A. Algorithmic Baselines

A.1. PPO

Proximal policy optimization (PPO) [54] is an on-policy algorithm that is designed to improve the stability and sample efficiency of policy gradient methods, which uses a clipped surrogate objective function to avoid large policy updates.

The policy loss is defined as:

$$L_\pi(\boldsymbol{\theta}) = -\mathbb{E}_{\tau\sim\pi}\left[\min\left(\rho_t(\boldsymbol{\theta})A_t, \operatorname{clip}\left(\rho_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon\right)A_t\right)\right], \tag{9}$$

where

$$\rho_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a}_t|\boldsymbol{s}_t)}, \tag{10}$$

and ϵ is a clipping range coefficient.

Meanwhile, the value network is trained to minimize the error between the predicted return and a target of discounted returns computed with generalized advantage estimation (GAE) [53]:

$$L_V(\boldsymbol{\phi}) = \mathbb{E}_{\tau\sim\pi}\left[\left(V_{\boldsymbol{\phi}}(\boldsymbol{s}) - V_t^{\text{target}}\right)^2\right]. \tag{11}$$

A.2. Random Search

Random search (RS) [11] is a simple yet effective method for hyperparameter optimization that randomly samples from the configuration space instead of exhaustively searching through all combinations. Compared to grid search, RS is particularly efficient in high-dimensional spaces, where it can outperform grid search by focusing on a wider area of the search space. It is especially beneficial when only a few hyperparameters significantly influence the model's performance, as it can effectively explore these critical dimensions without the computational cost of grid search. RS is also highly parallelizable and flexible, allowing for dynamic adjustments to the search process.

A.3. PBT

Population-based training (PBT) [33] is an asynchronous optimization method designed to optimize both model parameters and hyperparameters simultaneously. Unlike traditional hyperparameter tuning methods that rely on fixed schedules for hyperparameters, PBT adapts hyperparameters during training by exploiting the best-performing models and exploring new hyperparameter configurations. PBT operates by maintaining a population of models and periodically evaluating their performance, and using this information to guide the optimization of hyperparameters and model weights. This approach ensures efficient use of computational resources while achieving faster convergence and improved final performance, particularly in RL and generative modeling tasks.

A.4. PB2

Population-based bandits (PB2) [27] enhances PBT by using a multi-armed bandit approach to dynamically select and optimize hyperparameters based on performance. This method improves the exploration-exploitation trade-off, allocating resources to the most promising configurations and reducing computational costs while accelerating convergence.

A.5. SMAC+HB

SAMC [40] is a powerful framework for hyperparameter optimization that leverages Bayesian optimization (BO) to efficiently find well-performing configurations. It uses a random forest model as a surrogate to predict the performance of hyperparameter configurations, which is particularly effective for high-dimensional spaces. SMAC optimizes the hyperparameters of machine learning algorithms by iteratively selecting configurations based on a probabilistic model of the objective function. Additionally, SMAC integrates with Hyperband [39] for more efficient resource allocation.

A.6. TMIHF

TMIHF [49] introduces a novel approach for optimizing both continuous and categorical hyperparameters in reinforcement learning (RL). This method builds on the population-based bandits (PB2) [48] framework and addresses its limitation of only handling continuous hyperparameters. By employing a time-varying multi-armed bandit algorithm, TMIHF efficiently selects both continuous and categorical hyperparameters in a population-based training setup, thereby improving sample efficiency and overall performance. The algorithm's hierarchical structure allows it to model the dependency between categorical and continuous hyperparameters, which is crucial for tasks like data augmentation in RL environments.

B. Experimental Setting

B.1. Arcade Learning Environment

Figure 11. Screenshots of the ALE-5 environments. From left to right: *BattleZone*, *DoubleDunk*, *NameThisGame*, *Phoenix*, and *Q*Bert*.

PPO+ULTHO. In this part, we utilize the implementation of [38] for the PPO algorithm, and train the agent for 10M environment steps in each environment. For the hyperparameter clusters, we select the batch size, value loss coefficient, and entropy loss coefficient as the candidates, and the detailed values of each cluster are listed in Table 2. Additionally, we run a grid search over the exploration coefficient $c \in \{1.0, 5.0\}$ and the size of the sliding window used to compute the Q-values $W \in \{10, 50, 100\}$ to study the robustness of ULTHO. Finally, Table 3 illustrates the PPO hyperparameters, which remain fixed throughout all the experiments except for the hyperparameter clusters.

PPO+Relay-ULTHO. At the end of the **PPO+ULTHO** experiments, we count the number of times each cluster is selected and find out the cluster of interest and the neglected cluster of interest. Then we perform the experiments with the two clusters separately before reporting the best-performing cluster. Therefore, the actual training budget of Relay-ULTHO is three times that of ULTHO, *i.e.*, 30M environment steps. Similarly, we run a grid search over the exploration coefficient and the sliding window size and report the best results.

HPO Baselines. For RS, PBT, SMAF, and SMAF+HB, we utilize the implementations provided in ARLBench [8], which is a benchmark for hyperparameter optimization in RL and allows comparisons of diverse approaches. The training budget for each method is 320M environment steps with five runs, in which each configuration is evaluated on three random seeds. The results reported in this paper are directly obtained from the provided dataset in the ARLBench.

