
Automaton Constrained Q-Learning Appendix

Anonymous Author(s)

Affiliation

Address

email

1 A Proof of Proposition 1

2 In what follows, we formally prove and restate Proposition 1, which shows that Automaton Con-
3 strained Q-Learning (ACQL), under mild conditions, is guaranteed to return the optimal policy. Our
4 proof is based on the proof of convergence for Q-learning using stochastic approximation theory in
5 [1], extended with the theory of stochastic approximation under multiple timescales described in
6 Chapter 6 of [2]. Refer also to [3–5] for similar analyses. The outline of the section is as follows:

- 7 1. Express ACQL as a stochastic approximation algorithm [1, 2].
- 8 2. Show that Q^c and Q^r converge to an optimal fixed point for any fixed safety discount factor
9 $\gamma_c \in (0, 1)$.
- 10 3. Restate our Proposition 1 and prove it by showing that as $\gamma_c \rightarrow 1$, Q^r and Q^c converge to
11 the optimal state-action value function Q^{r*} and its corresponding optimal (undiscounted)
12 state-action safety function Q^{c*} .

13 A.1 Setup

14 **Assumption 1.** *The augmented Constrained Markov Decision Process (CMDP) $\mathcal{M}^A =$
15 $(\mathcal{S}^A, \mathcal{A}, \mathcal{T}^A, d_0^A, r^A, c^A, \gamma, L)$ for ACQL is defined on a finite state space \mathcal{S}^A and action space
16 \mathcal{A} . For every state $s \in \mathcal{S}^A$ and action $a \in \mathcal{A}$, there is an associated bounded deterministic reward
17 $r_{sa} = r^A(s, a)$ and bounded constraint feedback $c_{sa} = c^A(s, a)$ observed if action a is applied at
18 state s .*

19 Under Assumption 1, Q^c and Q^r are vectors in \mathbb{R}^d where $d = |\mathcal{S}^A \times \mathcal{A}|$ is finite. The ACQL
20 algorithm can be modeled as a distributed, asynchronous series of noisy updates to components of
21 Q^c and Q^r .

22 **Assumption 2.** *For each state-action pair $(s, a) \in \mathcal{S}^A \times \mathcal{A}$, there are an infinite number of updates
23 applied to the components $Q_{s,a}^r$ and $Q_{s,a}^c$.*

24 The updates to these components are given by

$$Q_{s,a}^c(n+1) = Q_{s,a}^c(n) + a(n) \left[\left((1 - \gamma_c)c_{sa} + \gamma_c \min\{c_{sa}, \bar{Q}_{s',\pi(s')}^c(n)\} \right) - Q_{s,a}^c(n) \right] \text{ and} \quad (1)$$

$$Q_{s,a}^r(n+1) = Q_{s,a}^r(n) + b(n) \left[(r_{sa} + \gamma \bar{Q}_{s',\pi(s')}^r(n)) - Q_{s,a}^r(n) \right], \quad (2)$$

25 where s' is a randomly sampled next state following the state s and action a . The elements of
26 $\bar{Q}^c(n)$ and $\bar{Q}^r(n)$ are potentially taken from older iterations $Q_{s,a}^c(\nu_{s,a}(n))$ and $Q_{s,a}^r(\nu_{s,a}(n))$ where
27 $\nu_{s,a}(n)$ is an integer satisfying $0 \leq \nu_{s,a}(n) \leq n$. Recall that the policy π here is defined as
28 $\arg \max_{a: \bar{Q}_{s,a}^c > L} \bar{Q}_{s,a}^r$ in terms of \bar{Q}^c and \bar{Q}^r . However, we assume that old information is eventually
29 discarded as $n \rightarrow \infty$.

30 **Assumption 3.** For all (s, a) , $\lim_{n \rightarrow \infty} \nu_{s,a}(n) = \infty$.

31 Assumption 3 is necessary to prove the convergence of distributed asynchronous stochastic approxi-
 32 mation algorithms using outdated values $(\bar{Q}^c$ and $\bar{Q}^r)$ [1, 6]. Using \bar{Q}^c is additionally necessary to
 33 define the operator in (4) such that a fixed policy π can be used to prove the contraction property in
 34 Lemma 1.

35 We also update γ_c infinitely often with

$$\gamma_c(n+1) = \gamma_c(n) + c(n) [(1 - \gamma_c(n)) - \gamma_c(n)]. \quad (3)$$

36 and assume that its updates are synchronized with the index n for the updates to the components of
 37 Q^c and Q^r .

38 **Assumption 4.** The step sizes $a(n)$, $b(n)$, and $c(n)$ for the above updates satisfy

$$\begin{aligned} \sum_{n=0}^{\infty} a(n) &= \sum_{n=0}^{\infty} b(n) = \sum_{n=0}^{\infty} c(n) = \infty \\ \sum_{n=0}^{\infty} a(n)^2, \sum_{n=0}^{\infty} b(n)^2, \sum_{n=0}^{\infty} c(n)^2 &< \infty, \end{aligned}$$

39 $b(n) \in o(a(n))$, and $c(n) \in o(b(n))$.

40 Now, let $g : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ and $h : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ be operators defined for each component
 41 (s, a) as

$$g_{s,a}(Q^r, Q^c, \gamma_c) = (1 - \gamma_c)c_{sa} + \gamma_c \mathbb{E}_{s'} \left[\min\{c_{sa}, Q_{s',\pi(s')}^c\} \right] \quad (4)$$

$$h_{s,a}(Q^r, Q^c, \gamma_c) = r_{sa} + \gamma_c \mathbb{E}_{s'} \left[Q_{s',\pi(s')}^r \right]. \quad (5)$$

42 Define a third mapping $f(Q^r, Q^c, \gamma_c) = 1 - \gamma_c$. Without loss of generality, we express all the
 43 operators as mappings from $\mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}$ to consider them as a coupled mapping from \mathbb{R}^{2d+1} to
 44 \mathbb{R}^{2d+1} . Finally, define two martingale difference sequences

$$M_{s,a}^c(n+1) = \gamma_c \min\{c_{sa}, \bar{Q}_{s',\pi(s')}^c(n)\} - \gamma_c \mathbb{E}_{s'} \left[\min\{c_{sa}, \bar{Q}_{s',\pi(s')}^c(n)\} \right] \text{ and} \quad (6)$$

$$M_{s,a}^r(n+1) = \gamma_c \bar{Q}_{s',\pi(s')}^r(n) - \gamma_c \mathbb{E}_{s'} \left[\bar{Q}_{s',\pi(s')}^r(n) \right]. \quad (7)$$

