Weeks 3 & 4: Implementing an RL Training Pipeline

BEH Chuen Yang

Abstract

This document contains experiments on the CartPole-v0 Environment by Towers et al. (2024). We compare the performance of a simple policy gradient (PG) algorithm and discuss the viability of certain design choices in the training pipeline.

1 Recap

Refer to Levine et al. (2023) and Beh (2025) for a recap of the foundations of PG methods. Essentially, PG algorithms optimize policies to maximize the expected return via the following algorithm:

Algorithm 1 PG Algorithm

```
Input: Policy $\pi_\theta$, learning rate $\alpha$
Output: Updated policy $\pi_\theta$
Initialise policy parameters $\theta$
while not converged do
    Sample a batch of trajectories $T$ from the policy $\pi_\theta$ and the environment
    Estimate the PG $\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{n} r(s_t,a_t)\right)\left(\sum_{t=0}^{n}\nabla_\theta \log(\pi_\theta(a_t|s_t))\right)$
    Update the policy parameters $\theta \leftarrow \theta + \alpha\nabla_\theta J(\pi_\theta)$
end while
```

Notice however that this is not the only way to estimate the PG. Particularly, we can switch out the PG estimation step (*in red*) with more sophisticated gradient estimation rules in order to improve the stability and convergence of the algorithm.

Levine et al. (2023) discuss several such improvements, and for brevity, we will only write out the PG estimate for each improvement in isolation.

- Rewards To Go: $\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=k}^{n} r(s_t,a_t)\right)\left(\sum_{t=k}^{n}\nabla_\theta \log(\pi_\theta(a_t|s_t))\right)$ where k is the current timestep in the trajectory.
- Advantage Normalization: $\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{n} A(s_t,a_t)\right)\left(\sum_{t=0}^{n}\nabla_\theta \log(\pi_\theta(a_t|s_t))\right)$, where $A(s_t,a_t) = \frac{r(s_t,a_t)-\mu}{\sigma}$ is the advantage function.
- Reward Discounting: $\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{n} \gamma^t r(s_t,a_t)\right)\left(\sum_{t=0}^{n}\nabla_\theta \log(\pi_\theta(a_t|s_t))\right)$, where $\gamma < 1$ is the discount factor.
- Use of Baseline: $\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{n}(r(s_t,a_t)-b(s_t))\right)\left(\sum_{t=0}^{n}\nabla_\theta \log(\pi_\theta(a_t|s_t))\right)$
- Generalized Advantage Estimation (GAE) (Schulman et al. (2018)): $\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=k}^{n} A(s_t,a_t)\right)\left(\sum_{t=k}^{n}\nabla_\theta \log(\pi_\theta(a_t|s_t))\right)$ where k is the current timestep, $\gamma < 1$ is a discount factor, $A(s_t,a_t) = \sum_{l=k}^{\infty}(\gamma\lambda)^l\delta_{t+l}$, and $\delta_t = r(s_t,a_t) + \gamma V(s_{t+1}) - V(s_t)$ is the temporal difference error.

Since Levine et al. (2023) only consider a discount factor of $\gamma = 1$ (i.e. no discount) on a finite horizon task, we will experiment empirically with only the first two improvements using a task due to Towers et al. (2024).

Symbol	Description	Notes
s_t	(x, v, θ, ω)	x: Cart position v: Cart velocity θ: Pole angle ω: Pole angular velocity
s_0	$(x, v, \theta, \omega) \sim U(-0.05, 0.05)$	
a_t	[0, 1]	Left or Right.
$r(s_t, a_t, s_{t+1})$	+1	
Termination	When pole ”falls over”, or $t \geq 200$	

Table 1: CartPole-v0 Environment Overview

2 CartPole-v0 Environment

The CartPole-v0 environment by Towers et al. (2024) is modelled after a classic reinforcement learning task by Barto et al. (1983), where the goal is to balance a pole on a cart by applying forces to the left or right.

Table 1 summarizes the salient information about the environment.

3 Training Pipeline

3.1 Template Overview

We were first provided with a template by Levine et al. (2023) *partially* implementing a training pipeline for reinforcement learning (RL) algorithms. The template includes a basic environment wrapper, a simple PG algorithm, and a training loop that collects data from the environment and updates the policy using the collected data.

3.2 Implementation

The template was missing most of the components for a theoretical PG algorithm.