B.2. Procgen

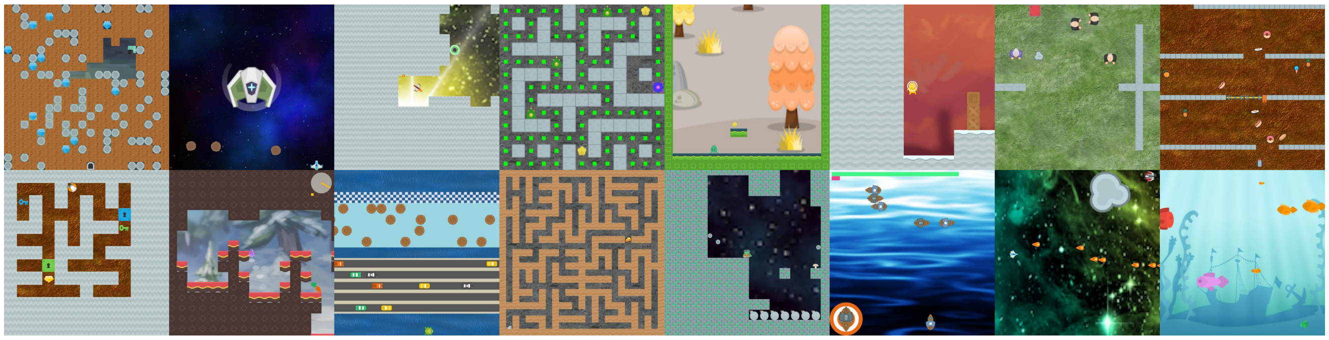

Figure 12. Screenshots of the sixteen Procgen environments.

PPO+ULTHO. In this part, we utilize the implementation of CleanRL for the PPO algorithm and train the agent for 25M environment steps on 200 levels before testing it on the full distribution of levels. For the hyperparameter clusters, we select the batch size, value loss coefficient, entropy loss coefficient, and number of update epochs as the candidates, and the detailed values of each cluster are listed in Table 2. Similarly, we run a grid search over the exploration coefficient $c \in \{1.0, 5.0\}$ and the size of the sliding window used to compute the Q-values $W \in \{10, 50, 100\}$ to study the robustness of ULTHO. Finally, Table 3 illustrates the PPO hyperparameters, which remain fixed throughout all the experiments except for the hyperparameter clusters.

PPO+Relay-ULTHO. Similar to the ALE experiments, we identify the two clusters of interest at the end of the ULTHO experiments. Then we also perform the experiments with the two clusters separately before reporting the best-performing cluster. The actual training budget of Relay-ULTHO is 75M environment steps for Procgen. Finally, we run a grid search over the exploration coefficient and the sliding window size and report the best results.

HPO Baselines. For PBT, PB2, and TMIHF, we leverage the official implementations reported in [49]. For each method, the population size is set as 4, and each is trained for 25M environment steps. Therefore, the total training budget is 100M environment steps. The results reported in this paper are directly obtained from the [49].

B.3. MiniGrid

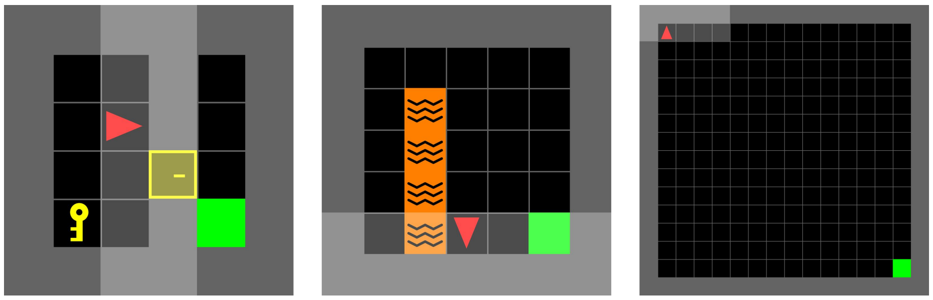
Figure 13. Screenshots of the three MiniGrid environments. From left to right: *DoorKey-6×6*, *LavaGapS7*, and *Empty-16×16*.

In this part, we use the implementation of [15] for the PPO agent and train each agent for 500K environment steps. For the hyperparameter clusters, we select the learning rate, batch size, and value loss coefficient as the candidates, and the detailed values of each cluster are listed in Table 2. The experiment workflow of ULTHO and Relay-ULTHO is the same as the experiments above. Finally, Table 3 illustrates the PPO hyperparameters, which remain fixed throughout all the experiments except for the hyperparameter clusters.

B.4. PyBullet

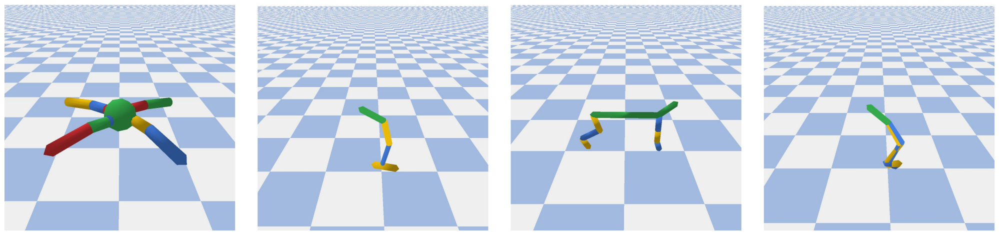
Figure 14. Screenshots of the four PyBullet environments. From left to right: *Ant*, *Hopper*, *HalfCheetah*, and *Walker2D*.

