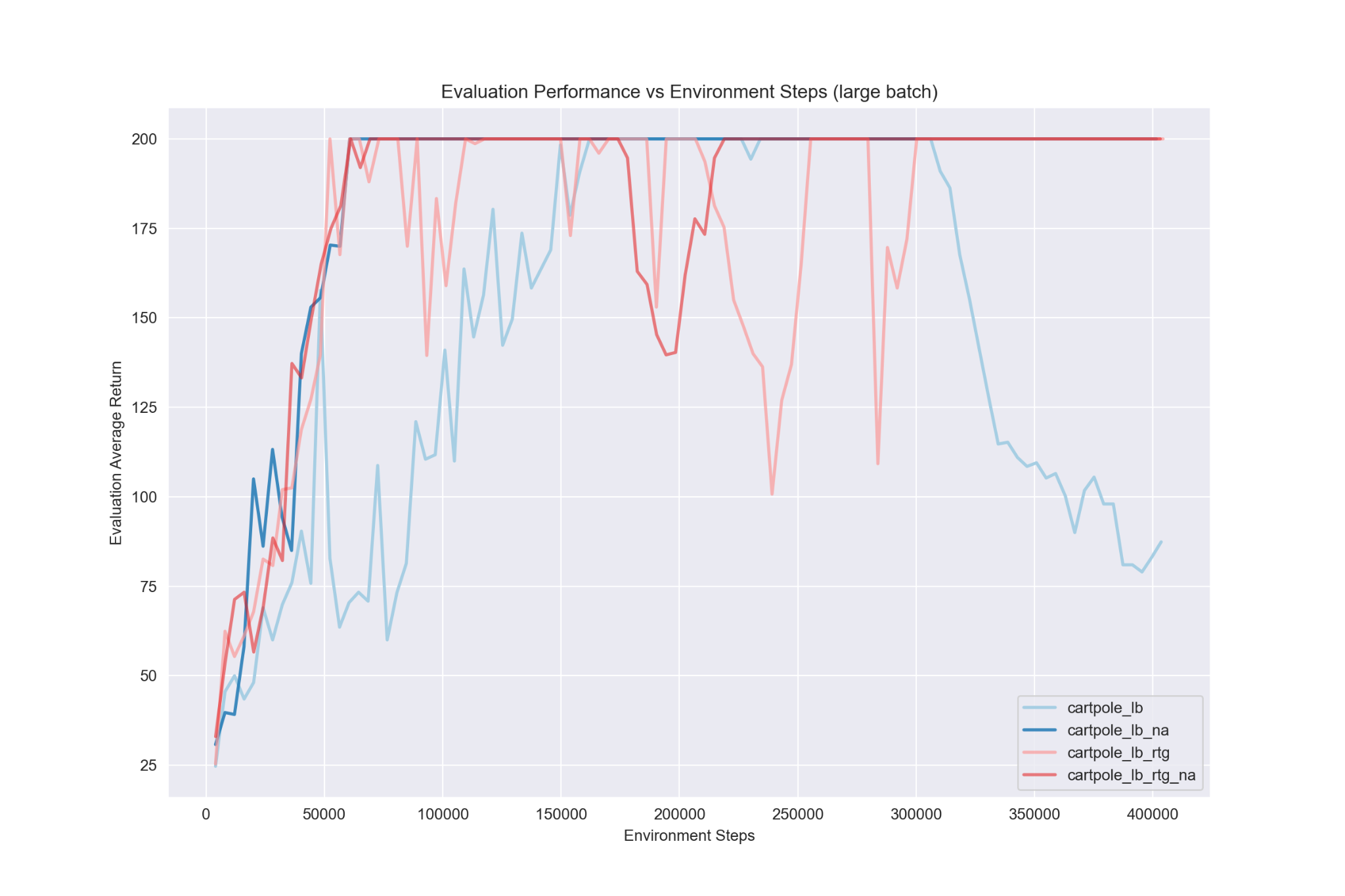
