We thank the reviewers for their constructive comments. We address the main concerns below.

R1/R3: Running time and practicality of ApproPO: In our experiments, we implement an RL oracle by a policy-gradient algorithm similar to RCPO (see Sec. 4), so our oracle has a similar *per-transition* running time as RCPO. The trajectories executed by our oracle have similar average lengths ($\approx$21) as those of RCPO ($\approx$24), so our oracle also has a similar *per-trajectory* running time as RCPO. We report the performance of our algorithm in terms of the total number of trajectories across all the RL oracle calls so far, so the part of the time that our algorithm spends in the RL oracle calls is commensurate with the RCPO time at any given number of trajectories. Our algorithm additionally performs updates of $\boldsymbol{\lambda}$, but these are orders of magnitude cheaper than the per-trajectory running time of the RL oracle / RCPO, because the dimension of $\boldsymbol{\lambda}$ is either 2 or 66 (without or with diversity constraints, respectively), whereas the policies π over which the oracle and RCPO optimize are two-layer networks described by 8,704 floating-point numbers.

In our implementation, it was crucial to use the improvements from Sec. 3.4. We ran the "positive response" version of ApproPO (Algorithm 5) for 2000 outer-loop iterations (i.e., 2000 updates of $\boldsymbol{\lambda}$), but needed to make at most 61 RL oracle calls in any of our replicates; the remaining outer-loop iterations recovered a positive response from cache. The cache look-up is fast since it only involves performing an inner product of $\boldsymbol{\lambda}$ with stored expected measurement vectors.

The main overhead of ApproPO compared with RCPO is not the running time, but the memory, because of the policies stored in cache. This is quite modest (unless policy networks are very large), since the cache only stores the results of RL oracle calls. Note that the policy mixture returned by ApproPO is just a weighted combination of the policies from cache.

We will add this discussion to the paper and also update plots, so they are in terms of transitions rather than trajectories.

R2: Clarification questions:

- **definition of $\Delta(\Pi)$:** It is the set of probability distributions over policies in Π. The elements of Δ are viewed as mixed policies (see paragraph starting at line 73).
- **distance between vector $\mathbf{z}$ and a set $\mathcal{C}$:** You are right: we mean the distance to the closest point in $\mathcal{C}$ under the Euclidean distance (see Eq. (3)). Using other distance measures is open for future work; we expect it will allow us to use other no-regret algorithms and possibly improve convergence (in some cases).
- **reward as a dot product of $\boldsymbol{\lambda}$ and $\mathbf{z}$:** We construct the reward as a dot product only when we invoke the RL oracle—this is by design, since the oracle is a standard RL algorithm that seeks to optimize a scalar reward. Our algorithm, however, solves the multi-dimensional feasibility (or distance minimization) problem, which prior work does not handle. Our algorithm accomplishes this by systematically updating $\boldsymbol{\lambda}$, calling the RL oracle for different $\boldsymbol{\lambda}$ vectors, and then combining the returned policies into a mixture.
- **trajectories = episodes**: Yes, trajectories coincide with episodes.

R2: ApproPO vs projected gradient methods, e.g., PNAC: Projected gradient methods implement constraints in the *policy parameter space*, our constraints are on the observable measurements coming from the environment. The measurements have clear behavioral interpretation, so arguably constraining the measurements is quite natural. On the other hand, it can be quite hard to design constraints on the policy parameters. For example, it is not clear how to cast constraints from our experiments in terms of the weights parameterizing our two-layer policy networks.

R3: Reproducibility checklist: You are right; we apologize for this oversight. We ran out of time to anonymize the code properly. We are planning to publish the code with the camera-ready version.

R3: Extending RCPO to convex constraints: This was actually our original motivation, but in the end, a systematic solution turned out to be non-trivial and led to this paper (we are not aware of any simple extension in the literature).

R3: Comparison with previous work: Kalathil et al. (2014) and CPO (Achiam el at., 2017) address more restricted settings. Kalathil et al. focus on long-term average measurements in irreducible MDPs, their algorithm is a variant of Q-learning, and they only prove asymptotic convergence. CPO only handles orthant constraints, only supports discounted-sum rewards, and requires all iterates to satisfy the constraints (this is desirable in safe RL, but may be restrictive otherwise). Our approach works with both Q-learning and policy search oracles, supports both average measurements and discounted sums, and comes with non-asymptotic convergence guarantees. The setting of RCPO is similar to ours and it has demonstrated competitive empirical performance, so it was a natural baseline to include.

R3: ApproPO plots end before RCPO plots: This is because ApproPO terminates after 2000 outer loop iterations (note that the constraints are at that point approximately satisfied).