Finally, we perform the experiments on the PyBullet benchmark using the PPO implementation of [38], and train each agent for 2M environment steps. Here, we leverage state-based observation rather than image-based observations. For the hyperparameter clusters, we select the learning rate, batch size, and value loss coefficient as the candidates, and the detailed values of each cluster are listed in Table 2. We also run experiments for both ULTHO and Relay-ULTHO algorithms and report the best results. Likely, Table 3 illustrates the PPO hyperparameters, which remain fixed throughout all the experiments except for the hyperparameter clusters.

HP Cluster	**ALE**	**Procgen**	**MiniGrid**	**PyBullet**
Learning Rate	N/A	{2.5e-4, 5e-4, 1e-3}	{1e-3, 2.5e-3, 5e-3}	{2e-4, 5e-4, 7e-4}
Batch Size	{128, 256, 512}	{512, 1024, 2048}	{128, 256, 512}	{64, 128, 256}
Vale Loss Coefficient	{0.25, 0.5, 1.0}	{0.25, 0.5, 1.0}	{0.25, 0.5, 1.0}	{0.25, 0.5, 1.0}
Entropy Loss Coefficient	{0.01, 0.05, 0.1}	N/A	N/A	N/A
Number of Update Epochs	N/A	{3, 2, 1}	N/A	N/A

Table 2. The selected hyperparameter clusters for each benchmark.

Hyperparameter	**ALE**	**Procgen**	**MiniGrid**	**PyBullet**
Observation downsampling	(84, 84)	(64, 64, 3)	(7,7,3)	N/A
Observation normalization	/ 255.	/ 255.	No	Yes
Reward normalization	Yes	Yes	No	Yes
LSTM	No	No	No	No
Stacked frames	4	No	No	N/A
Environment steps	10000000	25000000	500000	2000000
Episode steps	128	256	128	2048
Number of workers	1	1	1	1
Environments per worker	8	64	16	1
Optimizer	Adam	Adam	Adam	Adam
Learning rate	2.5e-4	5e-4	1e-3	2e-4
GAE coefficient	0.95	0.95	0.95	0.95
Action entropy coefficient	0.01	0.01	0.01	0
Value loss coefficient	0.5	0.5	0.5	0.5
Value clip range	0.2	0.2	0.2	N/A
Max gradient norm	0.5	0.5	0.5	0.5
Batch size	256	2048	256	64
Discount factor	0.99	0.999	0.99	0.99

Table 3. The PPO hyperparameters for the four benchmarks. These remain fixed for all experiments except for the selected clusters.