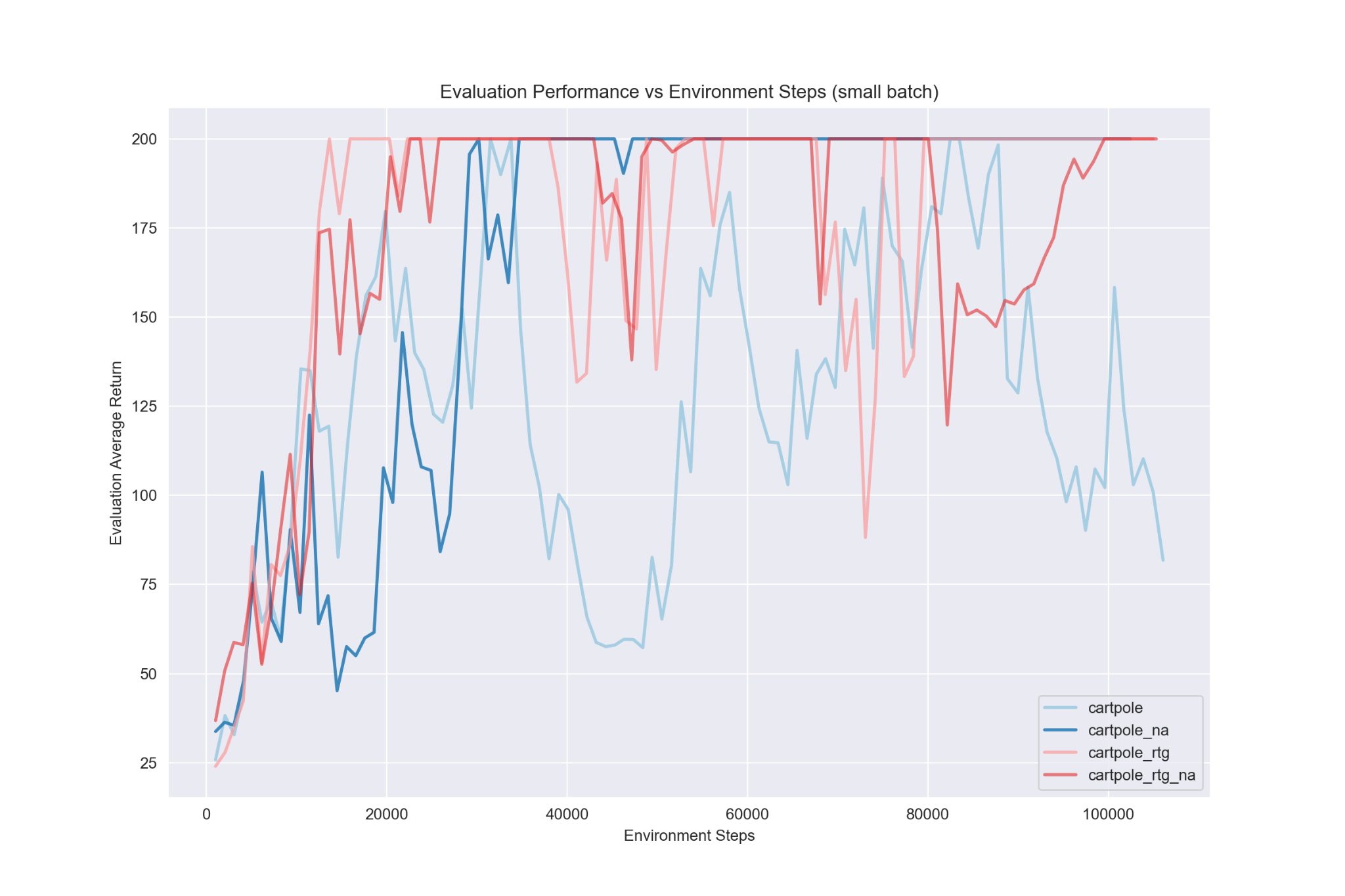
Class project 2

