, online RL is far more widely studied, and we believe it is useful to study the effects these ideas have in this setting. To the best of our knowledge, MCAC is the first application of these ideas to online actor-critic algorithms with deep function approximation, and we find that it yields surprising improvements in RL performance on complex high-dimensional continuous control tasks.

3 Problem Statement

We consider a Markov Decision Process (MDP) described by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, T)$ with a state set $\mathcal{S}$, an action set $\mathcal{A}$, a transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a discount factor γ, and finite time horizon T. In each state $s_t \in \mathcal{S}$ the agent chooses an action $a_t \in \mathcal{A}$ and observes the next state $s_{t+1} \sim p(\cdot|s_t, a_t)$ and a reward $r(s_t, a_t) \in \mathbb{R}$. The agent acts according to a policy π, which induces a probability distribution over $\mathcal{A}$ given the state, $\pi(a_t|s_t)$. The agent's goal is to find the policy π^* which at any given $s_t \in \mathcal{S}$ maximizes the expected discounted sum of rewards,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right], \tag{3.1}$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots s_T)$ and $\tau \sim \pi$ indicates the distribution of trajectories induced by evaluating policy π in the MDP.

In this work, we make additional assumptions specific to the class of problems we study. First, we assume that all transitions in the replay buffer are elements of complete trajectories; this is a reasonable assumption as long as all transitions are collected from rolling out some policy in the MDP. Second, we assume the agent has access to an offline dataset $\mathcal{D}_{\text{offline}}$ of (possibly suboptimal) task demonstrations. Finally, we focus on settings where the reward function is sparse in the sense that

most state transitions do not induce a change in reward, causing significant exploration challenges for reinforcement learning algorithms.

4 Preliminaries: Actor-Critic Algorithms

For a given policy π, its state-action value function Q^π is defined as

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k)\right]. \tag{4.1}$$

Actor-critic algorithms learn a sample-based approximation to Q^π, denoted Q^π_θ, and a policy π_ϕ which selects actions to maximize Q^π_θ, with a function approximator (typically a neural network) parameterized by θ and ϕ respectively. During the learning process, they alternate between regressing Q^π_θ to predict Q^π and optimizing π_ϕ to select actions with high values under Q^π_θ.

Exactly computing Q^π targets to train Q^π_θ is typically intractable for arbitrary policies in continuous MDPs, motivating other methods for estimating them. One such method is to simply collect trajectories $(s_t, a_t, r_t, s_{t+1}, \ldots s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ by executing the learned policy π_ϕ from state s_t, computing a Monte Carlo estimate of the reward-to-go defined as follows:

$$Q^{\text{target}}_{\text{MC}}(s_t, a_t) = \sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k), \tag{4.2}$$

and fitting Q^π_θ to these targets.

However, $Q^{\text{target}}_{\text{MC}}$ can be a high variance estimator of the reward-to-go [Schulman et al., 2016, Sutton and Barto, 2018], motivating the popular one-step temporal difference target (TD-1 target) to help stabilize learning:

$$Q^{\text{target}}_{\text{TD}}(s_t, a_t) = r(s_t, a_t) + \gamma Q^\pi_{\theta'}(s_{t+1}, a_{t+1}), \tag{4.3}$$

where $a_{t+1} \sim \pi_\phi(s_{t+1})$, which is recursively defined based on only a single (s_t, a_t, s_{t+1}, r_t) transition. Here θ' gives the parameters of a lagged target network as in [Van Hasselt et al., 2016]. There has also been recent interest in methods which computing TD-n targets, which instead sum up rewards for the next n steps and then incorporate Q-values from step $n+1$ [Wright et al., 2015, Schulman et al., 2016, Sharma et al., 2018].

5 Monte Carlo augmented Actor-Critic

5.1 MCAC Algorithm

The objective of MCAC is to efficiently convey information about sparse rewards from suboptimal demonstrations to Q^π_θ in order to accelerate policy learning while still maintaining learning stability. To do this, MCAC combines two different methods of computing targets for fitting Q-functions to enable efficient value propagation throughout the state-action space while also learning a Q-value estimator with low enough variance for stable learning. To operationalize this idea, MCAC defines a new Q-function target for training Q^π_θ by taking the maximum of the Monte Carlo Q-target (eq 4.2) and the temporal difference Q-target (eq 4.3): $\max\left[Q^{\text{target}}_{\text{TD}}(s_t, a_t), Q^{\text{target}}_{\text{MC}}(s_t, a_t)\right]$.

The idea here is that early in learning, a Q-function trained only with temporal difference targets will have very low values throughout the state-action space as it may be very difficult to propagate information about delayed rewards through the temporal difference target for long-horizon tasks with sparse rewards [Van Hasselt et al., 2018]. On the other hand, the Monte Carlo Q-target can easily capture long-term rewards, but often dramatically underestimates Q-values for poorly performing trajectories [Wright et al., 2013]. Thus, taking a maximum over these two targets serves to initially boost the Q-values for transitions near high performing trajectories while limiting the influence of underestimates from the Monte Carlo estimate.