C. Learning Curves

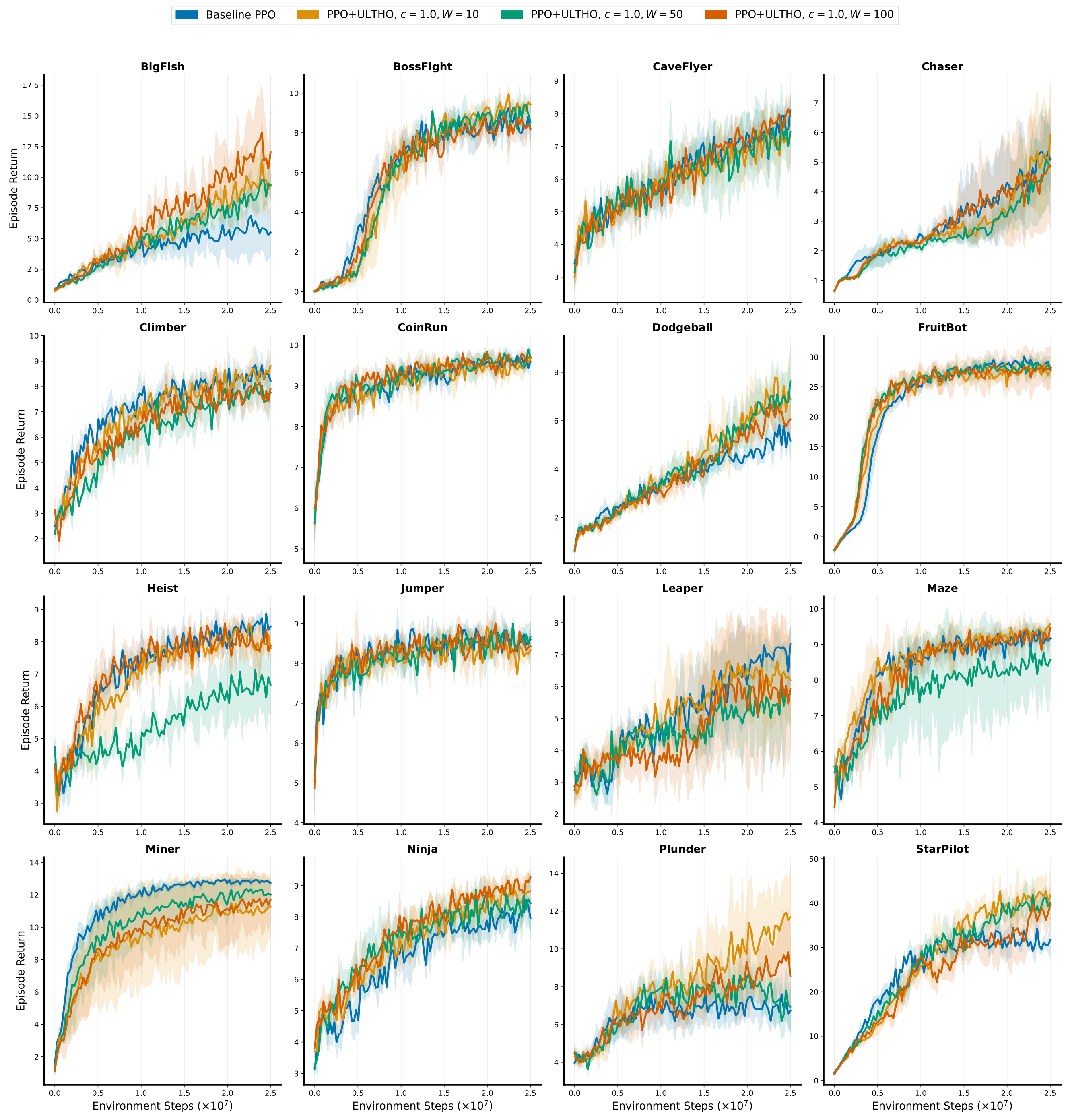

Figure 15. Learning curves of the vanilla PPO agent and ULTHO with different sizes of the sliding window on the Procgen benchmark. Here, the exploration coefficient c is set as 1.0. The mean and standard deviation are computed over five runs with different seeds.

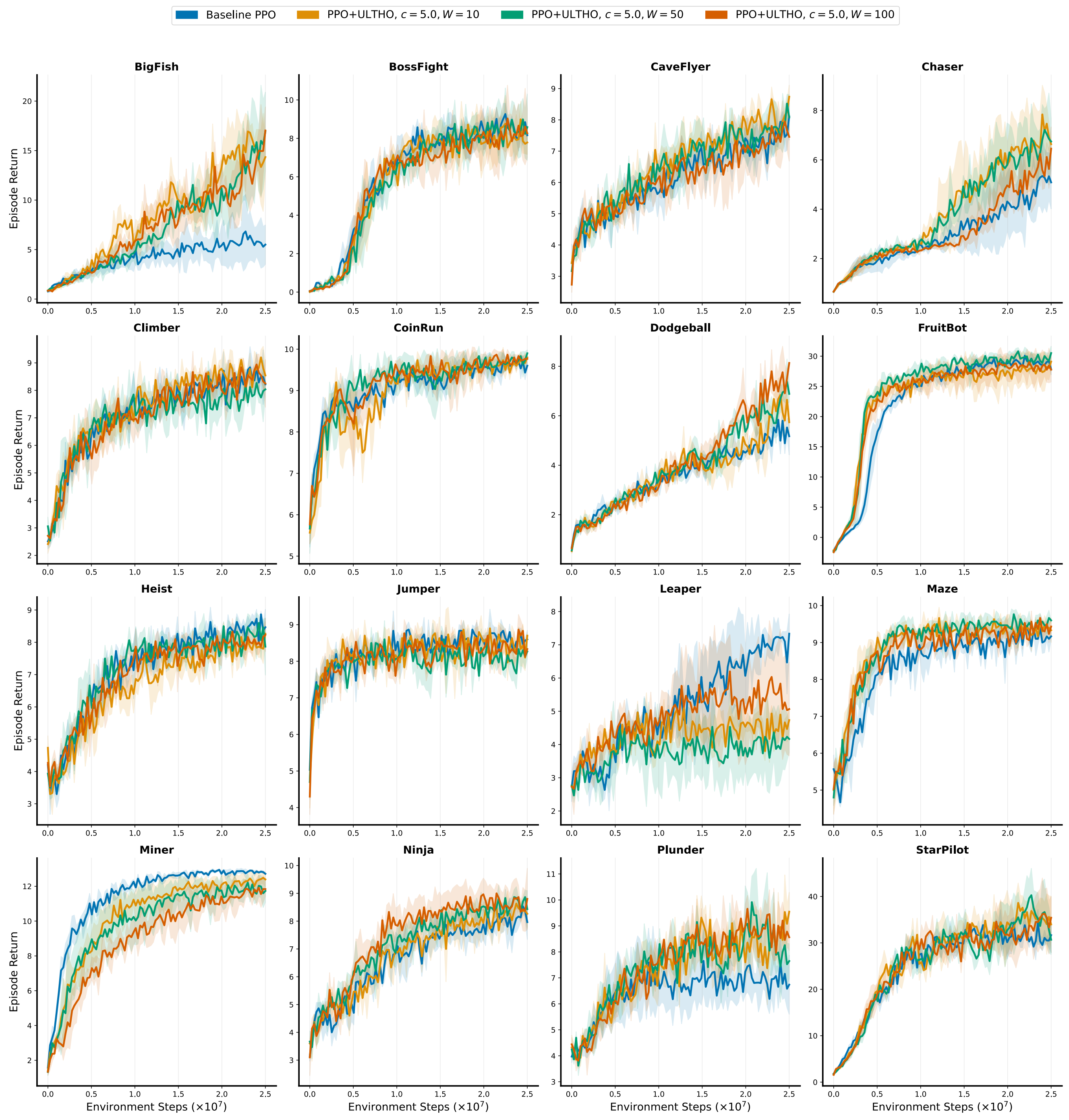

Figure 16. Learning curves of the vanilla PPO agent and ULTHO with different sizes of the sliding window on the Procgen benchmark. Here, the exploration coefficient c is set as 5.0. The mean and standard deviation are computed over five runs with different seeds.