45 The updates to γ_c are deterministic. Using the above, we can express the three ACQL updates (1),
 46 (2), and (3) as

$$Q_{s,a}^c(n+1) = Q_{s,a}^c(n) + a(n) [g_{s,a}(\bar{Q}^r(n), \bar{Q}^c(n), \gamma_c(n)) - Q_{s,a}^c(n) + M_{s,a}^c(n+1)], \quad (8)$$

$$Q_{s,a}^r(n+1) = Q_{s,a}^r(n) + b(n) [h_{s,a}(\bar{Q}^r(n), \bar{Q}^c(n), \gamma_c(n)) - Q_{s,a}^r(n) + M_{s,a}^r(n+1)], \quad (9)$$

$$\gamma_c(n+1) = \gamma_c(n) + c(n) [f(Q^r(n), Q^c(n), \gamma_c(n)) - \gamma_c(n)]. \quad (10)$$

47 Under the above assumptions and formulation, ACQL can now be analyzed as a distributed stochastic
 48 approximation algorithm under three timescales.

49 A.2 Convergence of Q^r and Q^c under a fixed γ_c

50 Since the update to γ_c happens much more slowly than the updates to Q^r and Q^c —formally, the
 51 step size $c(n)$ for γ_c shrinks faster than both $b(n)$ and $a(n)$ —we can treat γ_c as approximately fixed
 52 while analyzing the behavior of Q^r and Q^c . This allows us to study the convergence of Q^r and Q^c
 53 assuming that γ_c is a constant value in the interval $(0, 1)$.

54 **Lemma 1.** The mapping $g^c = g(Q^r, \cdot, \gamma_c) : \mathbb{R}^d \rightarrow \mathbb{R}^d$, for some fixed Q^r , some fixed feasible policy
 55 π (e.g., a policy based on \bar{Q}^c and \bar{Q}^r as in ACQL), and $\gamma_c \in (0, 1)$, is a contraction mapping.

Proof.

$$\begin{aligned}
|g^c(Q^c)_{s,a} - g^c(\hat{Q}^c)_{s,a}| &= |\gamma_c \mathbb{E}_{s'} [\min\{c_{sa}, Q^c_{s',\pi(s')}\}] - \gamma_c \mathbb{E}_{s'} [\min\{c_{sa}, \hat{Q}^c_{s',\pi(s')}\}]| \\
&= \gamma_c \mathbb{E}_{s'} [|\min\{c_{sa}, Q^c_{s',\pi(s')}\} - \min\{c_{sa}, \hat{Q}^c_{s',\pi(s')}\}|] \\
&\leq \gamma_c \mathbb{E}_{s'} [|Q^c_{s',\pi(s')} - \hat{Q}^c_{s',\pi(s')}|] \quad (|\min\{a, b\} - \min\{a, c\}| \leq |b - c|) \\
&\leq \gamma_c \|Q^c - \hat{Q}^c\|_\infty
\end{aligned}$$

Therefore, $\|g^c(Q^c) - g^c(\hat{Q}^c)\|_\infty \leq \gamma_c \|Q^c - \hat{Q}^c\|_\infty$. \square

Lemma 2. As $n \rightarrow \infty$, $Q^c(n)$ converges to a fixed point $\lambda_1(Q^r, \gamma_c)$ for some fixed Q^r and γ_c .

Proof. The convergence of Q^c for a fixed Q^r and γ_c follows from Theorem 3 (convergence of distributed stochastic approximation algorithms for a contraction mapping) in [1]. Assumptions 1, 2, and 3 in [1] are satisfied due to our Assumptions 3, 4 and the definitions of $M^c_{s,a}$, $M^r_{s,a}$ in (6) and (7). Furthermore, the contraction property of g^c (Lemma 1) is enough to satisfy Assumption 5 in [1]. Under these conditions, Theorem 3 in [1] holds true. \square

Lemma 3. As $n \rightarrow \infty$, $Q^r(n)$ updated with (9) using a fixed $Q^c(n) = \lambda_1(Q^r(n), \gamma_c)$, such that there is a feasible action in every state, converges to the optimal value function $Q^{r*}_{\gamma_c}$.

Proof. The mapping $h^r = h(\cdot, Q^c, \gamma_c) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ for a fixed $Q^c = \lambda(Q^r, \gamma_c)$ is a typical Bellman operator for a fixed policy using only actions from a constant non-empty subset of \mathcal{A} for each state s . As a result, Theorem 4 (convergence of standard Q-learning) in [1] applies. \square

Lemma 4. $Q^r(n)$ and $Q^c(n)$ asymptotically approach $Q^{r*}_{\gamma_c}$ and $Q^{c*}_{\gamma_c} = \lambda_1(Q^{r*}_{\gamma_c}, \gamma_c)$ as $n \rightarrow \infty$.

Proof. Lemmas 2 and 3 serve to satisfy Assumptions 1 and 2 in Chapter 6 of [2]. The boundedness of our rewards and constraint signals also result in a bounded $Q^r(n)$ and $Q^c(n)$, which satisfies Assumption 3 in Chapter 6 of [2]. The proof follows from Theorem 2 (convergence of two-timescale coupled stochastic approximation algorithms) in the same chapter. \square

A.3 Convergence of (Q^r, Q^c) and γ_c

We can apply a similar two-timescale argument now using γ_c on the slower timescale and $(Q^r(n), Q^c(n))$ on the faster timescale. The condition that the faster timescale converges to a fixed point $\lambda_2(\gamma_c)$ for a static γ_c is shown in Lemma 4. For the condition that the slower timescale converges to a fixed point with $(Q^r(n), Q^c(n)) = \lambda_2(\gamma_c)$ is true trivially because γ_c converges without depending on $(Q^r(n), Q^c(n))$ at all.

Lemma 5. γ_c converges to 1 as $n \rightarrow \infty$.

Proof. The update in (10) is a discretization of the ODE $\dot{\gamma}_c(t) = 1 - \gamma_c(t)$. The solution to this ODE is $\gamma_c(t) = 1 - (1 - \gamma_c(0))e^{-t}$, which asymptotically approaches 1 as $t \rightarrow \infty$. \square

Finally, similar to Proposition 1 in [7], we observe that $\lim_{\gamma_c \rightarrow 1} g^c_{s,a}(Q^c) = \min\{c_{sa}, \mathbb{E}_{s'} Q^c_{s',\pi(s')}\}$ yields a fixed point at $Q^{c*}_{s,a} = \mathbb{E}_{\tau \sim \pi} [\min_{t \in [0, \infty]} c_{sa_t} | s_0 = s, a_0 = a]$ matching the undiscounted minimum-safety constraint (Equation (2) in our main paper). Using this fact and another application of Theorem 2 in [2], we can prove our Proposition 1.