MCAC can also be viewed as a convenient way to balance the bias and variance properties of the Monte Carlo and temporal difference Q-targets. The Monte Carlo Q-target is well known to be an unbiased estimator of policy return, but have high variance [Wright et al., 2013]. Conversely,

Algorithm 1 Monte Carlo augmented Actor-Critic

Require: Offline dataset $\mathcal{D}_{\text{offline}}$.
Require: Total training episodes N, batch size M, Pretraining Steps N_p
Require: Episode time horizon T.
Initialize replay buffer $\mathcal{R} := \mathcal{D}_{\text{offline}}$.
Initialize agent π_ϕ and critic Q^π_θ using data from $\mathcal{D}_{\text{offline}}$.
for $i \in \{1, \ldots, N_p\}$ **do**
    Sample $\mathcal{B} \subsetneq \mathcal{R}$ such that $|\mathcal{B}| = M$.
    Optimize Q^π_θ on $\mathcal{B}$ to minimize loss in eq (5.4). Optimize policy π_ϕ to maximize Q_θ.
end for
for $i \in \{1, \ldots, N\}$ **do**
    Initialize episode buffer $\mathcal{E} = \{\}$.
    Observe state s^i_1.
    for $j \in \{1, \ldots, T\}$ **do**
        Sample and execute $a^i_t \sim \pi_\theta(s^i_j)$, observing s^i_{j+1}, r^i_j.
        $\tau^i_j \leftarrow (s^i_j, a^i_j, s^i_{j+1}, r^i_j)$
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\tau^i_j\}$.
        Sample $\mathcal{B} \subsetneq \mathcal{R}$ such that $|\mathcal{B}| = M$.
        Optimize Q^π_θ on $\mathcal{B}$ to minimize loss in eq (5.4). Optimize policy π_ϕ to maximize Q_θ.
    end for
    for $\tau^i_j \in \mathcal{E}$ **do**
        Compute $Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j)$ as in eq (5.2).
        $\tau^i_j \leftarrow (s^i_j, a^i_j, s^i_{j+1}, r^i_j, Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j))$.
    end for
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{E}$.
end for

temporal difference targets are known to be biased, but have much lower variance [Wright et al., 2013, Van Hasselt et al., 2016]. Thus, as the temporal difference target is typically negatively biased (an underestimate of the true policy return) on successful trajectories early in learning due to the challenge of effective value propagation, computing the maximum of the temporal difference and Monte Carlo targets helps push the MCAC target value closer to the true policy return. Conversely, the Monte Carlo targets are typically pessimistic on unsuccessful trajectories, and their high variance makes it difficult for them to generalize sufficiently to distinguish between unsuccessful trajectories that are close to being successful and those that are not. Thus, computing the maximum of the temporal difference and Monte Carlo targets also helps to prevent excessive pessimism in evaluating unsuccessful trajectories. Notably, MCAC does not constrain its Q-targets explicitly based on the transitions in the demonstrations, making it possible for the policy to discover higher performing behaviors than those in the demonstrations as demonstrated in Section 6.

5.2 MCAC Practical Implementation

MCAC can be implemented as a wrapper around any actor-critic RL algorithm; we consider 5 options in experiments (Section 6). MCAC starts with an offline dataset of suboptimal demonstrations $\mathcal{D}_{\text{offline}}$, which are used to initialize a replay buffer $\mathcal{R}$. Then, during each episode i, we collect a full trajectory τ^i, where the jth transition $(s^i_j, a^i_j, s^i_{j+1}, r^i_j)$ in τ^i is denoted by τ^i_j.

Next, consider any actor-critic method using a learned Q function approximator $Q_\theta(s_t, a_t)$ that, for a given transition $\tau^i_j = (s^i_j, a^i_j, s^i_{j+1}, r^i_j) \in \tau^i \subsetneq \mathcal{R}$, would be updated by minimizing a loss,

$$J(\theta) = \ell\left(Q_\theta(s^i_j, a^i_j), Q^{\text{target}}(s^i_j, a^i_j)\right), \tag{5.1}$$

where ℓ is an arbitrary differentiable loss function and Q^{target} is the target value for regressing Q_θ. We note that Q^{target} is defined by the choice of actor-critic method. To implement MCAC, we first calibrate with temporal difference targets (which provide infinite-horizon Q estimates) by computing the infinite horizon analogue of the Monte Carlo target defined in Equation 4.2, which assumes the last observed reward value will repeat forever and uses this to add an infinite sum of discounted

rewards, and is given by

$$Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j) = \gamma^{T-j+1}\frac{r^i_T}{1-\gamma} + \sum_{k=j}^{T}\gamma^{k-j} r(s^i_k, a^i_k). \tag{5.2}$$

Then, we simply replace the target with a maximum over the original target and the Monte Carlo target defined in Equation 5.2, given by

$$Q^{\text{target}}_{\text{MCAC}}(s^i_j, a^i_j) = \max\left[Q^{\text{target}}(s^i_j, a^i_j), Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j)\right]. \tag{5.3}$$

This results in the following loss function for training Q_θ:

$$J(\theta) = \ell\left(Q_\theta(s^i_j, a^i_j), Q^{\text{target}}_{\text{MCAC}}(s^i_j, a^i_j)\right). \tag{5.4}$$

The full training procedure alternates between updating Q^π_θ using the method described above, followed by optimizing the policy π_ϕ using any standard policy update method. The full MCAC algorithm is described in Algorithm 1.

6 Experiments

In the following experiments we study (1) whether MCAC enables more efficient learning when built on top of standard actor-critic RL algorithms and (2) whether MCAC can be applied to improve prior algorithms for RL from demonstrations. See the supplementary material for code and instructions on how to run experiments for reproducing all results in the paper and additional experiments studying the impact of demonstration quantity, quality, and other algorithmic choices such as whether to pretrain learned networks on demonstration data before online interaction.

6.1 Experimental Procedure

All experiments were run on a set of 24 Tesla V100 GPUs through a combination of Google Cloud resources and a dedicated lab server. We aggregate statistics over 10 random seeds for all experiments, reporting the mean and standard error across the seeds with exponential smoothing. Details on hyperparameters and implementation details are provided in the supplementary material.