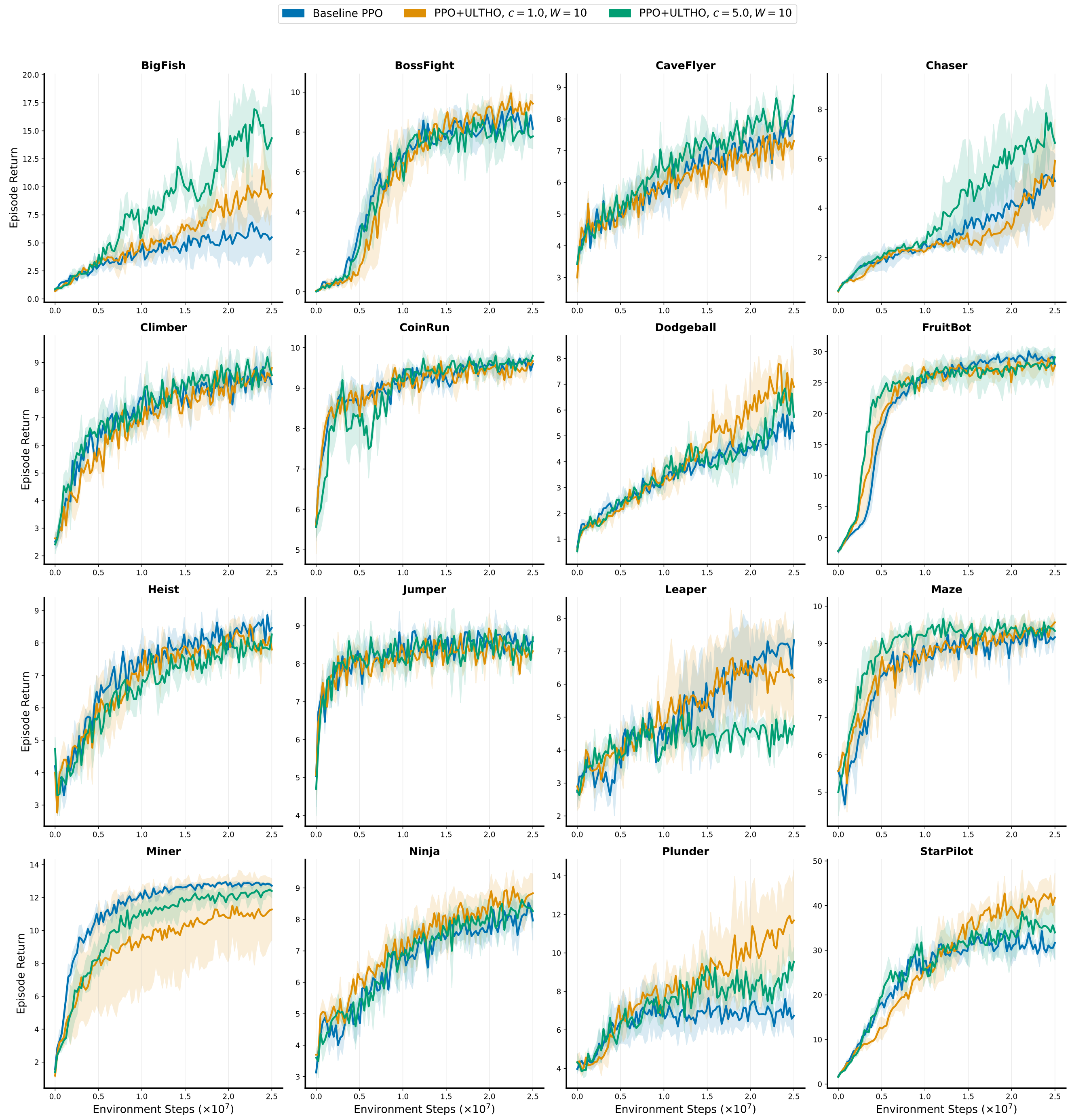

Figure 17. Learning curves of the vanilla PPO agent and ULTHO with different exploration coefficients on the Procgen benchmark. Here, the size W of the sliding window is set as 10. The mean and standard deviation are computed over five runs with different seeds.

