A PROOF OF PROPOSITIONS

We prove Proposition 2 before Proposition 1. We use the following Lemma, which is stated in two parts for clarity (part (B) generalizes part (A)). The Lemma considers the "N-Step Chain MDP", exemplified in Figure 2, with $N+1$ states, and b actions in each state, which each may have different reward distributions.

Lemma 1 (Augmentation-Bootstrapping Equivalence)**.**

(A) For any particular 2-step trajectory, $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, s_2)$, in a 2-Step Chain MDP, where the subtrajectory $\tau_0 = (s_0, a_0, r_0, s_1)$ appears n times in a given empirical dataset, and the subtrajectory $\tau_1 = (s_1, a_1, r_1, s_2)$ appears m times, the following return estimators are equivalent:

1. *Augmentation:*
   - *First form an augmented dataset of nm trajectories, by concatenating each pair of trajectory segments τ_0 and τ_1. Then estimate the return of τ as the empirical average of its return in the augmented dataset:*

$$G(\tau) = \frac{\sum_{i=1}^{nm} r_0^i + r_1^i}{nm}$$

2. *Bootstrapping:*
   - *First estimate the return of τ_1 as $G(\tau_1) = \sum_i^m r_1^i / m$. Then estimate the return of τ using the bootstrapped estimator:*

$$G(\tau) = \frac{\sum_{i=1}^{n} r_0^i + G(\tau_1)}{n}$$

(B) Given an N-step Chain MDP and an N-step trajectory τ, whose 1-step subtrajectories, $\{\tau_j = (s_j, a_j, r_j, s_{j+1})\}$, each appear m_j times in the empirical dataset, the following return estimators are equivalent:

1. *Augmentation:*
   - *First form an augmented dataset of $\Pi_j m_j$ trajectories, by concatenating each combination of trajectory segments. Then estimate the return of τ as the empirical average of its return in the augmented dataset:*

$$G(\tau) = \frac{\sum_{i=1}^{\Pi_j m_j} \sum_{k=0}^{N-1} r_k^i}{\Pi_j m_j}$$

2. *$N-1$ steps of Bootstrapping:*
   - *Denoting the trajectory slice from s_k to s_N as $\tau_{k:}$, estimate the return of τ using the bootstrapped estimator:*

$$G(\tau_{k:}) = \frac{\sum_{i=1}^{m_k} r_k^i + G(\tau_{k+1:})}{m_k}$$

   where $G(\tau_{N:}) := 0$.
3. *Sum of local reward estimators:*
   - *Estimate the return of τ as a sum of the returns of its 1-step components:*

$$G(\tau) = \sum_{k=1}^{N} \frac{\sum_{i=1}^{m_k} r_k^i}{m_k}$$

   where $G(\tau_{N:}) := 0$.

Proof. Part (A) is a special case of Part (B)(1)-(2). Both follow by induction from the trivial base case

$$G(\tau_{N-1}) = G(\tau_{N-1:}) = \frac{\sum_{i}^{m_{N-1}} r_{N-1}^i}{m_{N-1}},$$

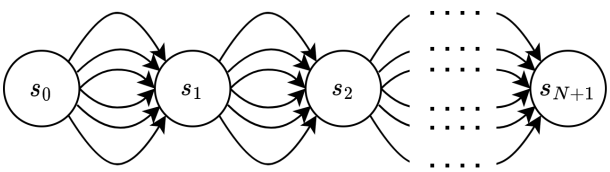

Figure 4: ***N*-Step Chain MDP:** The MDP is initialized in s_0 and terminates in s_{N+1}, with actions always leading to the next state in the chain. In this MDP there are b^N possible trajectories.