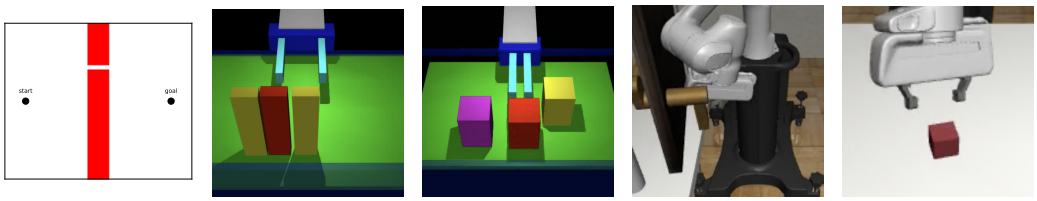
6.2 Domains

We consider the five long-horizon continuous control tasks shown in Figure 1. All tasks have relatively sparse rewards, making demonstrations critical for performance. We found that without demonstrations, SAC and TD3 made little to no progress on these tasks.

Pointmass Navigation: The first domain is a pointmass 2D navigation task (Figure 1(a)) with time horizon $T = 100$, where the objective is to navigate around the red barrier from start set $\mathcal{S}$ to a goal set $\mathcal{G}$ by executing 2D delta-position controls. If the agent collides with the barrier it receives a reward of -100 and the episode terminates. At each time step, the agent receives a reward of -1 if it is not in the goal set and 0 if it is in the goal set. To increase the difficulty of the task, we perturb the state with zero-mean Gaussian noise at each timestep. The combination of noisy transitions and sparse reward signal makes this a very difficult exploration task where the agent must learn to make it through the slit without converging to the local optima of avoiding both the barrier and the slit.

The demonstrator is implemented as a series of proportional controllers guiding it from the starting set to the slit, through the slit, and to the goal set. The actions are clipped to fit in the action space, and trajectories are nearly optimal. The agent is provided with 20 demonstrations.

Object Manipulation in MuJoCo: We next consider two object manipulation tasks designed in the MuJoCo physics simulator [Todorov et al., 2012], where the objective is to extract a block from a tight configuration on a table (Block Extraction, Figure 1(b)) and push each of 3 blocks forward on the plane (Sequential Pushing, Figure 1(c)). In the Block Extraction task, the action space consists of 3D delta position controls and an extra action dimension to control the degree to which the gripper is opened. In the Sequential Pushing environment, this extra action dimension is omitted and the



(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing (d) Door Opening (e) Block Lifting

Figure 1: We evaluate MCAC on five continuous control domains: a pointmass navigation environment, and four high-dimensional robotic control domains. All domains are associated with relatively unshaped reward functions, which only indicate constraint violation, task completion, or completion of a subtask.

gripper is kept closed. In the Block Extraction domain, the agent receives a reward of -1 for every timestep that it hasn't retrieved the red block and 0 when it has. In the Sequential Pushing domain, the reward increases by 1 for each block the agent pushes forward. Thus, the agent receives a reward of -3 when it has made no progress and 0 when it has completed the task. The Block Extraction task is adapted from Thananjeyan et al. [2021] while the Sequential Pushing task is adapted from Wilcox et al. [2021]. We use a time horizon of $T = 50$ for the Block Extraction task and a longer $T = 150$ for the Sequential Pushing task since it is of greater complexity.

The block extraction demonstrator is implemented as a series of proportional controllers guiding the arm to a position to grip the block, followed by an instruction to close the gripper and a controller to lift. We provide the agent with 50 demonstrations with zero-mean noise injected in the controls to induce suboptimality. For the sequential pushing environment, the demonstrator uses a series of proportional controllers to slowly push one block forward, move backwards, line up with the next block, and repeat the motion until all blocks have been pushed. Because it moves slowly and moves far back from each block it pushes, demonstrations are very suboptimal. For Sequential Pushing, the agent is provided with 500 demonstrations due to the increased difficulty of the task.

Robosuite Object Manipulation: Finally, we consider two object manipulation tasks built on top of Robosuite [Zhu et al., 2020], a collection of robot simulation tasks using the MuJoCo physics engine. We consider the Door Opening task (Figure 1(d)) and the Block Lifting task (Figure 1(e)). In the Door Opening task, a Panda robot with 7 DoF and a parallel-jaw gripper must turn the handle of a door in order to open it. The door's location is randomized at the start of each episode. The agent receives a reward of -1 if it has not opened the door and a reward of 0 if it has. In the Block Lifting task, the same Panda robot is placed in front of a table with a single block on its surface. The robot must pick up the block and lift it above a certain threshold height. The block's location is randomized for each episode and the agent receives a reward of -1 for every timestep it has not lifted the block and a reward of 0 when it has. Both Robosuite tasks use a time horizon of $T = 50$.

For both Robosuite tasks, demonstrators are trained using SAC on a version of the task with a hand-designed dense reward function, as in the Robosuite benchmarking experiments [Zhu et al., 2020]. In order to ensure suboptimality, we stop training the demonstrator policy before convergence. For each Robosuite environment we use the trained demonstrator policies to generate 100 suboptimal demonstrations for training MCAC and the baselines.

6.3 Algorithm Comparisons

We empirically evaluate the following baselines both individually and in combination with MCAC. All methods are provided with the same demonstrations which are collected as described in Section 6.2. See the supplement for more in depth details on implementation and training.

Behavior Cloning: Direct supervised learning on the offline suboptimal demonstrations.

Twin Delayed Deep Deterministic Policy Gradients (TD3) [Fujimoto et al., 2018]: State of the art actor-critic algorithm which trains a deterministic policy to maximize a learned critic.

Soft Actor-Critic (SAC) [Haarnoja et al., 2018]: State of the art actor-critic algorithm which trains a stochastic policy which maximizes a combination of the Q value of the policy and the expected entropy of the policy to encourage exploration.