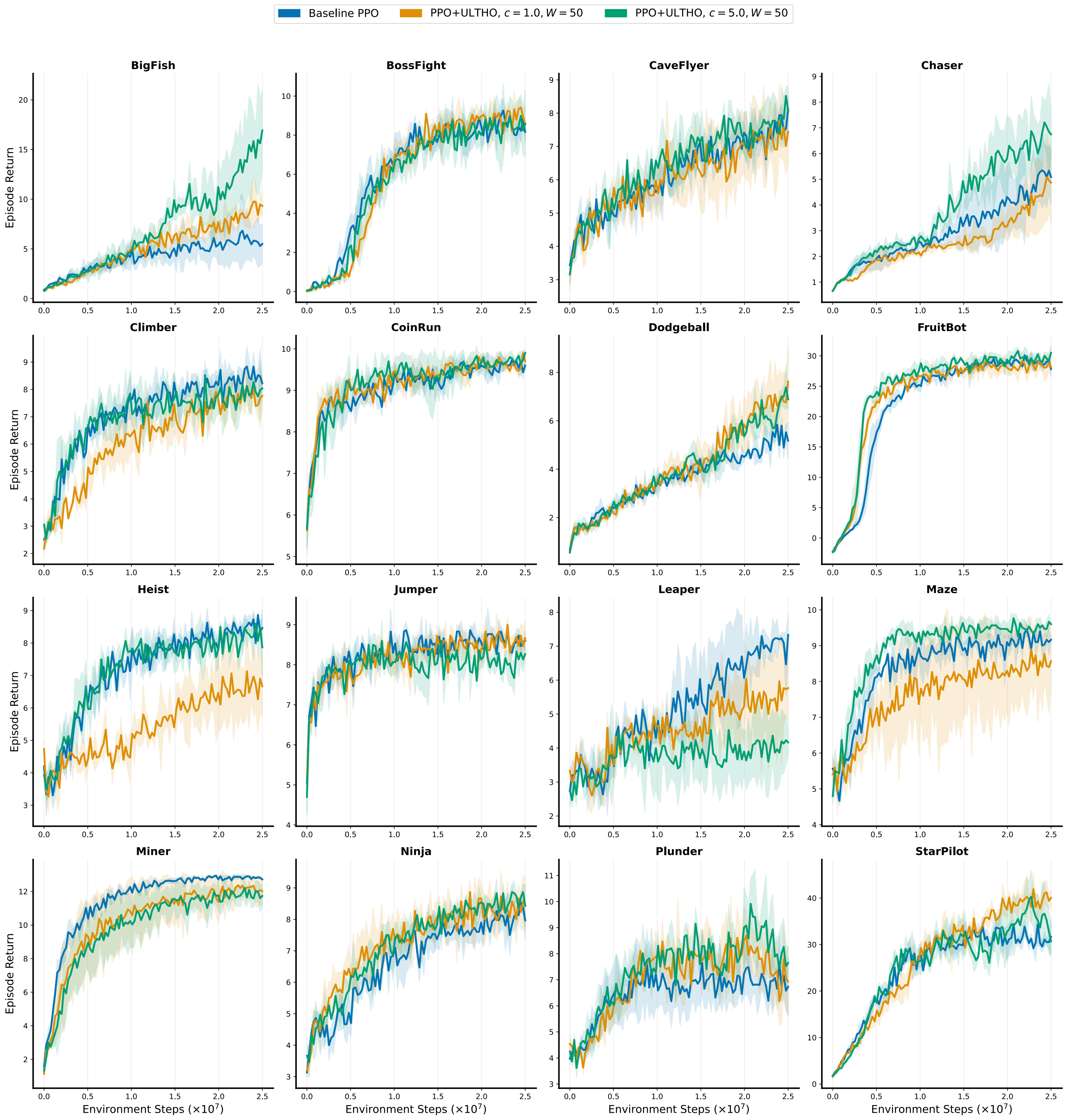

Figure 18. Learning curves of the vanilla PPO agent and ULTHO with different exploration coefficients on the Procgen benchmark. Here, the size W of the sliding window is set as 50. The mean and standard deviation are computed over five runs with different seeds.