Proposition 1. Let \mathcal{M}^A be an augmented CMDP with $|S^A| < \infty$, $|\mathcal{A}| < \infty$, and $\gamma \in [0, 1)$, and let $Q^c(n)$ and $Q^r(n)$ be models for the state-action safety and value functions indexed by n . Assume they are updated using Robbins-Monro step sizes $a(n)$ and $b(n)$, respectively, with $b(n) \in o(a(n))$ according to (1) and (2). Assume that $\gamma_c(n)$ is also updated with step sizes $c(n)$ such that $\gamma_c(n) \rightarrow 1$ and $c(n) \in o(b(n))$. Then $Q^c(n)$ and $Q^r(n)$ converge to Q^{c*} and Q^{r*} almost surely as $n \rightarrow \infty$.

91 *Proof.* By Theorem 2 from Chapter 6 of [2], whose conditions are satisfied by Lemmas 4 and 5,
 92 the coupled iterates $(\gamma_c(n), Q^r(n), Q^c(n))$ converge almost surely to a fixed point $(1, \lambda_2(1))$ with
 93 $\lambda_2(\gamma_c) = (Q_{\gamma_c}^{r*}, \lambda_1(Q_{\gamma_c}^{r*}, \gamma_c))$ as $n \rightarrow \infty$. As $\gamma_c \rightarrow 1$, $Q^c(n) = \lambda_1(Q_{\gamma_c}^{r*}, \gamma_c)$ also converges to
 94 $\mathbb{E}_{\tau \sim \pi} [\min_{t \in [0, \infty]} c_{sa_t} | s_0 = s, a_0 = a]$ with the policy π determined by $\lim_{n \rightarrow \infty} \bar{Q}^r(n) = Q^{r*}$.
 95 Therefore, the algorithm attains the optimal state-action value function and the corresponding optimal
 96 (undiscounted) state-action safety function Q^{c*} . \square

97 B Additional ACQL Details and Pseudocode

Algorithm 1 Automaton Constrained Q-Learning

Require: An MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, d_0, r, \gamma)$, an Signal Temporal Logic (STL) specification $\phi \in \Phi$,
 a safety limit $L \in [-1, 1]$, a learning rate α , an interpolation factor λ

- 1: $A \leftarrow \text{TRANSLATE}(\phi)$ \triangleright Sec. 4.1
- 2: $S, G \leftarrow \text{PARTITION}(A)$
- 3: $\mathcal{M}^A \leftarrow (\mathcal{S}^A, \mathcal{A}, \mathcal{T}^A, d_0^A, r^A, c^A, \gamma, L)$ \triangleright Sec. 4.1
- 4: $Q_\theta^c, Q_\psi^r \leftarrow \text{MAKENETWORKS}(\mathcal{S}^A, \mathcal{A})$
- 5: $\bar{\theta} \leftarrow \theta, \bar{\psi} \leftarrow \psi$
- 6: $R \leftarrow \text{MAKEREPLAYBUFFER}$
- 7: **for** $j = 1, \dots, N$ **do**
- 8: $\gamma_c \leftarrow \text{SAFETYGAMMASCHEDULER}(j)$
- 9: $\pi_j(s^A) \leftarrow \arg \max_{a: Q_\theta^c(s^A, a) > T} Q_\psi^r(s^A, a)$
- 10: $\tau \leftarrow \text{GETTRAJECTORY}(\mathcal{M}^A, \pi_j)$
- 11: $R \leftarrow R \cup \tau$.
- 12: $\mathcal{B}_j^\tau \sim R$
- 13: $\mathcal{B}_j^\tau \leftarrow \text{RELABEL}(\mathcal{B}_j^\tau)$
- 14: **for** $i = 1, \dots, M$ **do**
- 15: $\mathcal{B}_i \leftarrow \{(s_t^A, a_t, r_t, c_t, s_{t+1}^A)\} \sim \mathcal{B}_j^\tau$
- 16: $\theta \leftarrow \theta - \alpha \nabla_\theta L_{j,i}^c(\theta)$
- 17: $\psi \leftarrow \psi - \alpha \nabla_\psi L_{j,i}^r(\psi)$
- 18: $\bar{\theta} \leftarrow (1 - \lambda)\bar{\theta} + \lambda\theta, \bar{\psi} \leftarrow (1 - \lambda)\bar{\psi} + \lambda\psi$
- 19: **end for**
- 20: **end for**

98 **Training Summary** Algorithm 1 describes the full ACQL procedure. Our algorithm requires a
 99 base Markov Decision Process (MDP) \mathcal{M} , an STL specification ϕ , a safety limit $L \in [-1, 1]$, a
 100 learning rate α , and an interpolation factor λ as input. We translate ϕ into an automaton A (Line 1)
 101 that is partitioned to produce the mappings S and G (Line 2). These mappings and the automaton
 102 are combined with the input MDP to create the augmented CMDP in Line 3. The new state space
 103 \mathcal{S}^A of the CMDP and the action space \mathcal{A} are used to create the parameterized functions Q_θ^c and Q_ψ^r ,
 104 with their initial parameters θ and ψ , in Line 4. The initial parameters are copied to initialize the
 105 target parameters $\bar{\theta}$ and $\bar{\psi}$ (Line 5), and an empty replay buffer is initialized (Line 6). The algorithm
 106 proceeds to iterate for N epochs indexed by j . At each epoch, we obtain a value for the safety
 107 discount factor γ_c in Equation (4) in Section 4.2 of our main paper, which asymptotically approaches
 108 1.0 as training progresses (Line 8). The policy π_j for the epoch is defined as the function selecting
 109 the most-rewarding action according to Q_ψ^r constrained by Q_θ^c (Line 9). A trajectory τ is collected
 110 according to an epsilon-greedy version of this policy (Line 10), and this trajectory is added to the
 111 replay buffer (Line 11). We sample a batch of trajectories \mathcal{B}_j^τ and relabel them (Line 12-13). Finally,
 112 we sample M mini-batches of transitions \mathcal{B}_i from the relabeled trajectories (Line 15) and compute the
 113 Bellman losses for both Q_θ^c and Q_ψ^r over each mini-batch, with which we optimize their parameters
 114 via gradient descent (Lines 16-17). After each step, the target parameters $\bar{\theta}$ and $\bar{\psi}$ are updated toward
 115 the optimized parameters by the interpolation factor λ (Line 18). As the loop proceeds, the policy π_j
 116 converges to an effective and safe policy for the task specified by ϕ and \mathcal{M} .