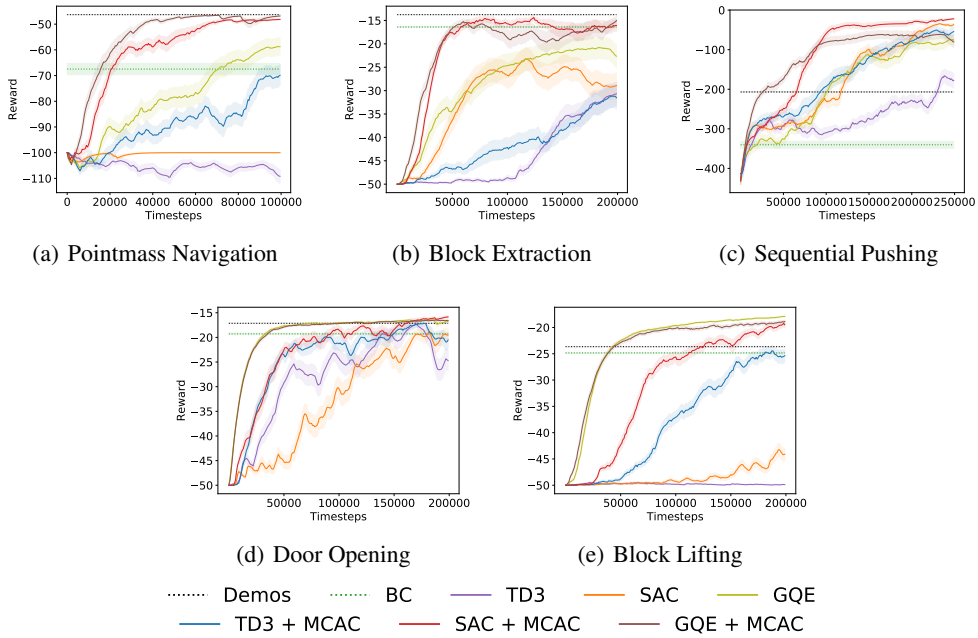


Figure 2: **MCAC and Standard RL Algorithms Results:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds. We find that MCAC improves the learning efficiency of TD3, SAC, and GQE across all 5 environments.

Generalized Q Estimation (GQE): A complex return estimation method from Schulman et al. [2016] for actor-critic methods, which computes a weighted average over TD-i estimates for a range of i. GQE is implemented on top of SAC (see supplement for more details). We tune the range of horizons considered and the eligibility trace value (corresponding to λ in GAE).

Overcoming Exploration from Demonstrations (OEFD) [Nair et al., 2018]: OEFD builds on DDPG [Lillicrap et al., 2015] by adding a loss which encourages imitating demonstrations and a learned filter which determines when to activate this loss. Our implementation does include hindsight experience replay since it is not applicable to most of our environments.

Conservative Q Learning (CQL) [Kumar et al., 2020]: A recent offline reinforcement learning algorithm that addresses value overestimation with a conservative Q function.

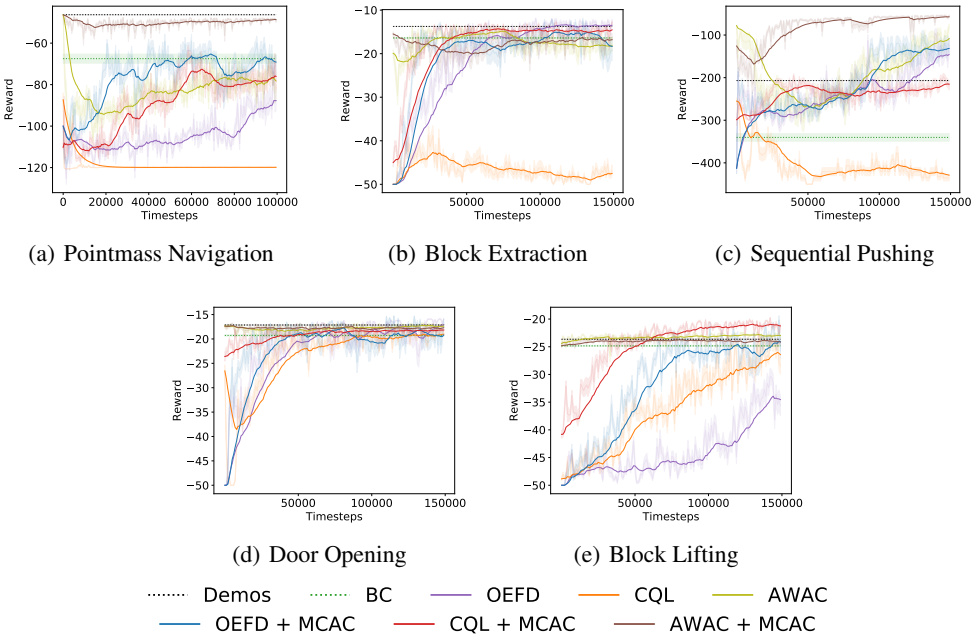
Advantage Weighted Actor-Critic (AWAC) [Nair et al., 2021]: A recent offline reinforcement learning algorithm designed for fast online fine-tuning.

We also implement versions of each of the above RL algorithms with MCAC (TD3 + MCAC, SAC + MCAC, GQE + MCAC, OEFD + MCAC, CQL + MCAC, AWAC + MCAC).

The behavior cloning comparison serves to determine whether online learning is beneficial in general, while the other comparisons study whether MCAC can be used to accelerate reinforcement learning for commonly used actor-critic algorithms (TD3, SAC, and GQE, which is essentially SAC with complex returns) and for recent algorithms for RL from demonstrations (OEFD, CQL and AWAC).

6.4 Results

In Section 6.4.1 we study whether MCAC enables more efficient learning when built on top of SAC and TD3, which are widely used actor-critic RL algorithms, as well as GQE, a complex return method. In Section 6.4.2 we study whether MCAC can provide similar benefits when applied to recent RL from demonstration algorithms (OEFD, CQL and AWAC). Additionally, in the supplement we provide experiments involving other baselines, investigating the sensitivity of MCAC to the quality and quantity of demonstration data, and investigating its sensitivity to pretraining.