Matouš Bohoněk, 2025/04

# Experiment 1 (CartPole)



**Q: Which value estimator has better performance without advantage normalization: the trajectory-centric one, or the one using reward-to-go?**

A: As we can see, the one with reward-to-go has generally better performance. It achieves the maximum score of 200 much sooner and tends to stay at the score compared to the trajectory-centric which takes more time to reach the maximum score and isn’t able to stay at that score. This can mean that the actual policy learned might not be as good since it’s really easy to throw it off with just a single training step.

**Q: Did advantage normalization help?**

A: Yes, although when using reward-to-go it took a bit longer to get to the maximum score when also using advantage normalization, it later helped stay at this maximum score, suggesting that the learned policy was more stable and less prone to falling apart.

**Q: Did the batch size make an impact?**

A: Generally I don’t think that the bigger batch size was better or worse. What I wrote above regarding reward-to-go and advantage normalization applies to both batch sizes. When we take a look at the graphs, the relative learning curves are very similar. The main difference is that the bigger batch size required more environment steps to learn get the same score.

Here are the command line configurations:  
**cartpole:** -n 100 -b 1000

**cartpole\_rtg:** -n 100 -b 1000 -rtg

**cartpole\_na:** -n 100 -b 1000 -na

**cartpole\_rtg\_na:** -n 100 -b 1000 -rtg -na

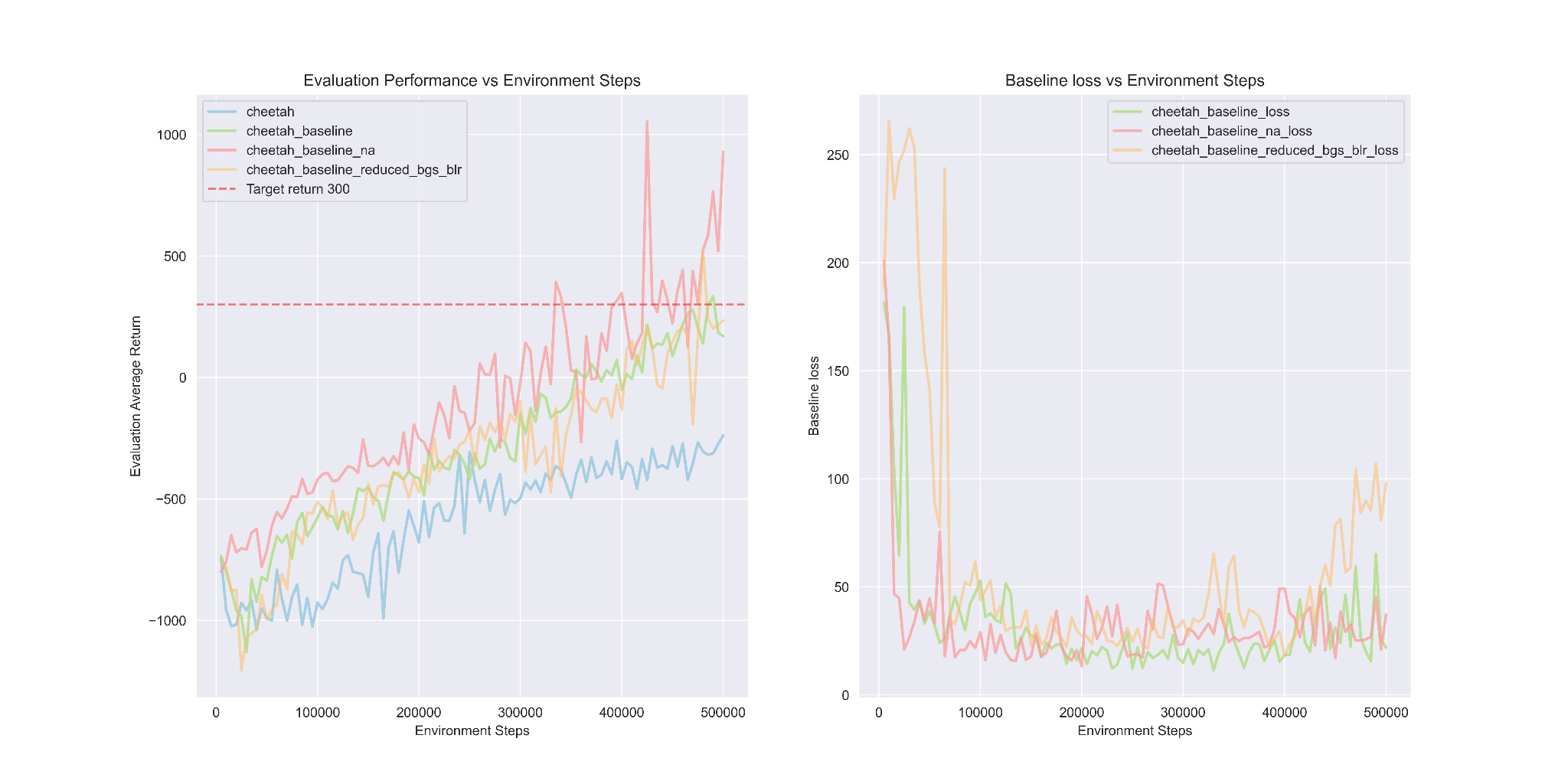
**cartpole\_lb:** -n 100 -b 4000

**cartpole\_lb\_rtg:** -n 100 -b 4000 -rtg

**cartpole\_lb\_na:** -n 100 -b 4000 -na

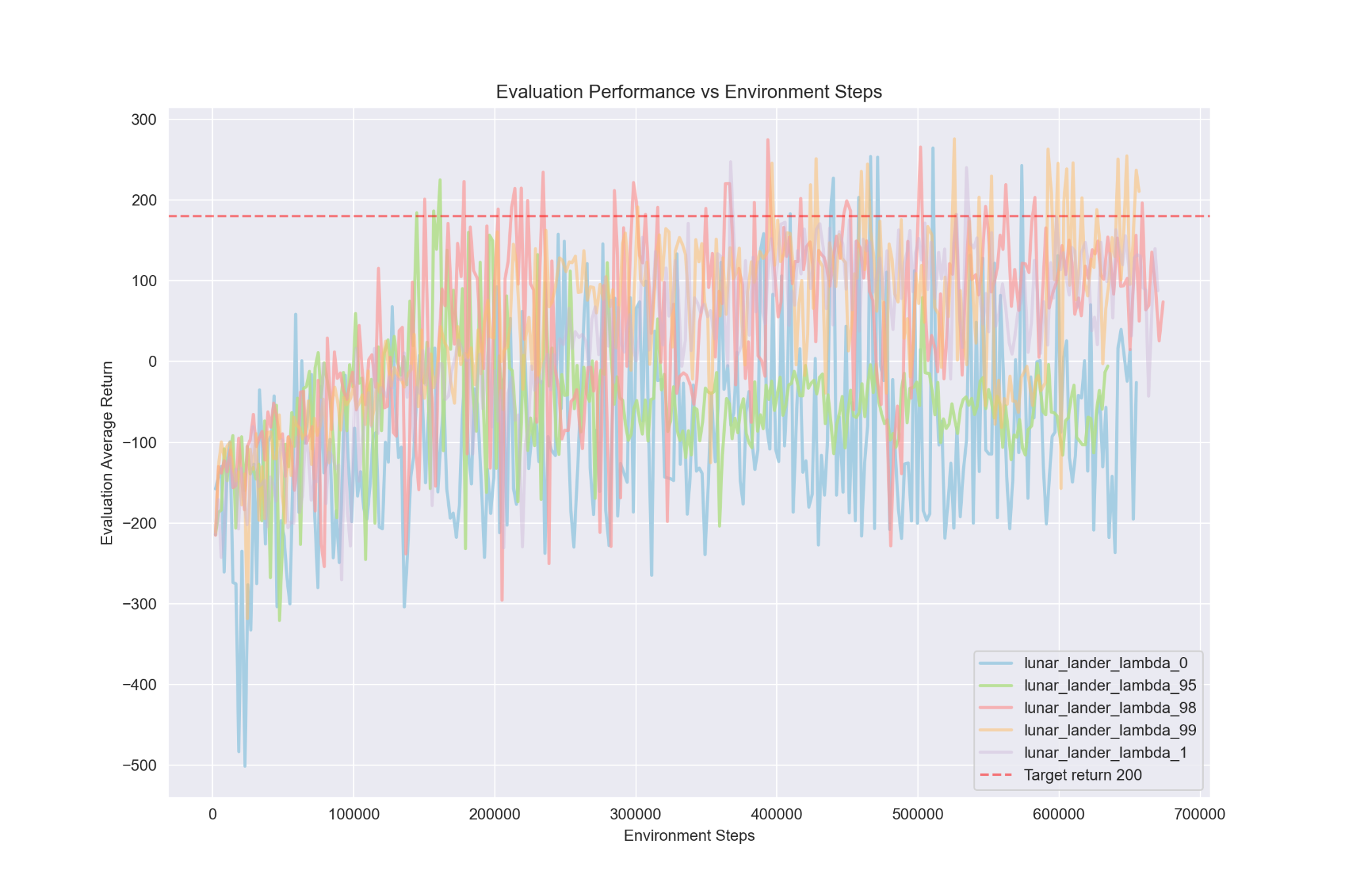
**cartpole\_lb\_rtg\_na:** -n 100 -b 4000 -rtg -na

# Experiment 2 (HalfCheetah)



As we can see the standard policy gradient wasn’t able to achieve positive scores. On the other hand the baselined version was able to achieve the expected return of just above 300. Reducing the learning rate and (from -blr 0.01 -bgs 5 to -blr 0.005 -bgs 3) resulted in a very similar performance in terms of policy, although the baseline loss ended up quite a bit worse. Lastly, adding advantage normalization helped dramatically, achieving a return of over 1000.