where the inductive step is:

$$
\begin{aligned}
G(\tau_{t:}) = \frac{\sum_{i=1}^{\Pi_{j=t}^{N-1} m_j} \sum_{k=t}^{N-1} r_k^i}{\Pi_{j=t}^{N-1} m_j} &= \frac{\sum_{i=1}^{\Pi_{j=t}^{N-1} m_j} \left(r_t^i + \sum_{k=t+1}^{N-1} r_k^i \right)}{m_t \Pi_{j=t+1}^{N-1} m_j} \\
&= \frac{\sum_{i=1}^{\Pi_{j=t}^{N-1} m_j} r_t^i}{m_t \Pi_{j=t+1}^{N-1} m_j} + \frac{\sum_{i=1}^{\Pi_{j=t}^{N-1} m_j} \sum_{k=t+1}^{N-1} r_k^i}{m_t \Pi_{j=t+1}^{N-1} m_j} \\
&= \frac{\sum_{i=1}^{m_t} r_t^i}{m_t} + \frac{m_t G(\tau_{t+1:})}{m_t} \qquad (*) \\
&= \frac{\sum_{i=1}^{m_t} r_t^i + G(\tau_{t+1:})}{m_t}
\end{aligned}
$$

Part (B)(3) follows by repeatedly unrolling the last term in equation (∗) above. □

As a straightforward consequence of the above Lemma, we have:

Proposition 2 (Coverage of Trajectory Space)**.** *In a deterministic N-step Chain MDP with branching factor b, it is possible for N-step bootstrapping to (implicitly) capture full "coverage" of the size b^N trajectory space with only b empirical trajectories—an exponential increase in coverage relative to no bootstrapping. Coverage here refers to the percentage of unique trajectories present in the dataset, where we consider two trajectories with the same actions taken in every state as equivalent.*

Proof. This follows directly from Lemma 1(B)(1)-(2) if every possible 1-step subtrajectory $\{\tau_j = (s_j, a_j, r_j, s_{j+1})\}$ appears exactly once in the b empirical trajectories. □

We can additionally make the following related statement about the sample complexity of valuing each trajectory in the Chain MDP when rewards are stochastic (note that in the Chain MDP, a trajectory is equivalent to a policy, so that the following proposition is a finite sample complexity bound on "every policy" policy valuation in the Chain MDP).

Proposition 2a (Sample Complexity of Policy Valuation in Stochastic Case)**.** *Consider an N-Step Chain MDP with branching factor b, whose rewards at each action are bounded random variables with $R(s, a) \in [0, 1]$.*

(A) Suppose we have n samples $\{\tau^{(i)}\}_{i=1\ldots n}$ of each length N trajectory. There are b^N such trajectories, providing a total of $m = Nb^N n$ samples of length 1 subtrajectories. Without bootstrapping, if

$$
n \geq \frac{N^2}{2\epsilon^2} \log \frac{2b^N}{\delta}, \text{ or equivalently, } m \geq \frac{N^3 b^N}{2\epsilon^2} \log \frac{2b^N}{\delta},
$$

then, with probability at least $1 - \delta$, we have:

$$
max_\tau \left(\frac{\sum_i G(\tau^{(i)})}{n} - \mathbb{E}\left[G(\tau)\right] \right) \leq \epsilon.
$$

(B) *Suppose we have at least ℓ samples of each length 1 subtrajectory $\tau_j = (s_j, a_j, r_j, s_{i+1})$. There are bN such subtrajectories, providing a total of $m = bN\ell$. samples of length 1 subtrajectories. Using N-Step bootstrapping as described in Lemma 1, if*

$$\ell \geq \frac{N^2}{2\epsilon^2} \log \frac{2bN}{\delta}, \text{or equivalently, } m \geq \frac{N^3 b}{2\epsilon^2} \log \frac{2bN}{\delta},$$

then, with probability at least $1 - \delta$, we have:

$$max_\tau \left(\frac{\sum_i G(\tau^{(i)})}{n} - \mathbb{E}\left[G(\tau)\right] \right) \leq \epsilon.$$

Therefore, in an N-Step Chain MDP with $b > 1$, plain RvS requires b^{N-1} times as many (i.e., exponentially more) samples to obtain the same precision as N-step bootstrapping.

Proof.

(A) This follows from Hoeffding's Inequality by noting that $G(\tau) \in [0, N]$ and taking a union bound over the b^N trajectories.

(B) This follows from Hoeffding's Inequality using $\epsilon' = \epsilon/N$, so that the trajectory error (a sum of N 1-step subtrajectories) is bounded by ϵ, and taking a union bound over the bN 1-step subtrajectories.