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

(d) Door Opening (e) Block Lifting

Figure 3: **MCAC and RL from Demonstrations Algorithm Results:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds. When OEFD or AWAC achieve high performance almost immediately, MCAC has little impact on performance. However, when OEFD and AWAC are unable to learn efficiently, MCAC accelerates and stabilizes policy learning.

6.4.1 MCAC and Standard RL Algorithms

In Figure 2, we study the impact of augmenting SAC, TD3 and GQE with the MCAC target Q-function. Note that all methods, both with and without MCAC, we initialize their replay buffers with the same set of demonstrations. Results suggest that MCAC is able to accelerate learning for both TD3 and SAC across all environments, and is able to converge to performance either on-par with or better than the demonstrations. In the Pointmass Navigation and Block Lifting tasks, SAC and TD3 make no task progress without MCAC. MCAC also accelerates learning for GQE for the Pointmass Navigation, Block Extraction, and Sequential Pushing environments. In the Door Opening and Block Lifting environments, MCAC leaves performance largely unchanged since GQE already achieves performance on par with the next best algorithm without MCAC.

6.4.2 MCAC and RL From Demonstrations Algorithms

In Figure 3, we study the impact of augmenting OEFD [Nair et al., 2018], CQL [Kumar et al., 2020] and AWAC [Nair et al., 2021] with the MCAC target Q-function. Results suggest that MCAC improves the learning efficiency of OEFD on the Pointmass Navigation, Sequential Pushing, and Block Lifting tasks, but does not have a significant positive or negative affect on performance for the Block Extraction and Door Opening tasks. MCAC improves the performance of AWAC on the Pointmass Navigation and Sequential Pushing environments, stabilizing learning while the versions without MCAC see performance fall off during online fine tuning. On the other 3 environments where AWAC is able to immediately converge to a stable policy after offline pre-training, MCAC has no significant negative effect on its performance. In all tasks, MCAC improves the performance of CQL. In particular, for the Pointmass Navigation, Block Extraction and Sequential Pushing tasks, CQL makes almost no progress while the version with MCAC learns to complete the task reliably.

6.5 Replay Buffer Study

In order to better understand the way MCAC affects Q estimates, we visualize Q estimates on the entire replay buffer, including offline demonstrations, after 50000 timesteps of training in the Pointmass Navigation environment, shown in Figure 4.

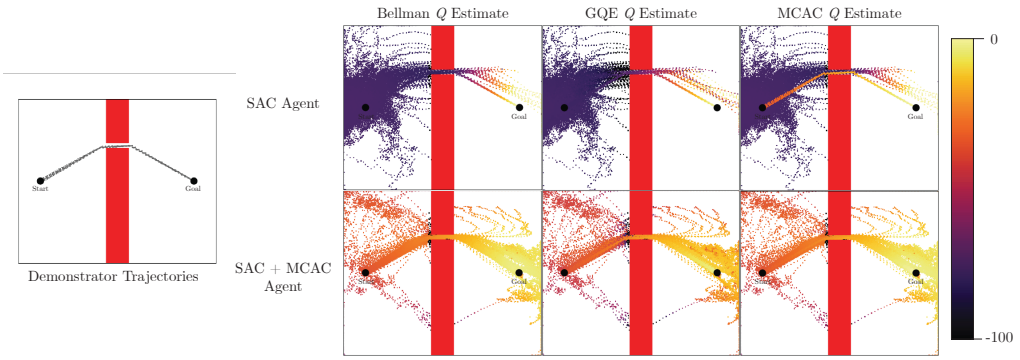


Figure 4: Scatter plots showing Bellman, GQE and MCAC Q estimates on the entire replay buffer, including offline demonstrations, for SAC learners with and without the MCAC modification after 50000 timesteps of training. The top row shows data and Q estimates obtained while training a baseline SAC agent, while the bottom row shows the same with an MCAC agent. The left column shows Bellman Q estimates on each replay buffer sample while the middle column shows GQE estimates and the right column shows MCAC estimates. Results suggest that MCAC is helpful for propagating rewards along demonstrator trajectories.

We see that in the top row, when training without MCAC, the learner is never able to learn a useful Q function and thus does not learn to complete the task, where the only successful trajectories shown are offline data. However, even when this is the case the MCAC estimate is able to effectively propagate reward signal backwards along the demonstrator trajectories, predicting higher rewards early on (top right). We see that the GQE estimates (top middle) are somewhat more effective than the Bellman ones at propagating reward, but not as effective as MCAC. On the bottom row, where the agent was trained with MCAC, we see (bottom left) that the agent learns a useful Q function that it uses to reliably complete the task. As expected with a high-performing policy, its Bellman estimates, GQE estimates and MCAC estimates are similar.

7 Discussion, Limitations, and Ethical Considerations

We present Monte Carlo augmented Actor-Critic (MCAC), a simple, yet highly effective, change that can be applied to any actor-critic algorithm in order to improve reinforcement learning from demonstrations. We present empirical results suggesting that MCAC often significantly improves, and does not significantly degrade, performance when applied to two state-of-the-art actor-critic RL algorithms and two RL from demonstrations algorithms on five different continuous control domains.

Despite strong empirical performance, MCAC also has some limitations. We found that while encouraging higher Q-value estimates is beneficial for sparse reward tasks, when rewards are dense and richly informative, MCAC is not helpful and can even hinder learning by overestimating Q-values. From an ethical standpoint, reinforcement learning algorithms such as MCAC can be used to automate aspects of a variety of interactive systems, such as robots, online retail systems, or social media platforms. However, poor performance when policies haven't yet converged, or when hand-designed reward functions do not align with true human intentions, has the potential to cause significant harm to the humans using these systems. Before deploying RL algorithms such as MCAC in real-world settings, it is critical to ensure that appropriate precautions are taken to minimize harm and damage. Lastly, while MCAC shows convincing empirical performance, in future work it would be helpful to provide theoretical analysis on its convergence properties and its effect on the bias-variance tradeoff.