# Experiment 3 (LunarLander-v2)

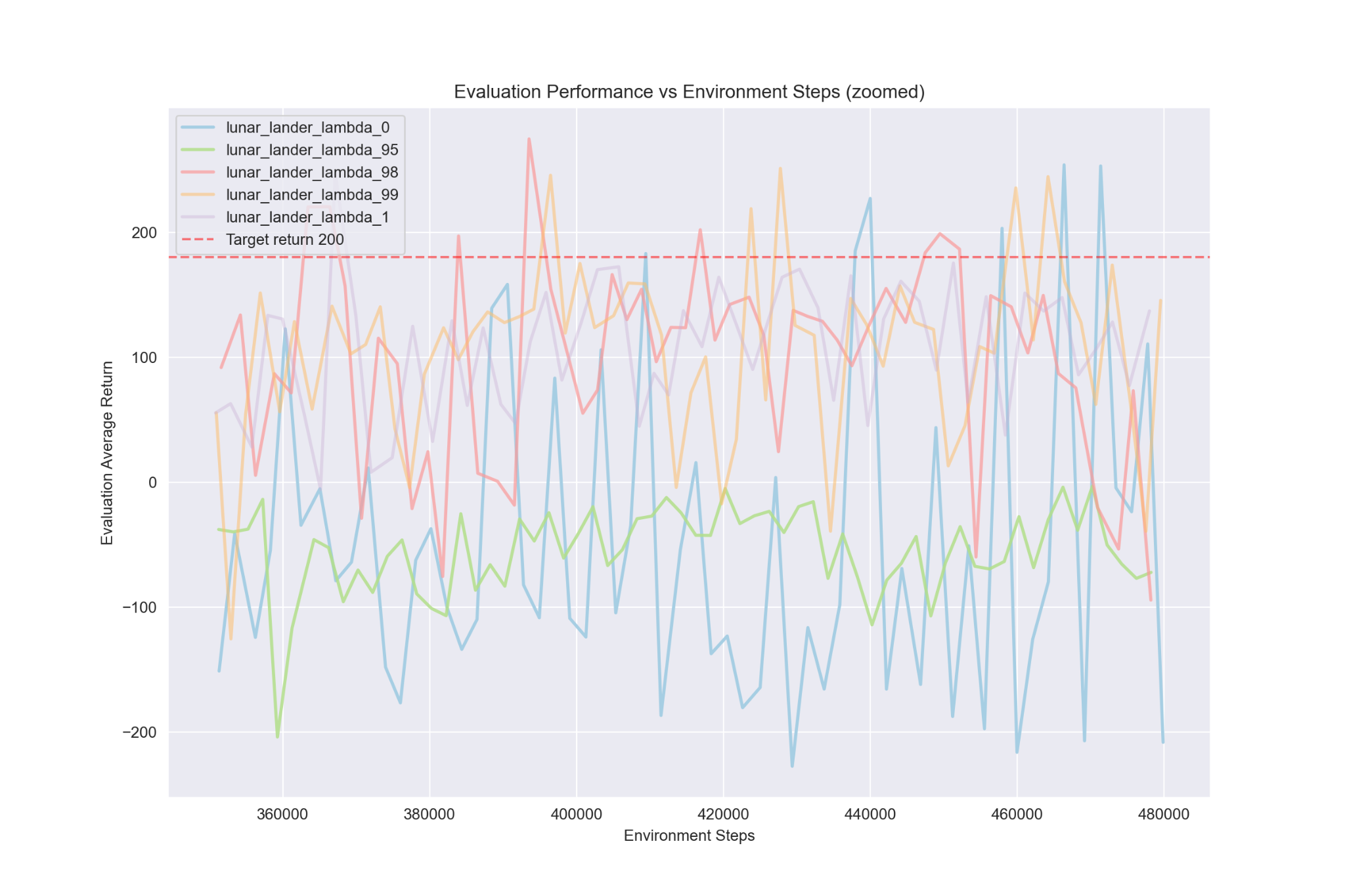


Setting lambda to 0 results in a lower variance but higher bias as it solely relies on bootstrapping:



Because when we follow the sum, the first item will be the standard bootstrapped estimate multiplied by (gamma \* lambda)^0, where if lambda = 0 we get 0^0 = 1. For all subsequent elements in the sum we multiply the delta\_t’ by 0.

On the other hand setting lambda to 1 will result in getting a standard single sample Monte Carlo estimate, having higher variance but lower bias.