- Trajectory Collection: The template only partially defined the collection of a single trajectory. By extension, we cannot collect multiple trajectories.
- State-Value Estimation: The template did not calculate Q-values out-of-the-box.
- Advantage Estimation: The template did not calculate advantages out-of-the-box.
- Policy: While a network was built, the forward-pass and decision-making were not implemented.
- Update Loop: Loss Calculation and backpropagation were not implemented.

The specific changes made to the template have been detailed in the README of the edited template, which can be found at code/wk3/README.md.

4 Experiments

In keeping with Levine et al. (2023), we ran the training pipeline on the *CartPole-v0* environment in order to study performance and convergence of the various PG algorithms using 1 (and optionally, either rewards-to-go (RTG) or advantage normalization (AN) as in 1).

For each algorithm variant, we ran trials with 10 different random seeds and recorded the average train return over 200 sampling steps.

The training script for this experiment can be found at code/wk3/train.py.

The average returns for each algorithm variant across *ALL* seeds is shown in Figure 1.

Algorithm Variants	RTG	AN
PG	✗	✗
PG + RTG	✓	✗
PG + AN	✗	✓
PG + RTG and AN	✓	✓

Table 2: PG Algorithms Tested

Hyperparameter	Value
Trial Seeds	[941, 2370, 3295, 4495, 4943, 5653, 6548, 8497, 9127, 9540]
Batch Size	1000, 4000
Sampling Iterations	200

Table 3: Hyperparameters for the Experiments

5 Discussion

5.1 Theoretical Expectations

Due to the discussion of improvements to the PG algorithm in Levine et al. (2023) and Beh (2025), we would expect the following trend:

- PG should perform the worst, as it does not use any of the improvements.
- PG + RTG and PG + AN should perform better than PG since both techniques reduce variance in the gradient estimate, but we cannot say which will perform better.
- PG + RTG + AN performs the best, as it combines both improvements.

Moreover, by statistical intuition, we would expect the PG algorithm to work more reliably with a larger batch size, which lends itself to more stable gradient estimates and hence a smoother learning curve.

5.2 Empirical Results

The empirical results in Figure 1 show that:

- PG performs the worst, as expected.
- PG + RTG performs better than PG, but not as well as PG + AN, or PG + RTG + AN.
- Surprisingly, PG + AN performs better than PG + RTG + AN.
- A larger batch size leads to a smoother learning curve.

[1]

We offer some conjectures as to why this is the case in the following subsections.

5.2.1 *Impact of Advantage Normalization*

Advantage normalization (AN) predictably improves the performance of the PG algorithm. By directly reducing the variance in the gradient estimate (through the advantage term), the algorithm takes more stable steps in the direction of the gradient, which *CAN* lead to better convergence properties. [2]

[1] Detailed breakdowns of the performance of each algorithm variant across each individual seed and experiment type can be found in the appendices. See B and C respectively.

[2] This is not necessarily the case. See Andrychowicz et al. (2020) for data to the contrary.