Acknowledgements

We would like to thank all reviewers for invaluable commentary which helped us to strengthen the paper during the rebuttal period. This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, and the CITRIS "People and Robots" (CPAR) Initiative. The authors were supported in part by donations from Google, Toyota Research Institute, and NVIDIA.

References

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Sebastian Dittert. Cql. `https://github.com/BY571/CQL`, 2021.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pages 1582–1591, 2018.

Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Mingxuan Jing, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Chao Yang, Bin Fang, and Huaping Liu. Reinforcement learning from imperfect demonstrations under soft expert guidance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5109–5116, 2020.

Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International Conference on Machine Learning*, pages 2469–2478. PMLR, 2018.

Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Learning from limited demonstrations. In *NIPS*, pages 2859–2867. Citeseer, 2013.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Jens Kober and Jan Peters. Policy search for motor primitives in robotics. In *Learning Motor Skills*, pages 83–117. Springer, 2014.

George Konidaris, Scott Niekum, and Philip S Thomas. Td_gamma: Re-evaluating complex backups in temporal difference learning. *Advances in Neural Information Processing Systems*, 24:2402–2410, 2011.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of ai ingenuity. *DeepMind Blog*, 2020.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1179–1191. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf`.

Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pages 5556–5566. PMLR, 2020.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL `http://arxiv.org/abs/1509.02971`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets, 2021.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019.

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, 2018.

Stefan Schaal et al. Learning from demonstration. *Advances in neural information processing systems*, pages 1040–1046, 1997.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*, 06 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Sahil Sharma, Girish Raguvir J, Srivatsan Ramesh, and Balaraman Ravindran. Learning to mix n-step returns: Generalizing lambda-returns for deep reinforcement learning. 2018.

Harshit Sikchi and Albert Wilcox. pytorch-AWAC. URL `https://github.com/hari-sikchi/AWAC`.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Pranjal Tandon. pytorch-soft-actor-critic. URL `https://github.com/pranz24/pytorch-soft-actor-critic`.

Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E. Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks, 2020.

Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3): 4915–4922, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad, 2018.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, 2018.

Albert Wilcox, Ashwin Balakrishna, Brijen Thananjeyan, Joseph E. Gonzalez, and Ken Goldberg. Ls3: Latent space safe sets for long-horizon visuomotor control of sparse reward iterative tasks. In *Conference on Robot Learning (CoRL)*. PMLR, 2021.

Robert Wright, Steven Loscalzo, Philip Dexter, and Lei Yu. Exploiting multi-step sample trajectories for approximate value iteration. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 113–128, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40988-2.

Robert William Wright, Xingye Qiao, Lei Yu, and Steven Loscalzo. Cfqi: Fitted q-iteration with complex returns. In *AAMAS*, pages 163–170. Citeseer, 2015.

Zheng Wu, Wenzhao Lian, Vaibhav Unhelkar, Masayoshi Tomizuka, and Stefan Schaal. Learning dense rewards for contact-rich manipulation tasks. *arXiv preprint arXiv:2011.08458*, 2020.

Jiexin Xie, Zhenzhou Shao, Yue Li, Yong Guan, and Jindong Tan. Deep reinforcement learning with optimized reward functions for robotic trajectory planning. *IEEE Access*, 7:105669–105679, 2019.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The Abstract and introduction claim that we introduce a new reinforcement learning algorithm and experiments on a number of domains suggesting that it accelerates learning for sparse reward tasks. This is consistent with the algorithm presented in Section 5 and the experiments in Figures 2 and 3 in Section 6
   (b) Did you describe the limitations of your work? [Yes] Yes, see Section 7
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Yes, see Section 7
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Yes, this work conforms to the ethics guidelines in the sense that it does not have direct negative societal impacts that we know of and is fully reproducible, with released code and results reported over 10 random seeds. That being said, reinforcement learning research in general, including our work, has possible ethical concerns related to misuse in systems which interact with humans, such as robots, online retail systems, and social media platforms. Specifically, poorly specified reward functions can cause unexpected consequences when algorithms discover behaviors different from what the human reward designer intended. Suboptimal behaviors during policy optimization also have potential to cause harm. We discuss these concerns in detail in Section 7.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] There are no theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A] There are no theoretical results.
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] A zip file containing all code is included in the supplementary material. This code is associated with a README that can be used to replicate all experiments.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All training details are described at a high level in Section 6 and in detail in the supplementary material for all experiments.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All results in all plots are reported over 10 random seeds, with the mean displayed in the plot with standard error shading to indicate the margin of error over the 10 runs.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Yes, see the beginning of Section 6.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] Yes, the sources of all experimental domains used are cited in Section 6.2. We also cite the sources of all algorithms we build on in Section 6.3 and reference implementations used of these algorithms in the supplement.

(b) Did you mention the license of the assets? [N/A] There were no explicit licences associated with the datasets/code we used, but we cited the papers they came from and the Github repositories associated with them.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes, we have included code and datasets to reproduce all results in this paper in the supplement.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All data was collected in computational simulations, so no explicit human permission was required.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No data collected was explicitly collected by human users, so this was not necessary.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] There were no human subject studies.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] There were no human subject studies.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] There were no human subject studies.

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 [Dittert, 2021] for all environments, and pretrained 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.