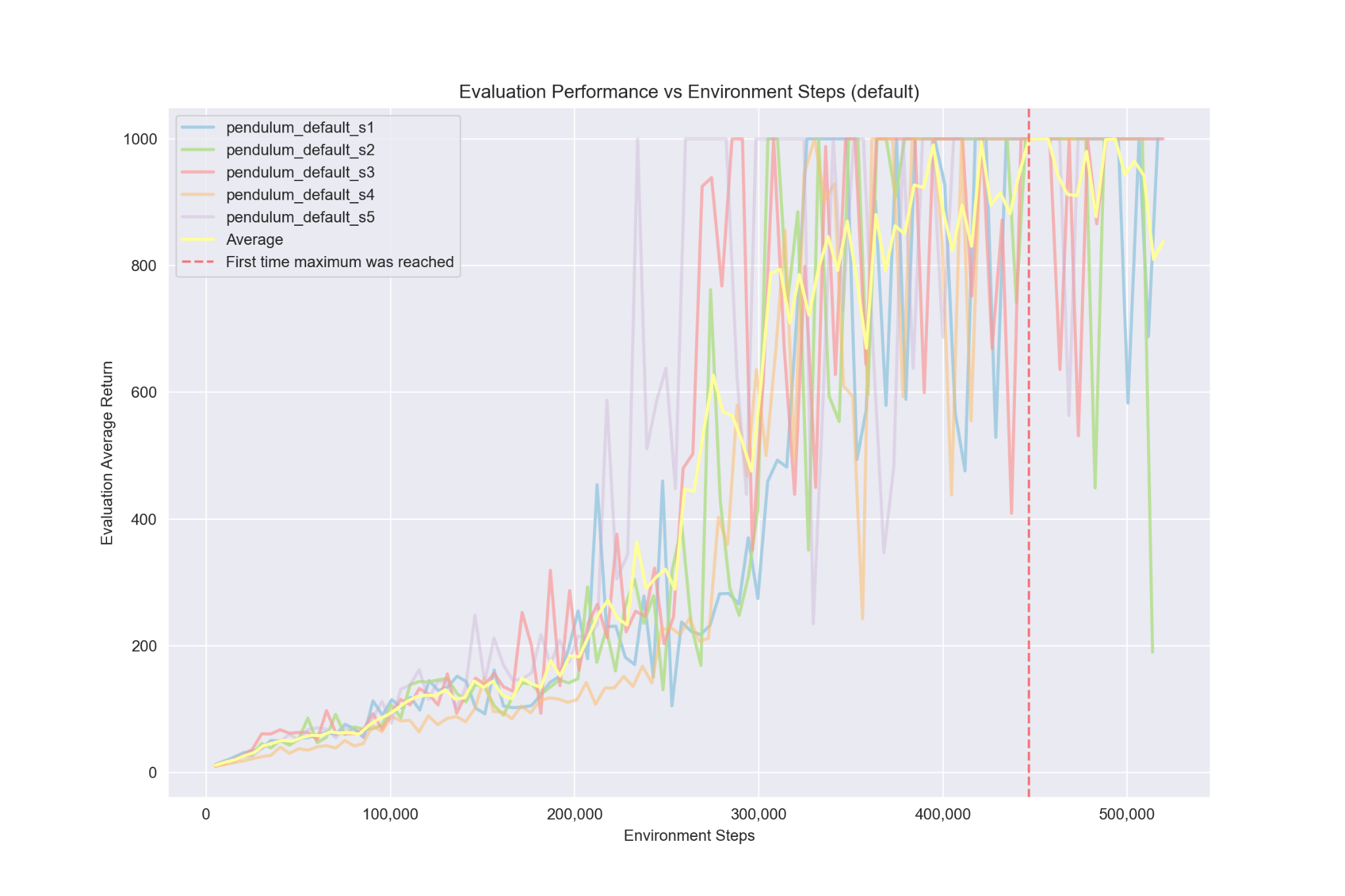


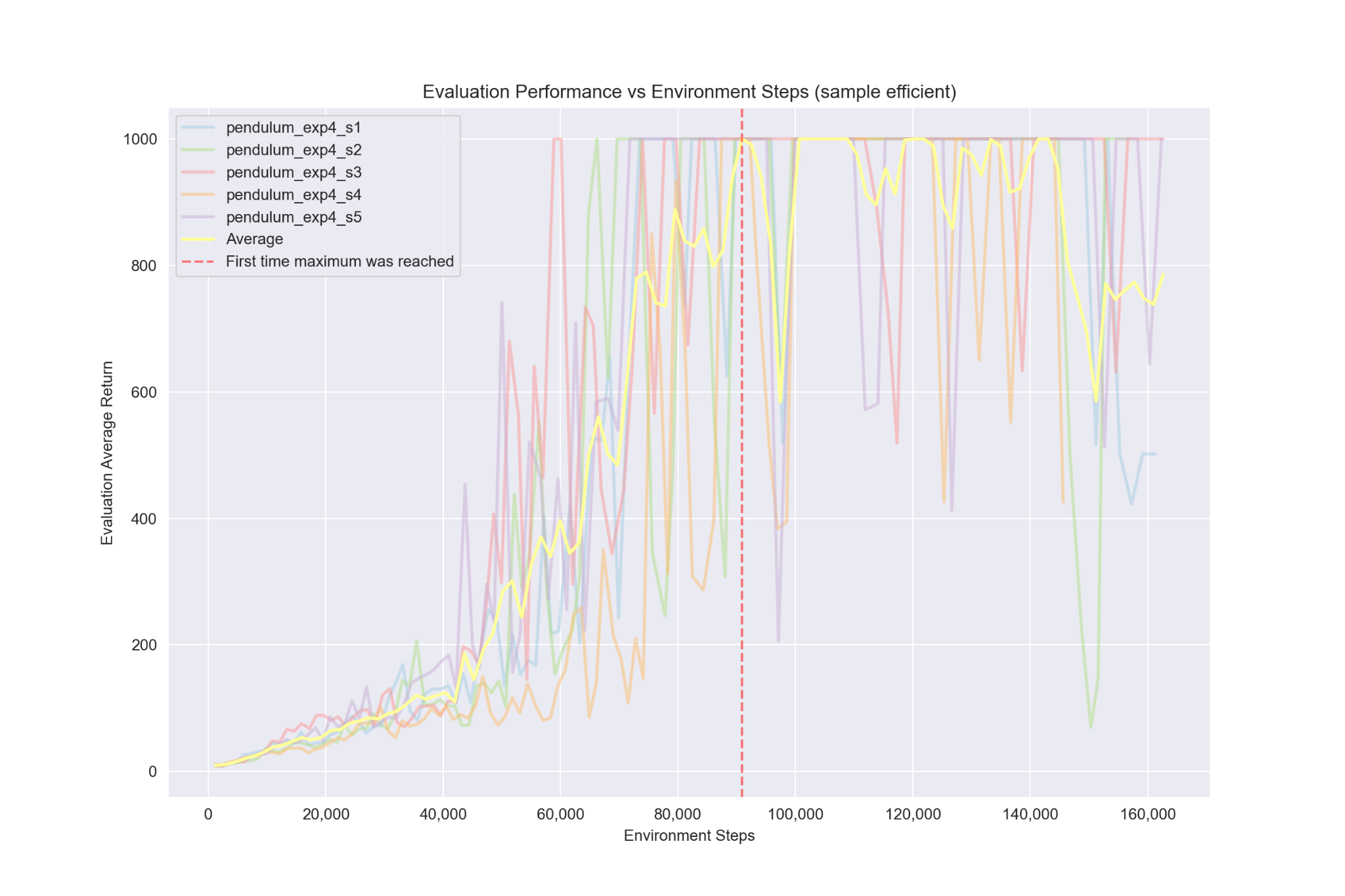
As we can see from the graph, all values of lambda were able to achieve the 200 evaluation threshold (lambda = 0.95 achieved this at a different time than we can see here), however if we take a look at the following graph where the learning curves are smoothed using rolling average:



We can see that using higher lambdas (lambda >=0.98) resulted in more stable results throughout the training process.

# Experiment 4 (InvertedPendulum)





When we change the hyperparameters from the default ones:

-rtg --use\_baseline -na --batch\_size 5000 —-discount 1 --n\_layers 2 —-layer\_size 64