117 **Automaton Analysis** To better illustrate the initial automaton analysis in ACQL (Line 2), consider
 118 the automaton in Figure 1. There is only one non-accepting sink-components of this automaton, and

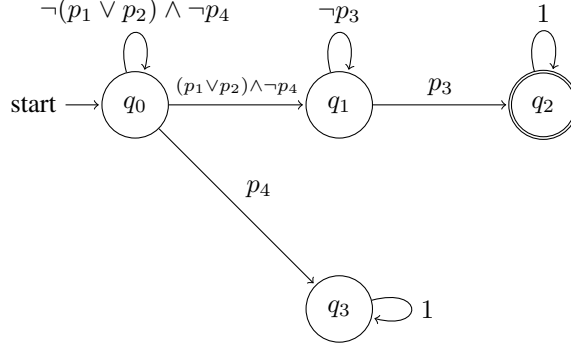


Figure 1: Automaton for the task “Reach goal g_1 or g_2 while never entering an unsafe-region u_1 . Then reach g_3 .”, where achieving the goal g_i corresponds to the atomic proposition p_i and entering u_1 corresponds to p_4 . The full Linear-time Temporal Logic (LTL) expression is $\neg p_4 \mathcal{U} ((p_1 \vee p_2) \wedge \Diamond p_3)$. The proposition p_4 is only relevant to the task’s safety constraint, and the propositions p_1 , p_2 , and p_3 are only relevant to the task’s liveness constraints.

119 it is the component consisting of the single node q_3 . For q_0 , there is only one transition into this
 120 component via the edge labeled by p_4 , so the safety for q_0 condition is $S(q_0) = \neg p_4$. For q_1 and
 121 q_2 , there are no transitions to a non-accepting sink-component and so their safety conditions are
 122 $S(q_1) = S(q_2) = 1$, meaning that the safety condition is trivially satisfied at all times in those states.
 123 For completeness, we also consider the unsafe states themselves as having safety conditions equal
 124 to the conjunction of their negated incoming transitions. Therefore, q_3 also has the safety condition
 125 $S(q_3) = \neg p_4$.

126 Now we can obtain the liveness constraints, which are summarized in the liveness condition mapping
 127 $O : \mathcal{Q} \rightarrow \Phi$ (See Section 4.1 in our main paper). For q_0 , the only remaining outgoing edge is the
 128 one labeled $\neg(p_1 \vee p_2) \wedge \neg p_4$. Since $S(q_0) = \neg p_4$, we can eliminate it from this transition predicate
 129 to obtain $O(q_0) = (p_1 \vee p_2)$. For q_1 , one can simply obtain $O(q_1) = p_3$ from its only outgoing
 130 edge. For completeness, we also set $O(q_2) = p_3$ using the incoming edges for q_2 since it is an
 131 accepting state. This is purely to inform the agent of the goal associated with an accepting state once
 132 it has reached it. From the above values for O , the mapping G can be defined by $G(q_0) = \{g_1, g_2\}$,
 133 $G(q_1) = \{g_3\}$ and $G(q_2) = \{g_3\}$.

134 **Subroutines** Algorithms 2, 3, and 4 present the pseudo-code for collecting trajectories in \mathcal{M}^A ,
 135 relabeling them with achieved goals, and computing the safety discount factor γ_c .

136 The GetTrajectory function, in Algorithm 2, begins by sampling an initial state from the distribution
 137 d_0^A . We also randomly sample positions from the environment to associate with each subgoal
 138 proposition $p \in AP_{\text{subgoal}}$. This task randomization promotes collecting trajectories that explore a
 139 greater portion of the state space through a variety of subgoal sequences, and we found that this was
 140 necessary to stabilize training while using Hindsight Experience Replay (HER). We speculate that
 141 the greater variety of state and subgoal combinations is needed to train a robust subgoal-reaching
 142 policy. The agent proceeds to interact with the environment for a total of T steps. In a loop indexed
 143 by t , actions are selected from an epsilon-greedy version of the input policy π (Line 5). The reward,
 144 constraint feedback, and next state are observed (Lines 6-8) and stored in the trajectory τ (Line 9).
 145 After T steps have executed, the trajectory is returned.

146 The Relabel function, in Algorithm 3, takes a batch of trajectories \mathcal{B}_τ . For each trajectory in the batch,
 147 the function determines the final achieved state g' and overwrites a single subgoal for every step in
 148 the entire trajectory with g' . It also overwrites the reward for each step with 1 for steps that were
 149 sufficiently close to g' and 0 for steps that were not. We found that this relatively simple strategy,
 150 despite the fact that trajectories were collected with multiple-subgoals in-mind, was sufficient to train
 151 reliable goal-reaching policies.

152 The SafetyGammaScheduler function, in Algorithm 4, generates values for γ_c , starting from an initial
 153 value and gradually increasing toward 1.0 with exponential decay. To ensure accurate learning of
 154 state-action safety function models, it was necessary to cap γ_c at a value slightly below 1.0.

Algorithm 2 GetTrajectory

```
1: function GETTRAJECTORY( $\mathcal{M}^A, \pi$ )
2:    $s_0^A \sim d_0^A, g_i \sim P(\mathcal{G})$  for  $p_i \in AP_{\text{subgoals}}$ 
3:    $t \leftarrow 0, \tau \leftarrow ()$ 
4:   while  $t < T$  do
5:      $a_t \sim \pi_{\epsilon\text{-greedy}}(s_t^A)$ 
6:      $r_t \leftarrow r^A(s_t^A, a_t)$ 
7:      $c_t \leftarrow c^A(s_t^A, a_t)$ 
8:      $s_{t+1}^A \sim \mathcal{T}^A(s_t^A, a_t)$ 
9:      $\tau \leftarrow \tau \cup (s_t^A, r_t, c_t, a_t)$ 
10:     $t \leftarrow t + 1$ 
11:   end while
12:   return  $\tau$ 
13: end function
```

Algorithm 3 Relabel

```
1: function RELABEL( $\mathcal{B}_r$ )
2:   for  $\tau \in \mathcal{B}_r$  do
3:      $g' \leftarrow$  the final goal state achieved in  $\tau$ .
4:     for  $s_t^A, r_t \in \tau$  do
5:       Replace  $g_1$  in  $s_t^A$  with  $g'$ 
6:        $r_t \leftarrow 1$  if  $g'$  achieved in  $s_t^A$  and 0 otherwise.
7:     end for
8:   end for
9:   return  $\mathcal{B}_r$ 
10: end function
```

C Model Architectures, Environment Implementation Details, and Hyper-Parameters

Model Architecture and Policy Our method trains two models: the state-action value function, Q_ψ^r , and the state-action safety function, Q_θ^c . Both models employ twin neural networks and use the minimum of their predicted values to mitigate overestimation in value and safety estimation. For the ablation study using a state-action sum-of-costs function, we likewise use twin networks, but instead take the maximum of the two predictions to maintain a conservative (i.e., pessimistic) estimate.