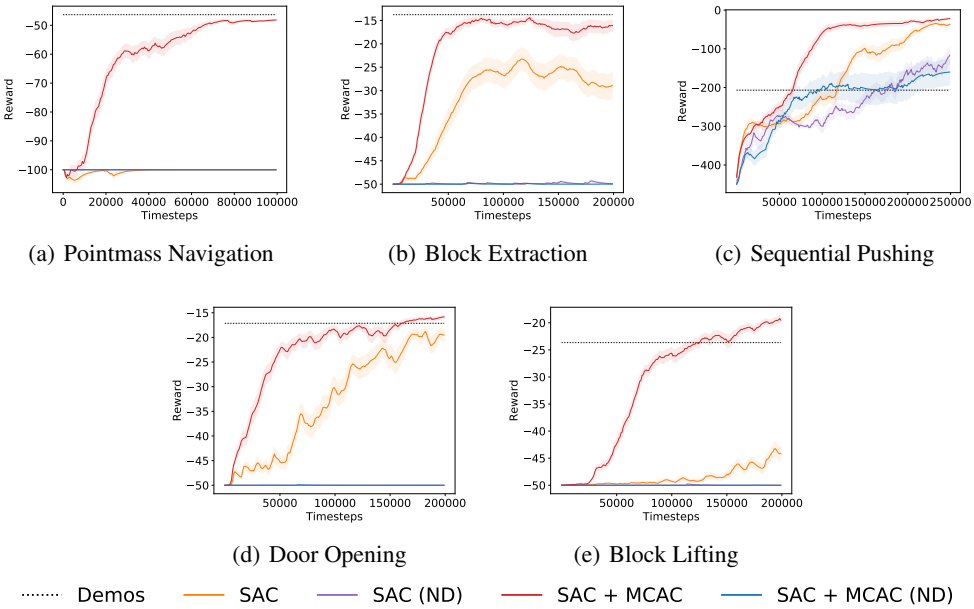
B Additional Experiments

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

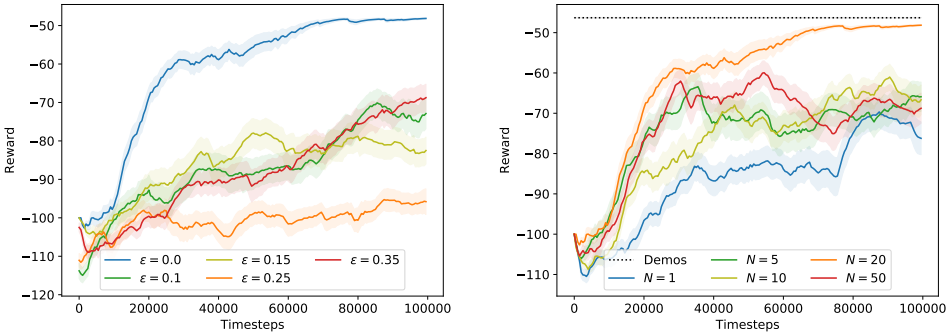
In Figure 6, we first study the impact of demonstration quality (Figure 6(a)) and quantity (Figure 6(b)) on MCAC when applied to SAC (SAC + MCAC) on the Pointmass Navigation domain. We evaluate sensitivity to demonstration quality by injecting ϵ-greedy noise into the demonstrator for the Pointmass Navigation 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.

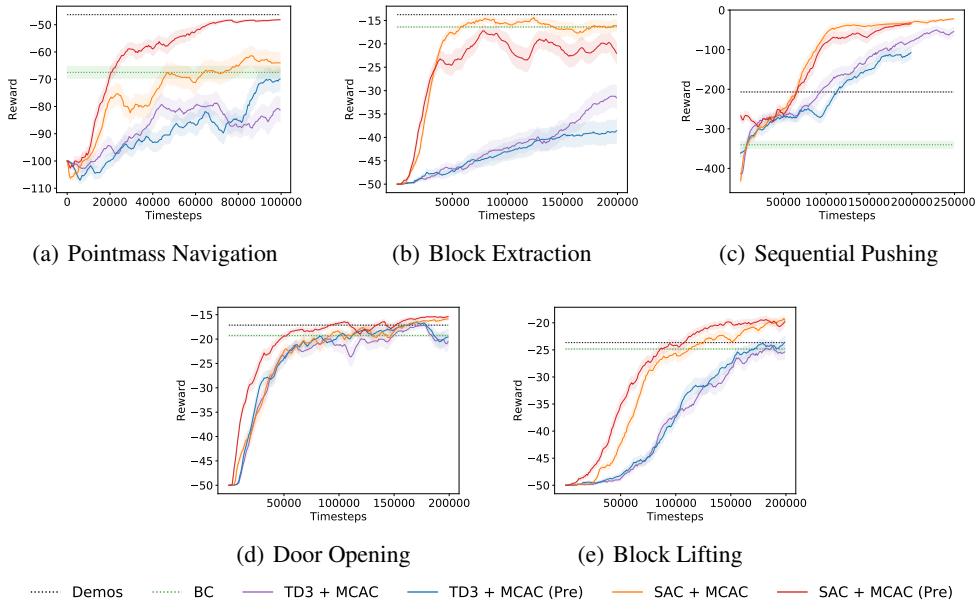


(a) Demonstration Quality (b) Demonstration Quantity

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 $Q_\lambda^{\text{target}}$ given by

$$Q_\lambda^{\text{target}} = (1-\lambda)Q^{\text{target}}(s_j^i, a_j^i) + \lambda Q_{\text{MC-}\infty}^{\text{target}}(s_j^i, a_j^i). \tag{B.1}$$

