Thank you for the feedback. We presented an efficient method to combine λ-returns and experience replay for the first time, yielding faster learning on several Atari games. This method is generally applicable to replay-based learning and enables interesting avenues for prioritized replay and dynamic λ selection, both of which we explored in our work.

Addressing **Reviewer 3's** comments, dynamic λ selection performed well on only some of the games. We have explored an alternative strategy (see figure below) that outperforms the baseline on 3 games and performs comparably on the other 3. We define $R_t^{\lambda}=\text{median}(R_t^{\lambda=0/k},R_t^{\lambda=1/k},\ldots,R_t^{\lambda=k/k})$, where we used $k=20$. That is, we compute many λ-returns offline, and then select the median at each timestep. This is appealing because it integrates multiple λ-values in an intuitive way, is robust to outliers, and eliminates the hyperparameter δ. Because it does not require a binary search, the expected runtime is similar or better. We can include these results in the main paper or the supplementary material.

Reviewer 2 expressed concerns about computational cost. Figure 8 (supplementary material) shows that DQN(λ) runs about 33% faster than standard DQN. This is because cache chunks are larger than a typical minibatch, so we achieve much better GPU parallelization when training. We can include additional runtime figures in the final paper, if desired.

Reviewer 1 asked 6 questions, which we answer here:

1. Each λ-return needs only 1 Q-value due to the recursive formula: $R_t^{\lambda}=R_t^{(1)}+\gamma\lambda[R_{t+1}^{\lambda}-\max_{a'\in\mathcal{A}}Q(\hat{s}_{t+1},a')]$. Consequently, a 100-length chunk will require 100 Q-values, *i.e.* an average of 1 Q-value per λ-return.

2. The main practical benefit of refreshing Q-values (instead of using a target network) is saving memory.

3 & 6. We prioritize experience by stochastically mixing two *truncated* cache copies (one sorted by TD error magnitude, one shuffled) with probability parameter p. The shuffling/sorting operations occur before truncation; therefore, on average, the sorted copy has a higher absolute TD error than the shuffled copy, but it is also biased. Training solely on the sorted copy ($p=1$) could yield a poor policy as a result. On the other hand, $p=0$ would completely disable prioritization and could miss opportunities for faster learning.

Consequently, we recommend annealing $p\rightarrow 0$ to initially encourage faster learning and later facilitate accurate Q-value estimation. Annealing p too slowly could result in convergence to a bad policy. Annealing p too quickly could slow learning. While we experimented with this new technique, prioritization methods from previous work are certainly possible here too, and would benefit from the true TD error information provided by the cache.

4. Figure 2 is our ablation study, where score is plotted against timesteps. Although the Pong curves look similar, they are significant based on the standard deviation. We can include additional games in this study for the final paper.

5. The oversampling ratio X controls how many extra samples are promoted to the cache during training, which has implications for the amount of bias and the total runtime. If the cache is very large ($X\gg 1$), then minibatches drawn from it will be mostly uncorrelated, but a great number of computed λ-returns would be wasted. It is generally desirable to use a lower X value for faster runtime when some bias can be tolerated. However, if $X=1$ (minimal), our prioritization method could not work because there would be no extra samples to truncate from the cache copies.

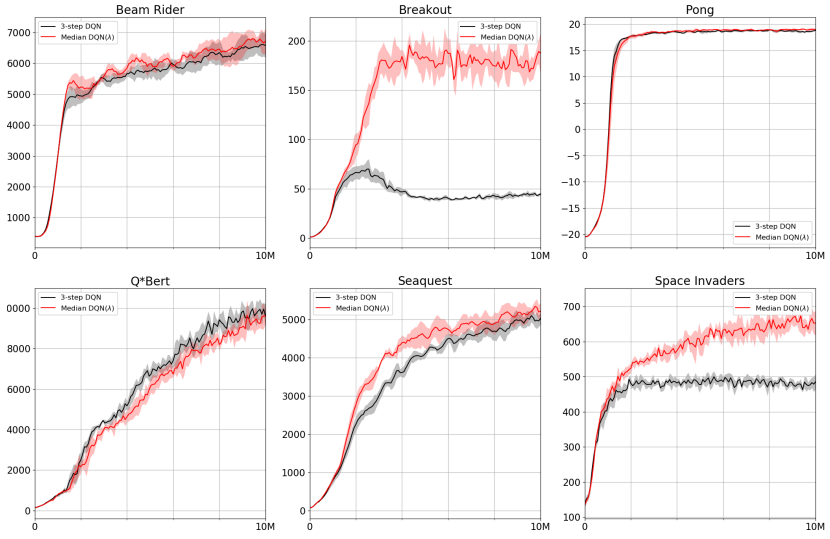

Figure 1: DQN(λ) with median-based λ selection against a 3-step target-network DQN baseline on six Atari games, trained for 10 million timesteps. Results were averaged over 5 random seeds and standard deviation is shown.