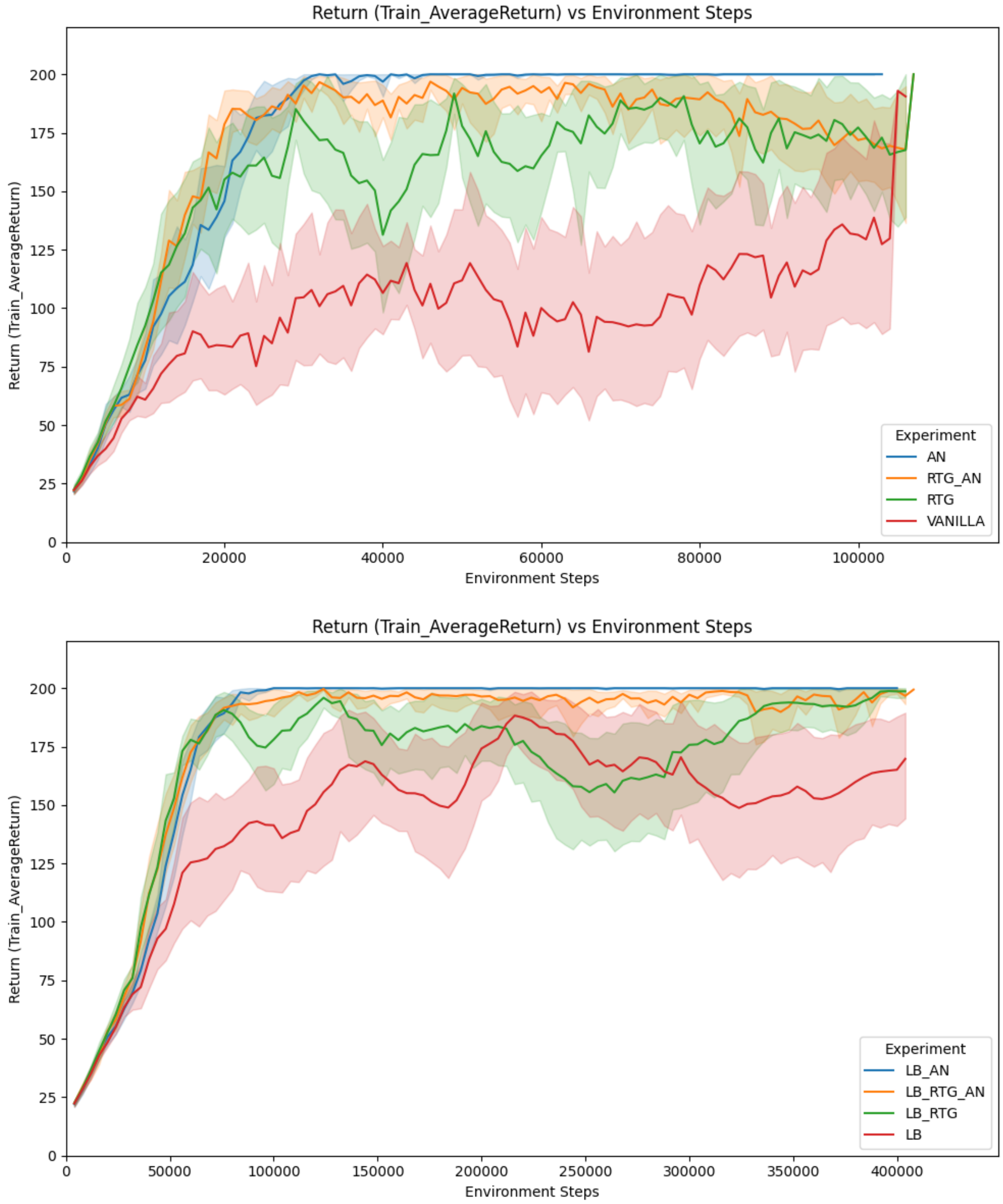

Figure 1: Average Returns for Each Algorithm Variant. For convenience of presentation, each datapoint was rounded to the nearest batchsize.
(Above) Small Batch Size (1000). (Below) Large Batch Size (4000).

5.2.2 *Impact of Rewards-to-Go*

Likewise, the PG + RTG algorithm performs better than the PG algorithm, and it is likely due to the fact that RTG reduces the variance in the gradient estimate, as discussed in Levine et al. (2023) and Beh (2025).

5.2.3 *Interplay of RTG and AN*

However, combining RTG and AN in the PG + RTG + AN algorithm yields a lower performance boost than would be suggested by the sum of the individual improvements.

This is perhaps unique to our setup, which does not incorporate a value function baseline. Specifically, we suspect that normalization of the RTG induces a bias towards earlier steps in the trajectory.

Since we are blindly averaging the returns *from the current timestep onwards* across all steps in the trajectory, and our reward function is strictly non-negative on a per-timestep basis, we neglect the fact that later timesteps are likelier to have lower returns as a result.

Moreover, as the time dimension is not in the agent's state, the agent does not learn why it receives lower returns at later timesteps.

Put together, this means that the agent will be unable to model the time-dependent aspects of the environment, and will optimize for the earlier timesteps in the trajectory, which are more likely to have higher returns. [3] [4]

This causes the agent to learn a policy that is biased towards earlier timesteps, which can lead to suboptimal performance.

5.2.4 *Impact of Batch Size*

As expected, a larger batch size leads to better performance as evidenced in Figure 1. Every experiment with a larger batch size, at least when we look at the percentage-wise progress of the average return, is better than the corresponding experiment with a smaller batch size.

5.3 Miscellaneous Observations

In addition to the above observations, we also note a few points of interest when running the experiments:

- The PG algorithm is very sensitive to the choice of random seed, perhaps due to the high variance in the gradient estimate.
- We occasionally observe extended periods of performance degradation (more obvious in Figure **??**), where the average return drops appreciably before recovering.
    - This is possibly due to *catastrophic forgetting* (Goodfellow et al. (2015)), which occurs as a consequence of the online nature of the algorithm, where collected data is always discarded after each update.
    - Notably, since we can even see it in the averaged returns in Figure 1, this proves to be a relatively common occurrence.