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

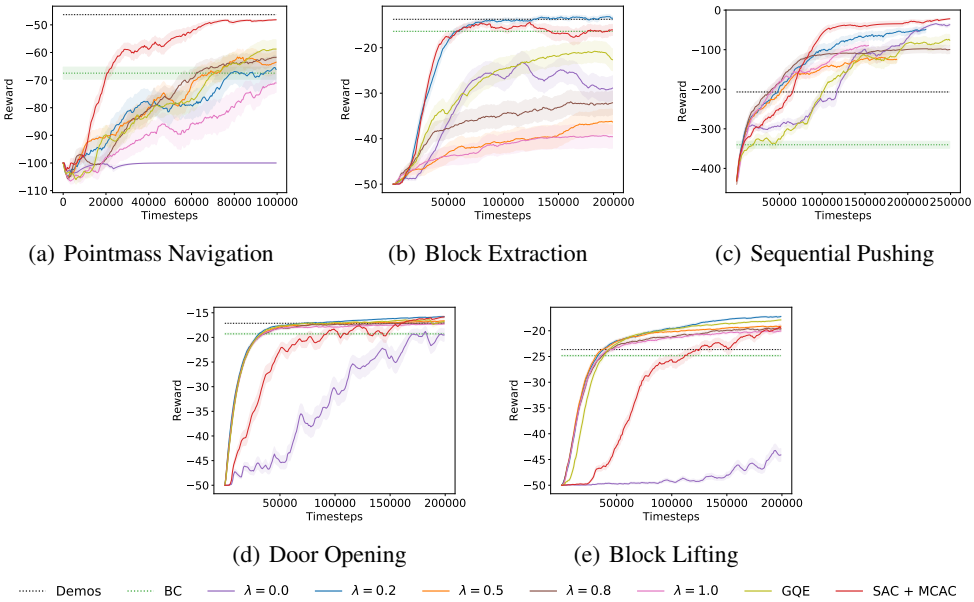
In this experiment, we consider replacing $Q_{\text{MC-}\infty}^{\text{target}}$ defined in the main text with a new target, which replaces the infinite geometric sum with the critic's Q value estimate. Formally, for a trajectory given by $[(s_i, a_t, r_i), (s_{i+1}, a_{i+1}, r_{i+1}), \ldots, (s_T,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

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

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 $Q_{\text{MC-}\infty}^{\text{target}}$ 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

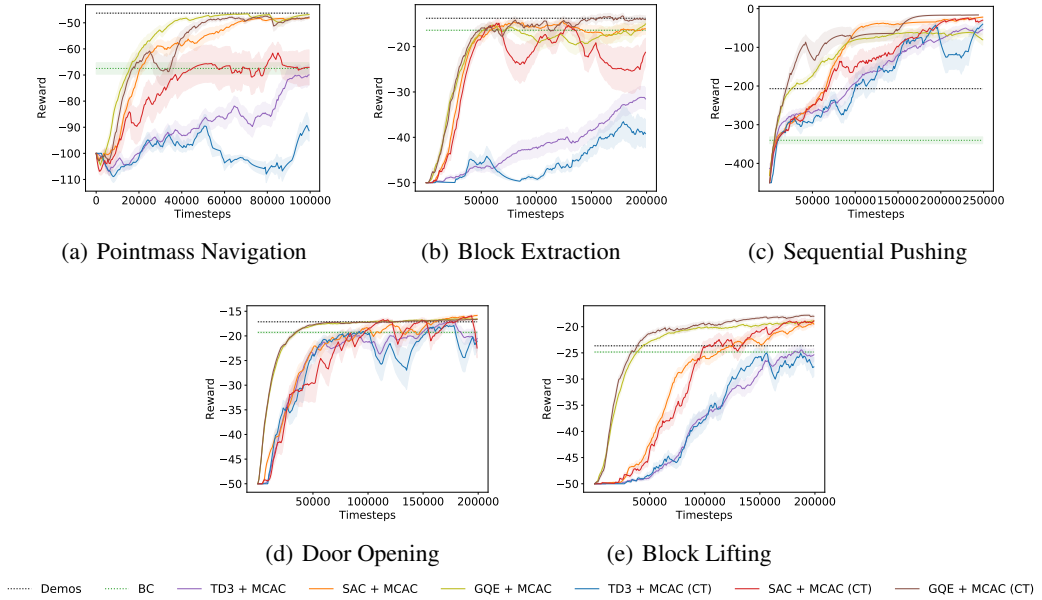
In this experiment we investigate how the scales of the various Q value estimates we consider vary as the agent trains. In Figure 10(a) we present average values for Q estimates based on samples from the replay buffer for successful trajectories, while in Figure 10(b) we present the same data for failed 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.

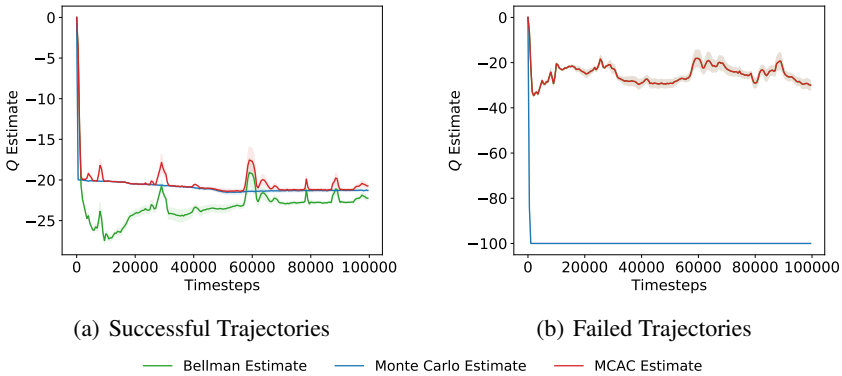


(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing

(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.

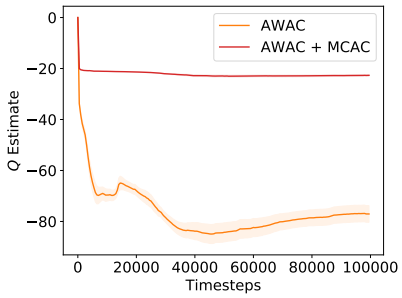


Figure 11: **AWAC Q Estimates:** Plots show mean and standard error across five random seeds of the Q estimates from the AWAC and AWAC+MCAC learners. At the 0th timestep, each algorithm has undergone a pretraining procedure as described in Section A.7. We see that while the original Q estimate plummets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