□

The Chain MDP used in Proposition 2 reveals a general construction to prove necessity, as follows.

Proposition 1 (Necessity)**.** *For any positive integer n, there exists an MDP $\mathcal{M}$ and data generating policy π_e for which n-step bootstrapping is strictly necessary to generate dataset $\mathcal{D}$ containing (s, a, g^*), where g^* is the maximum reward achievable for a trajectory starting in (s, a).*

Proof. For the general case, consider an $n + 1$ state Chain MDP with a single optimal trajectory, where $\mathcal{D}$ is generated by a combination of n non-Markovian policies that each contain a single, unique length 1 segment of the optimal trajectory. A single step of SUPERB bootstrapping can add at most 1 new length 1 segment to any particular (augmented) return label, so that n steps of SUPERB bootstrapping are necessary to compose all n length 1 segments from the optimal policy. □

B IMPLEMENTATION DETAILS

We implemented our experiments on top of the code for RvS (Emmons et al., 2021) found at `https://github.com/scottemmons/rvs`.

B.1 RETURN MODEL

To model the distribution of returns we use an ensembled Quantile Regression Network (QRN), as proposed by Dabney et al. (2018), which maps states to a value distribution represented by 20 quantiles. The quantile regression is trained using the Huber quantile loss proposed by Dabney et al. (2018) with $k = 1$, and is optimized for 5 epochs (Antmaze) or 10 epochs (Gym) using AdamW (Loshchilov & Hutter, 2019) using a batch size of 1024, and a constant learning rate of 1e-3 for AntMaze and 3e-4 for the Gym tasks.

Our QRN is a ensemble of 5 feedforward neural networks, each with 3 hidden layers of 512 neurons and ReLU activations. Both the inputs to the networks, and the output targets are normalized. For AntMaze experiments, we apply value clipping to clip target return values to their known feasible range (after reward transformation) of $[-1/(1-\gamma), 0]$. In each bootstrapping step, a new randomly initialized QRN is trained on the current augmented dataset.

To form augmented labels, we first use the QRN to propose return labels for all trajectory suffixes in the dataset. This is done by calling each member of the QRN ensemble on the first observation

in the trajectory suffix, taking the mean (Antmaze) or minimum (Gym) across the ensemble, and average the top 5 quantiles of the result.

Proposed return labels in hand, we then apply the following backward induction procedure to relabel all return labels in $\mathcal{D}$:

```
def qr_augmented_return_labels(traj, proposed_labels, discount_factor):
    rewards = traj.rewards
    returns = []
    ret = 0
    traj_len = len(rewards)

    for i in reversed(range(traj_len)):
        ret *= discount_factor
        ret += (float(rewards[i]))
        if (i + 1 < traj_len):
            ret = max(ret, float(rewards[i]) +\
                  discount_factor * proposed_labels[i + 1])
        returns.append(ret)
    returns = list(reversed(returns))
    return returns
```

On each iteration of SUPERB, we keep only the most recently generated augmented labels, and discard both the QRN for that step and the data it was trained on. After the final step, we train one more QRN to use as the value function for the purpose of forming reward targets for the RvS policy, described next.

B.2 RvS Policy

For simplicity, we base our policy learning heavily off of the hyperparameters discussed in Emmons et al. (2021). In order to support conditioning on higher values of return, we implemented input normalization for the policy network, where inputs are normalized by the mean and standard deviation of the inputs from the dataset. Due to this normalization, policy training could be completed in fewer epochs and we decreased the number of training epochs from 2000 to 400 for D4RL-gym and trained for only 100 epochs on AntMaze. As in Emmons et al. (2021), our policy consists of a simple feedforward neurial network with two hidden layers with width equal to 1024. We optimized our policy network using AdamW (Loshchilov & Hutter, 2019).

As discussed in section 3.5, we dynamically choose a return target g^* by using our learned return model. However, we found that we can additionally improve performance by increasing the target by some Δ to just above the value predicted by $V_\phi(G \mid s)$. We tune Δ separately for each environment and number of iterations, as the return distribution varies with these variables.