6 Conclusion

In this report, we have completed a training pipeline for a simple PG algorithm and run experiments on the CartPole-v0 environment, with the goal of studying the performance of various tweaks to the simplest version of the PG algorithm.

We have observed most significantly that the PG algorithm is sensitive to the choice of random seed, that larger batch sizes can improve and destabilise PG algorithms' performance, and that, at least without a baseline, advantage normalization (AN) is a more consistent and effective improvement than rewards-to-go (RTG). However, AN and RTG appear to synergize poorly.

[3] This can be alleviated by including a time dimension in the agent's state. However, it appears that this is not common practice (at least, within Towers et al. (2024)).

[4] A more permanent solution is to extend the environment into the infinite horizon case. This can be done by bootstrapping rewards with a value function, an approach best highlighted in temporal difference learning (TD-Learning), as exemplified in Sutton (1988), and Schulman et al. (2018).

References

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020. URL https://arxiv.org/abs/2006.05990.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13 (5):834–846, 1983.

Chuen Yang Beh. Week 2: Foundations of policy gradient methods, 2025.

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015. URL https://arxiv.org/abs/1312.6211.

Sergey Levine, Kyle Stachowicz, Vivek Myers, Joey Hong, and Kevin Black, 2023. URL https://rail.eecs.berkeley.edu/deeprlcourse/deeprlcourse/static/homeworks/hw2.pdf.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.

Appendix

A Constant Hyperparameters

In addition to the hyperparameters listed in Table 3, the following hyperparameters were kept constant throughout the experiments:

Hyperparameter	Value
α	0.01
γ	1.0
Policy Network Architecture	2 Linear layers with 64 units each, ReLU activation
Optimizer	Adam
Optimizer Betas	(0.9, 0.999)

B Individual Seed Plots

The following plots show the average return for each algorithm variant across each individual seed.