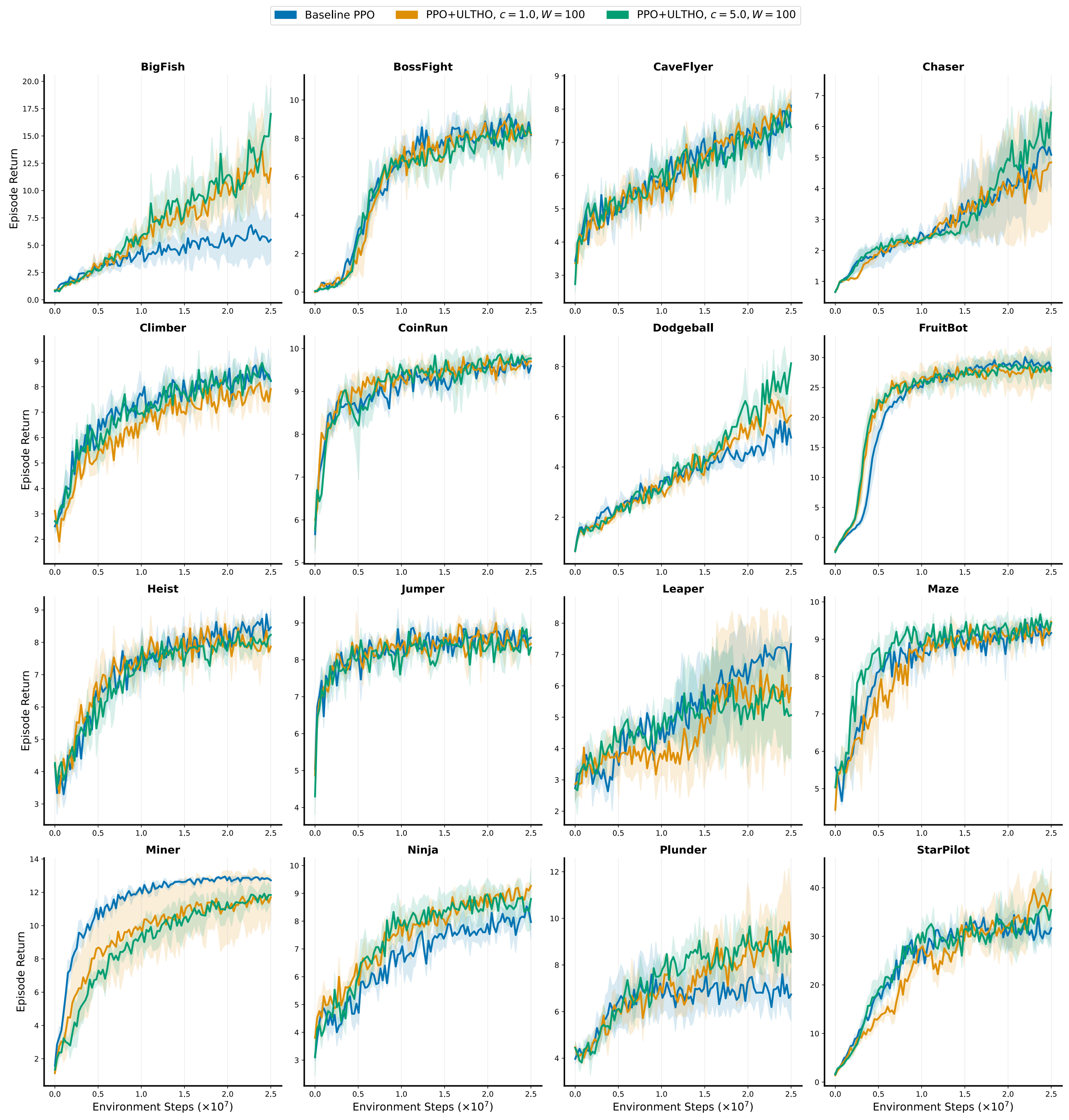

Figure 19. Learning curves of the vanilla PPO agent and ULTHO with different exploration coefficients on the Procgen benchmark. Here, the size W of the sliding window is set as 100. The mean and standard deviation are computed over five runs with different seeds.

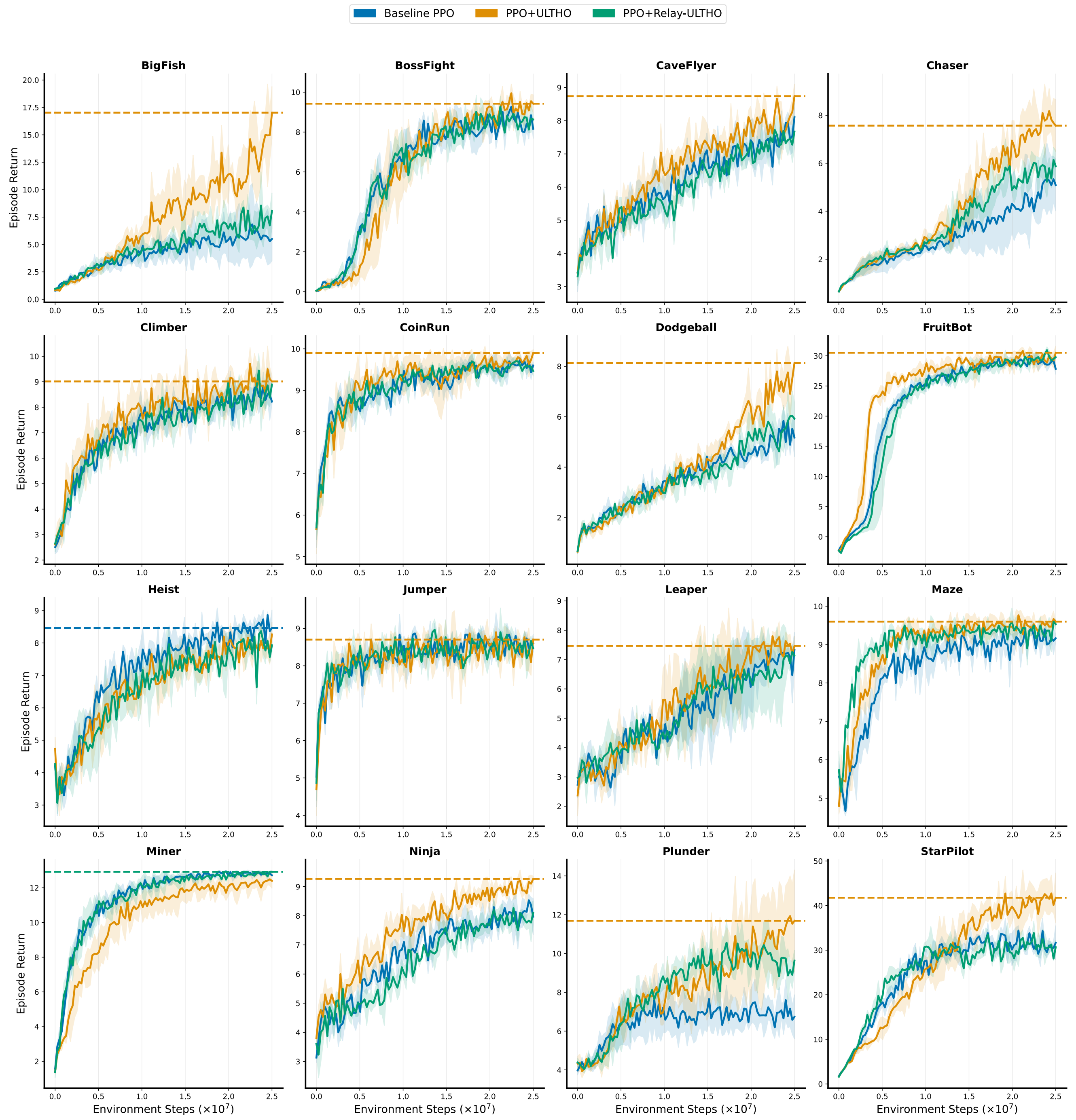

Figure 20. Learning curves of the vanilla PPO agent and two ULTHO algorithms on the Procgen benchmark. The mean and standard deviation are computed over five runs with different seeds.