To simplify handling multiple goals, Q_ψ^r is defined using a goal-conditioned state-action value function model, $Q_\psi^{GC} : \mathcal{S} \times \mathcal{G} \times \mathcal{Q} \times \mathcal{A} \rightarrow \mathbb{R}$, parameterized by ψ . At runtime, Q_ψ^{GC} is called for each subgoal $g \in g^+$. This approach exploits the fact that goal-conditioned value functions form a Boolean algebra under the min and max operators [8, 9]. For example, consider two subgoal propositions $p_1, p_2 \in AP_{\text{subgoal}}$. In the simplest case, when $O(q) = p_1$, we compute $Q_\psi^r(\langle s, g_1, q \rangle, a)$ as $Q_\psi^{GC}(s, g_1, q, a)$. When $O(q) = p_1 \wedge p_2$ (i.e., both subgoals must be achieved to progress), we compute the value as the minimum of $Q_\psi^{GC}(s, g_1, q, a)$ and $Q_\psi^{GC}(s, g_2, q, a)$. Conversely, when $O(q) = p_1 \vee p_2$ (i.e., achieving either subgoal suffices), the value is the maximum of $Q_\psi^{GC}(s, g_1, q, a)$ and $Q_\psi^{GC}(s, g_2, q, a)$. This pattern generalizes to arbitrarily complex Boolean formulae, allowing us to efficiently approximate Q_ψ^r using a single goal-conditioned network.

We also observed that, because liveness constraints are separated into the goal input, conditioning behavior on the automaton state is only necessary when safety constraints differ between automaton states. As a result, we do not require a distinct “mode” for every automaton state $q \in \mathcal{Q}$, but only for each unique safety condition in the mapping S . To encode this, we train Q_ψ^r using a multi-headed neural network, where each output head corresponds to a distinct safety condition. For example, in the automaton shown in Figure 1, the mapping S assigns states to one of two safety conditions: $\neg p_4$ or 1. Accordingly, Q_ψ^{GC} has two output heads, selected based on the currently active safety condition. In all ACQL experiments, Q_ψ^{GC} shared a hidden layer of 256 neurons across all heads; each head then had an additional hidden layer of 256 neurons. All layers used ReLU activations, and the final output layer had $|\mathcal{A}|$ neurons with no activation function.

Algorithm 4 SafetyGammaScheduler

```
1: function SAFETYGAMMASCHEDULER( $j$ )  
2:    $x \leftarrow j \div \text{update\_period}$   
3:    $y \leftarrow 1.0 - (1.0 - \text{init\_value}) \cdot \text{decay\_rate}^x$   
4:   return  $\begin{cases} \text{max\_value} & \text{if } y \geq \text{max\_value} \\ y & \text{o.w.} \end{cases}$   
5: end function
```

182 The model Q_θ^c follows the same architecture as Q_ψ^r , with a goal-conditioned, multi-headed neural
183 network, but differs in two key respects. First, for disjunctive goal conditions ($p_1 \vee p_2$), the output
184 is defined as the minimum of $Q_\theta^{GC}(s, g_1, q, a)$ and $Q_\theta^{GC}(s, g_2, q, a)$, to ensure conservative safety
185 by taking the worst-case estimate across disjunctive paths. Second, Q_θ^{GC} uses a different network
186 architecture: it has two shared hidden layers of 64 neurons each, and each output head includes two
187 additional hidden layers with 64 and 32 neurons, respectively. All layers use ReLU activations. The
188 final output layer consists of $|\mathcal{A}|$ neurons with a tanh activation function.

189 The policy was implemented in terms of these two value functions according to the constrained
190 maximization $\pi(s) = \arg \max_{a: Q_\theta^c(s, a) > L} Q_\psi^r(s, a)$. In the case that no action was deemed feasible
191 by Q_θ^c , the safest action $\max_a Q^c(s, a)$ was chosen. The exploration policy $\pi_{\epsilon\text{-greedy}}$ would behave
192 as above with probability $1 - \epsilon$, and with probability ϵ select a random action without considering
193 Q^r or Q^c .

194 **Environment Implementation** For our simulated experiment environments, we used the Brax
195 physics simulator [10] and assets provided in JaxGCRL [11] for the PointMass, Quadcopter, and Ant
196 environments. Acting in these environments was facilitated by a set of discrete actions corresponding
197 to movement in the cardinal directions. Specifically, the actions in the PointMass and Quadcopter
198 environments output a constant low-level action to accelerate in one of the four or six available
199 directions for 5 consecutive steps. The actions for our UR5e environment, which we used to train our
200 real-world-deployed policies, similarly moved the robot’s end effector in the 6 cardinal directions for
201 a single time step. The actions in the AntMaze environment were policies trained separately using
202 Proximal Policy Optimization (PPO) [12] for 50000000 environment interactions with the objective
203 of maximizing velocity in each of the four cardinal directions and would run for 4 consecutive steps
204 when executed. All further details regarding environment geometry and task definitions for our
205 simulated and real-world experiments are included in our Anonymized Code Repository¹.

206 **Hyper-Parameters** Table 1 reports the hyperparameter values most commonly used in our experi-
207 ments, including hyperparameters for the safety gamma (γ_c) scheduler described in Appendix B. For
208 a complete account of hyperparameters, as well as ACQL, baseline, and environment implementation
209 details, refer to our Anonymized Code Repository²1.