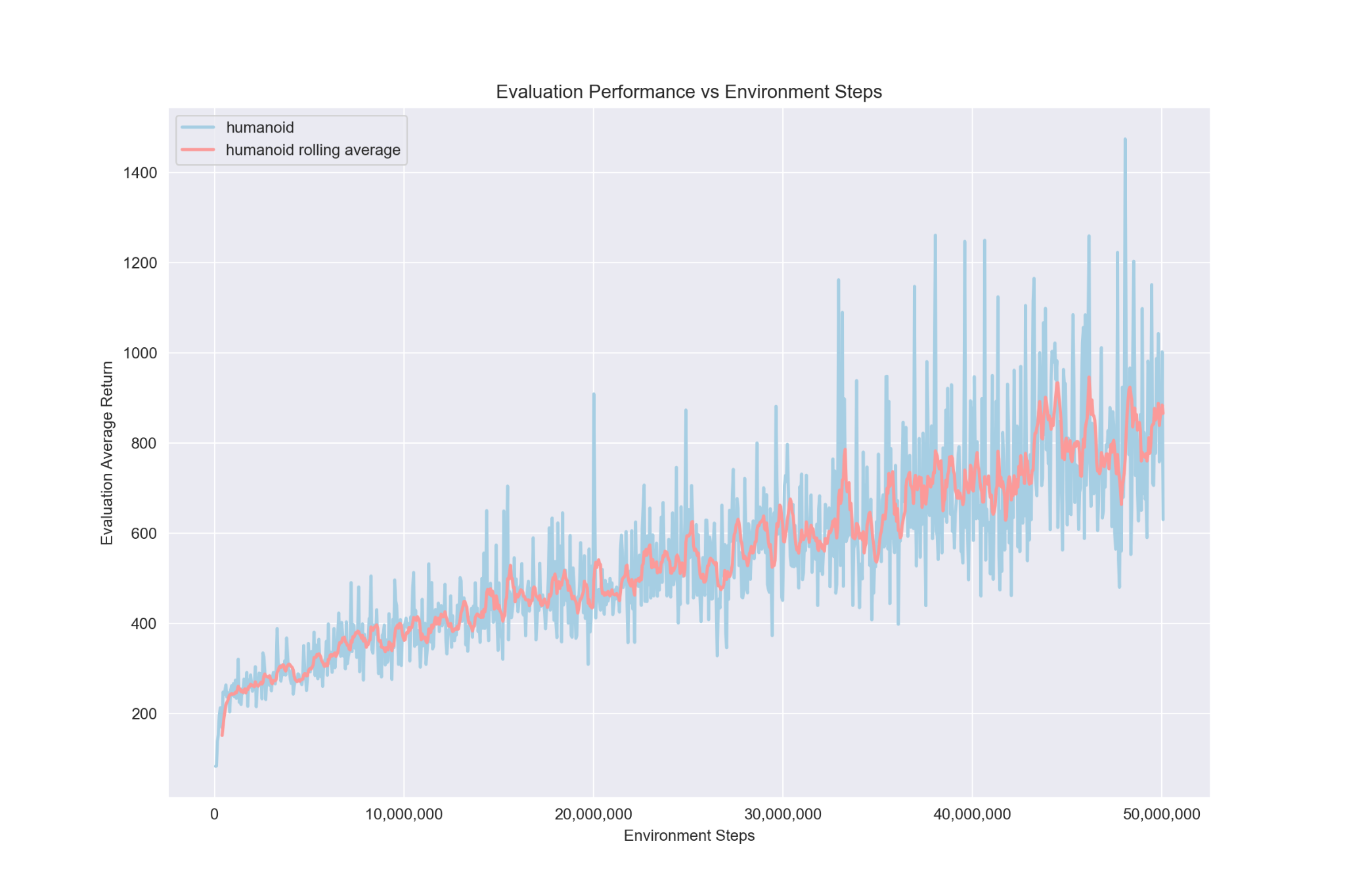
To these:

-rtg --use\_baseline -na --batch\_size 1200 --discount 0.98 --n\_layers 3 --layer\_size 36 —-lambda\_gae 0.98

We can converge to the maximum score much faster, at around 90,000 environment steps compared to the default’s 440,000 environment steps, which is almost a 5x improvement!

# Experiment 5 (Humanoid-v4)

Trying to learn the policy for the humanoid environment took just over 4 hours in total and we were able to get a stable evaluation return over 600, sometimes even achieving over 1000. As we can see, the required number of environment steps increased by about 10x compared to the experiments done above.



# Survey

I spent most time on the first part (Policy gradients), because it took me a while to get to know the code structure and because I had to fill in most of the code there. It could’ve taken me around 2 hours maybe. Running the experiments was usually quite fast, here all runs took under a minute.

Implementing part 4 (Baseline) could’ve taken me under an hour with some debugging. Running the code was just as fast as in part 3.

The last part (GAE) could’ve been the quickest to implement if I hadn’t made a small mistake that took me a while to debug. Each lunar lander experiment took around 4 minutes to run and I had to try 4 hyperparameter sets to achieve my final results. The humanoid experiment took 4 hours and 12 minutes to finish.

Writing the report and creating the graphs could’ve taken me around 1-2 hours total.

The assignment was quite interesting and mostly fun.