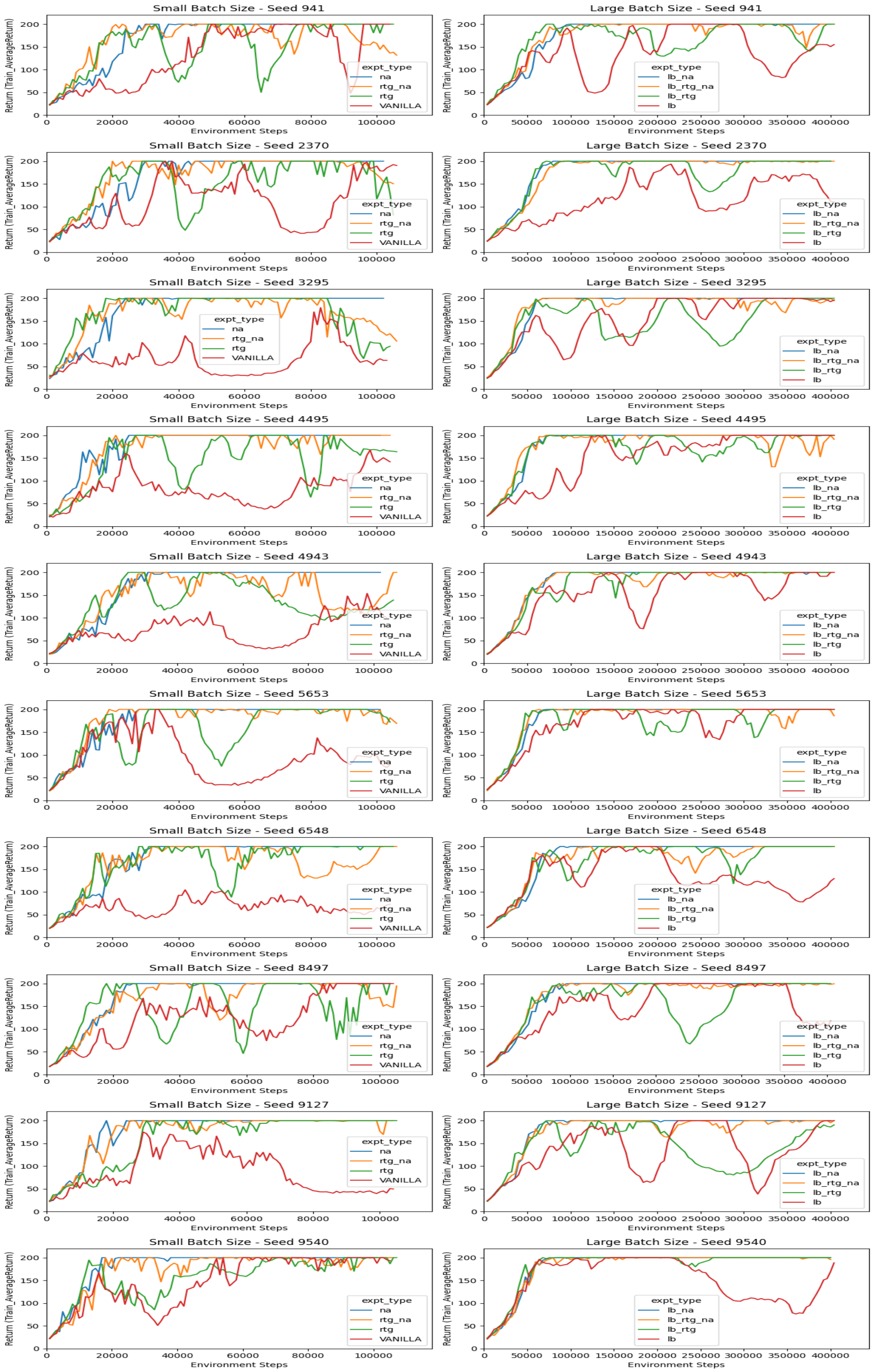

Figure 2: Returns for Each Algorithm Variant by Seed. As with Figure 1, each datapoint was rounded to the nearest batch size for convenience of presentation.

C Experiment Type Plots

The following plots show the average return for each algorithm variant across each experiment type.

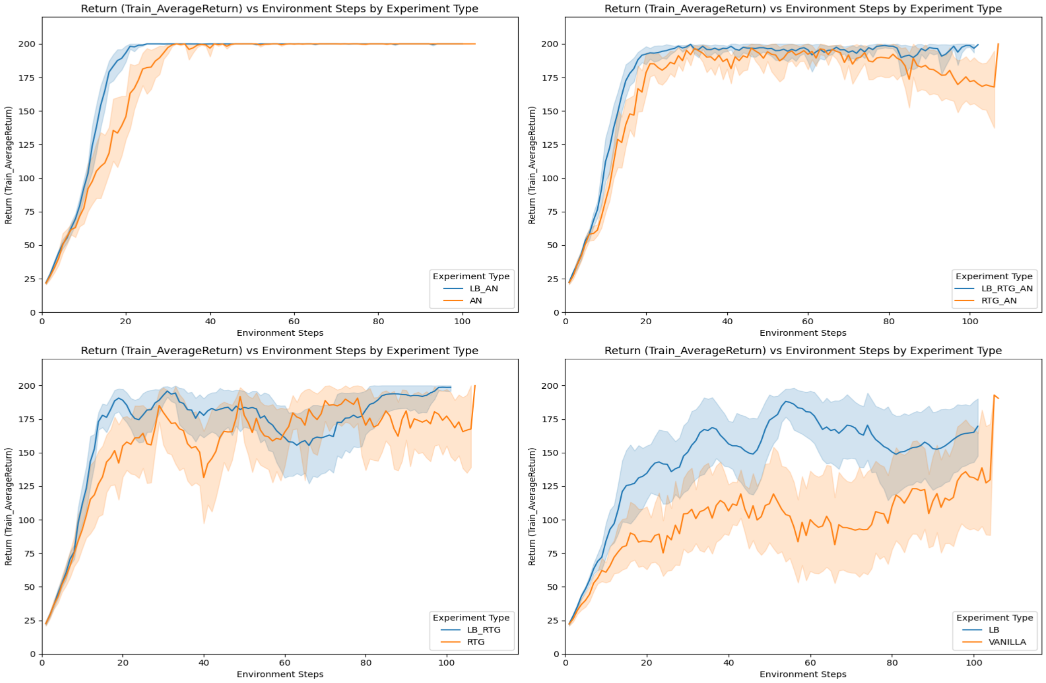

Figure 3: Returns for Each Algorithm Variant by Experiment Type. As with Figure 1, each datapoint was rounded to the nearest batch size for convenience of presentation.