Table 1: Hyperparameter values used for experiments in Tables 1 and 2 in our main paper

Hyperparameter Name	Value
Episode length (T)	1000
Discount factor (γ)	0.99
Learning rate (α)	$1 \cdot 10^{-4}$
ϵ -greedy factor (ϵ)	0.1
Safety limit (L)	0.0
Safety Gamma Init Value	0.80
Safety Gamma Update Period	250,000
Safety Gamma Decay Rate	0.15
Safety Gamma Max. Value	0.98
Target parameter interpolation factor (λ)	0.005

¹<https://anonymous.4open.science/r/acql-4B4C>

Compute Resource Requirements All experiments were conducted on a single NVIDIA RTX 3090 GPU (24 GB VRAM), using a local workstation equipped with an 12th Gen Intel i7-12700F CPU, 32 GB RAM. No cloud services or compute clusters were used. Each individual experimental run required approximately 30 minutes of compute time on the GPU. The full experiment grid consists of 225 runs for the comparative analysis and 90 runs for the ablations, amounting to approximately 315 GPU-hours. Minor additional compute was used for initial hyperparameter tuning and development.

D Expanded Experimental Results

Figures 2 and 3 show the average reward and success rate throughout training for the baseline comparison experiments summarized in Table 1. The Logical Options Framework (LOF) baseline cannot be depicted on these plots as it does not learn a single policy in the same MDP as the other methods, and instead learns a policy that chooses subgoal-specific options for an abstracted state space $\mathcal{Q} \times \mathcal{S}_g$ constructed from the automaton states \mathcal{Q} and the finite set of states $\mathcal{S}_g \subset \mathcal{S}$ corresponding to task subgoals. Figures 4 and 5 show the average reward and success rate throughout training for the experiments summarized in Table 2 in our main paper. The differences in amount of training steps depicted by the figures is due to the different design and training pipelines that the two algorithms observe. ACQL collects complete trajectories to store in the Replay Buffer and Counterfactual Experiences for Reward Machines with Reward Shaping (CRM-RS) just collects individual transitions. We want to highlight that our algorithm converges earlier during the training and this difference does not play a significant role in the performance gap reported in Table 1.

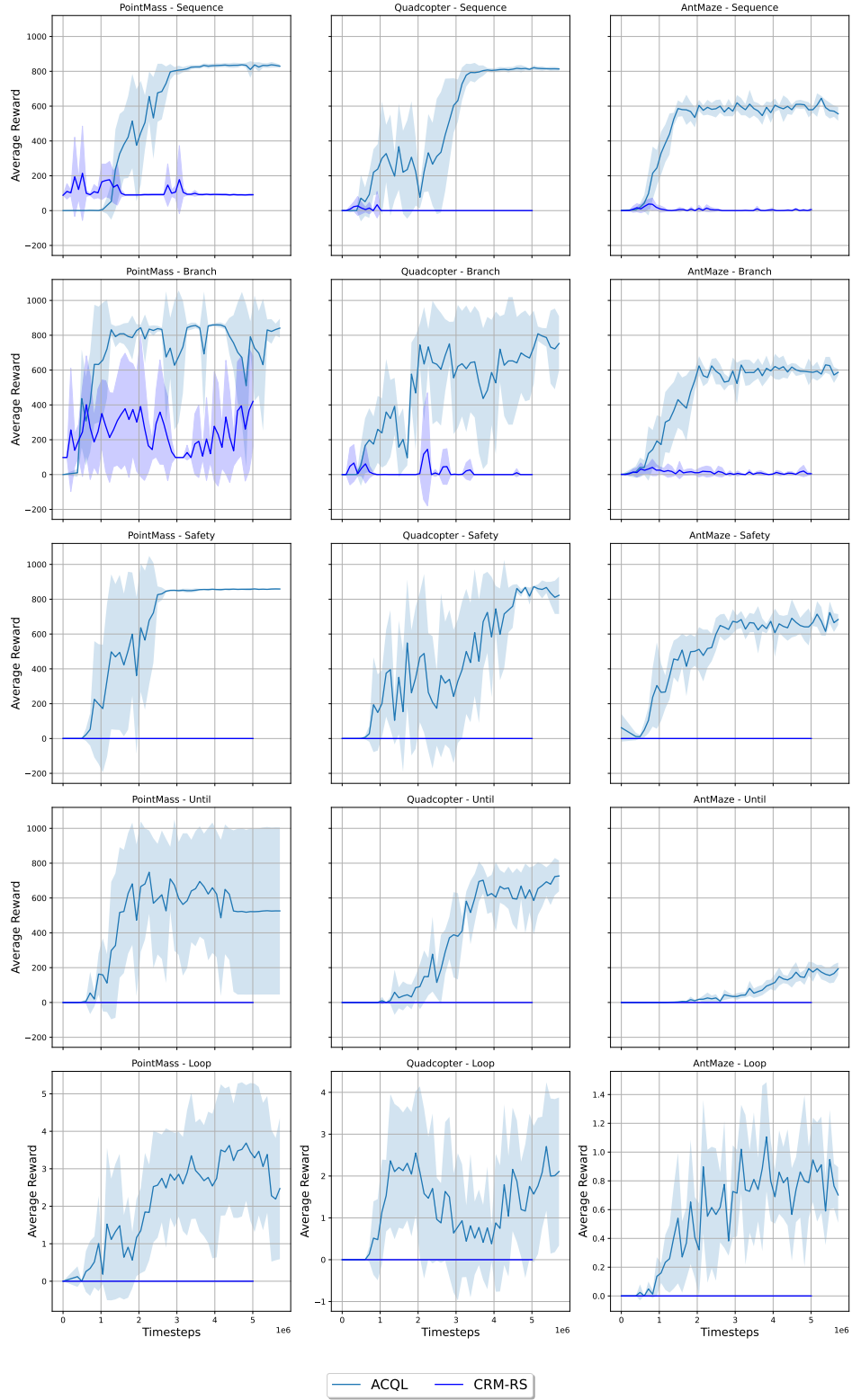


Figure 2: Average and one standard deviation of episode reward throughout training for the five runs per method that are summarized in Table 1 in our main paper.

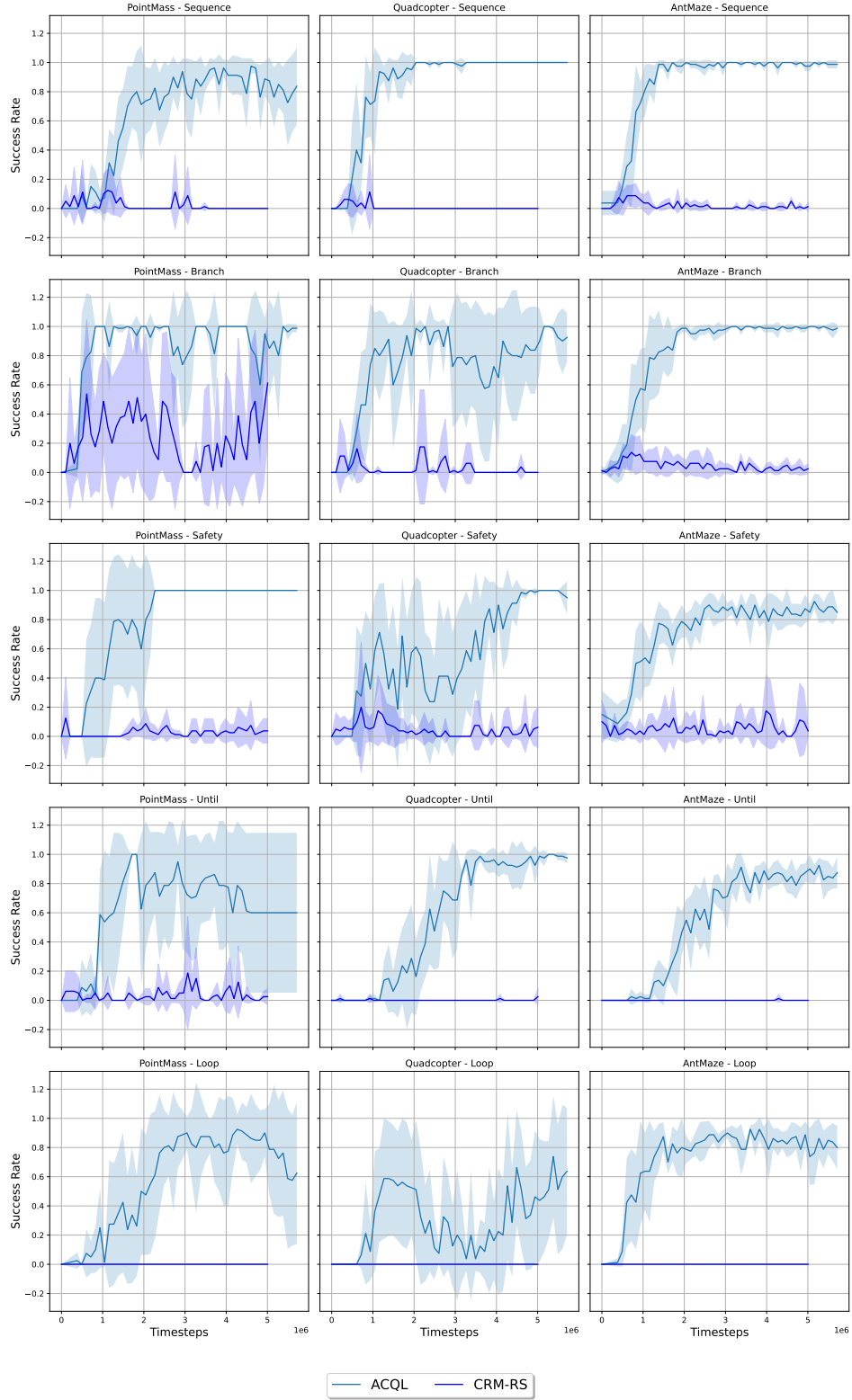


Figure 3: Average and one standard deviation of episode success rate throughout training for the five runs per method that are summarized in Table 1 in our main paper.

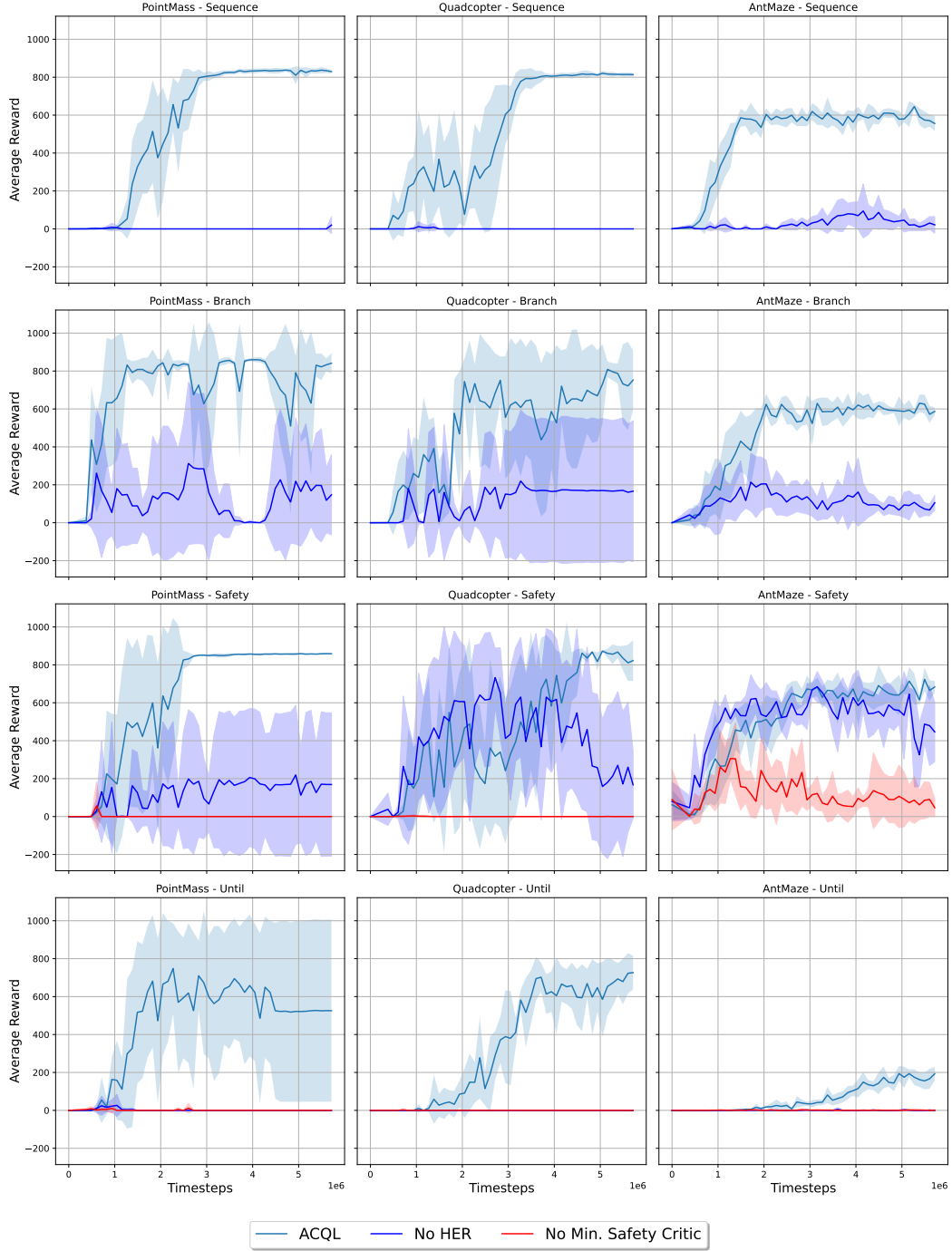


Figure 4: Average and one standard deviation of episode reward throughout training for the five runs per ablation group that are summarized in Table 2 in our main paper.

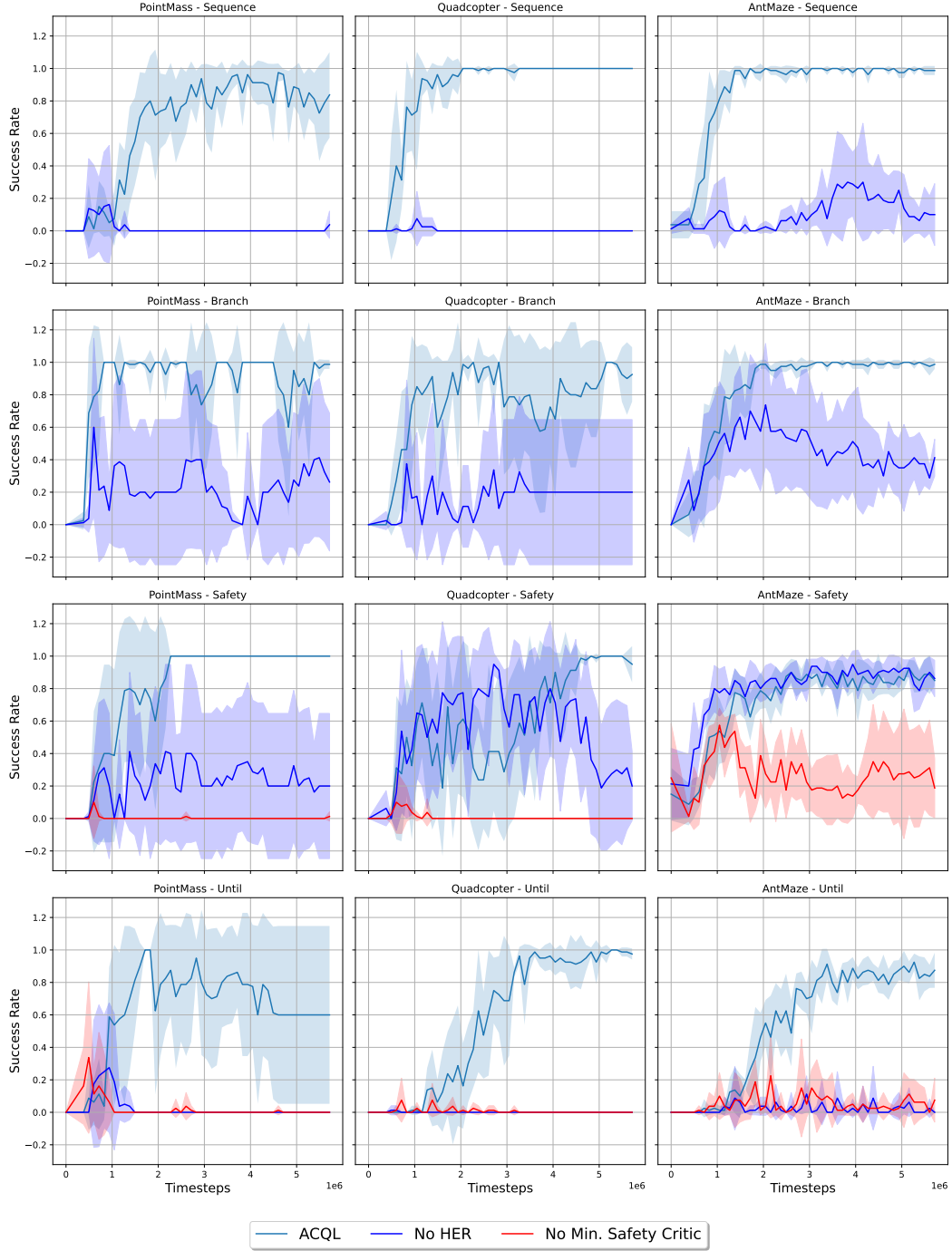


Figure 5: Average and one standard deviation of episode success rate throughout training for the five runs per ablation group that are summarized in Table 2 in our main paper.

References

- [1] John N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Mach. Learn.*, 16(3):185–202, September 1994.
- [2] Vivek S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint*. Hindustan Book Agency Gurgaon, 2008.
- [3] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria, 2017.
- [4] Haitong Ma, Changliu Liu, Shengbo Eben Li, Sifa Zheng, and Jianyu Chen. Joint synthesis of safety certificate and safe control policy using constrained reinforcement learning. In Roya Firoozi, Negar Mehr, Esen Yel, Rika Antonova, Jeannette Bohg, Mac Schwager, and Mykel Kochenderfer, editors, *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168 of *Proceedings of Machine Learning Research*, pages 97–109. PMLR, 23–24 Jun 2022.
- [5] Dongjie Yu, Haitong Ma, Shengbo Li, and Jianyu Chen. Reachability constrained reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 25636–25655. PMLR, 17–23 Jul 2022.
- [6] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 2015.
- [7] Jaime F. Fisac, Neil F. Lugovoy, Vicenç Rubies-Royo, Shromona Ghosh, and Claire J. Tomlin. Bridging hamilton-jacobi safety analysis and reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8550–8556, 2019.
- [8] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A boolean task algebra for 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 9497–9507. Curran Associates, Inc., 2020.
- [9] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. Skill machines: Temporal logic composition in reinforcement learning. In *Deep Reinforcement Learning Workshop @ NeurIPS*, 2022.
- [10] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021.
- [11] Michał Bortkiewicz, Władek Pałucki, Vivek Myers, Tadeusz Dziarmaga, Tomasz Arczewski, Łukasz Kuciński, and Benjamin Eysenbach. Accelerating goal-conditioned rl algorithms and research. *arXiv preprint arXiv: 2408.11052*, 2024.
- [12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.