D. Ablation Studies

ULTHO versus baselines (same training budgets). The Figure below compares ULTHO and the baselines on the ALE-5 benchmark using the same training budget, 100M environment steps. By achieving highly frequent HP tuning within a limited budget, ULTHO consistently outperforms the baselines.

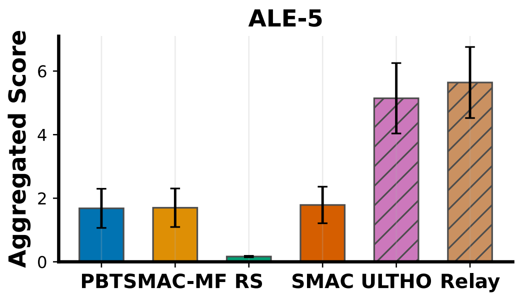

Figure 21. The aggregated performance comparison of ULTHO and baselines on the ALE-5 benchmark with the same training budgets.

ULTHO's performance with varying sets of HPs. The Figure below compares the performance of ULTHO on the Procgen benchmark with the varying number of HP clusters and intra-cluster HPs. There is a notable performance gain when using more HP clusters. However, as a lightweight framework and constrained by the capability of MAB algorithms, continually increasing the types and granularity of HPs only achieves limited performance gains.

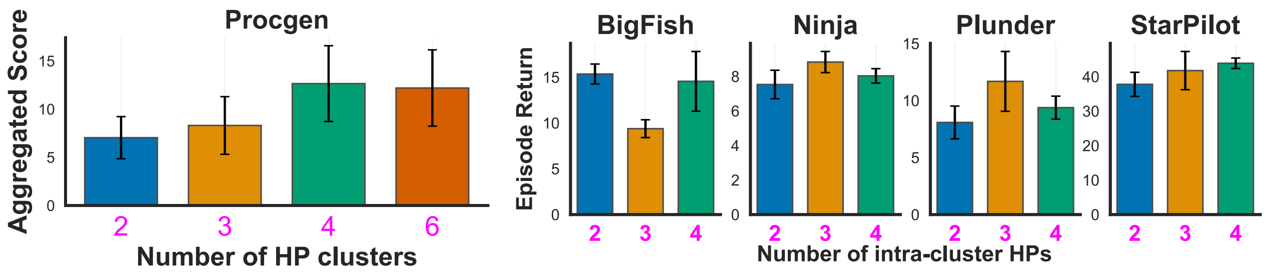

Figure 22. ULTHO's performance comparison with the varying number of HP clusters and intra-cluster HPs.

ULTHO's performance with off-policy RL algorithms. It is simple to apply ULTHO to off-policy RL algorithms. Take SAC [26] for example, every T steps, we use ULTHO to select a HP before using it for model update in the next T steps. The utility of each HP can be evaluated using the action value function: $U_i(\psi) = \frac{1}{W}\sum_{j=1}^{W}\frac{1}{|\mathcal{B}_j|}\sum_{(\boldsymbol{s},\boldsymbol{a})\sim\mathcal{B}_j} Q(\boldsymbol{s},\boldsymbol{a})$, where $\mathcal{B}$ is a sampled batch. We perform an experiment using two tasks from the DMC benchmark [57], and the Figure above indicates that ULTHO effectively enhances SAC's performance. However, at the current stage, we do not recommend applying ULTHO to off-policy algorithms as the return estimation based on sampling from a replay buffer is relatively unstable.

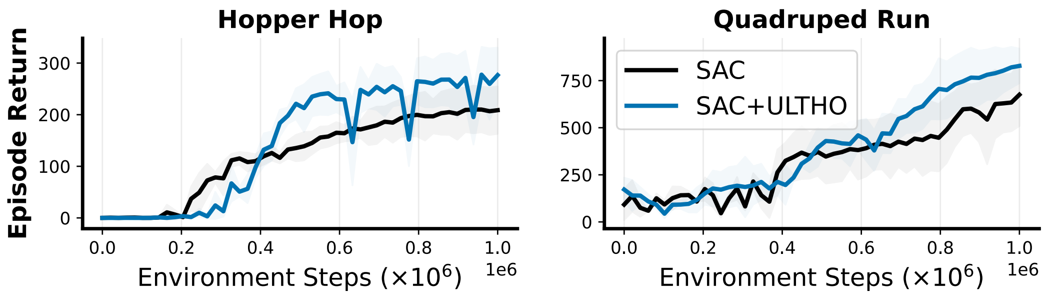

Figure 23. The performance of SAC+ULTHO on the DMC benchmark.

E. Detailed Decision Processes

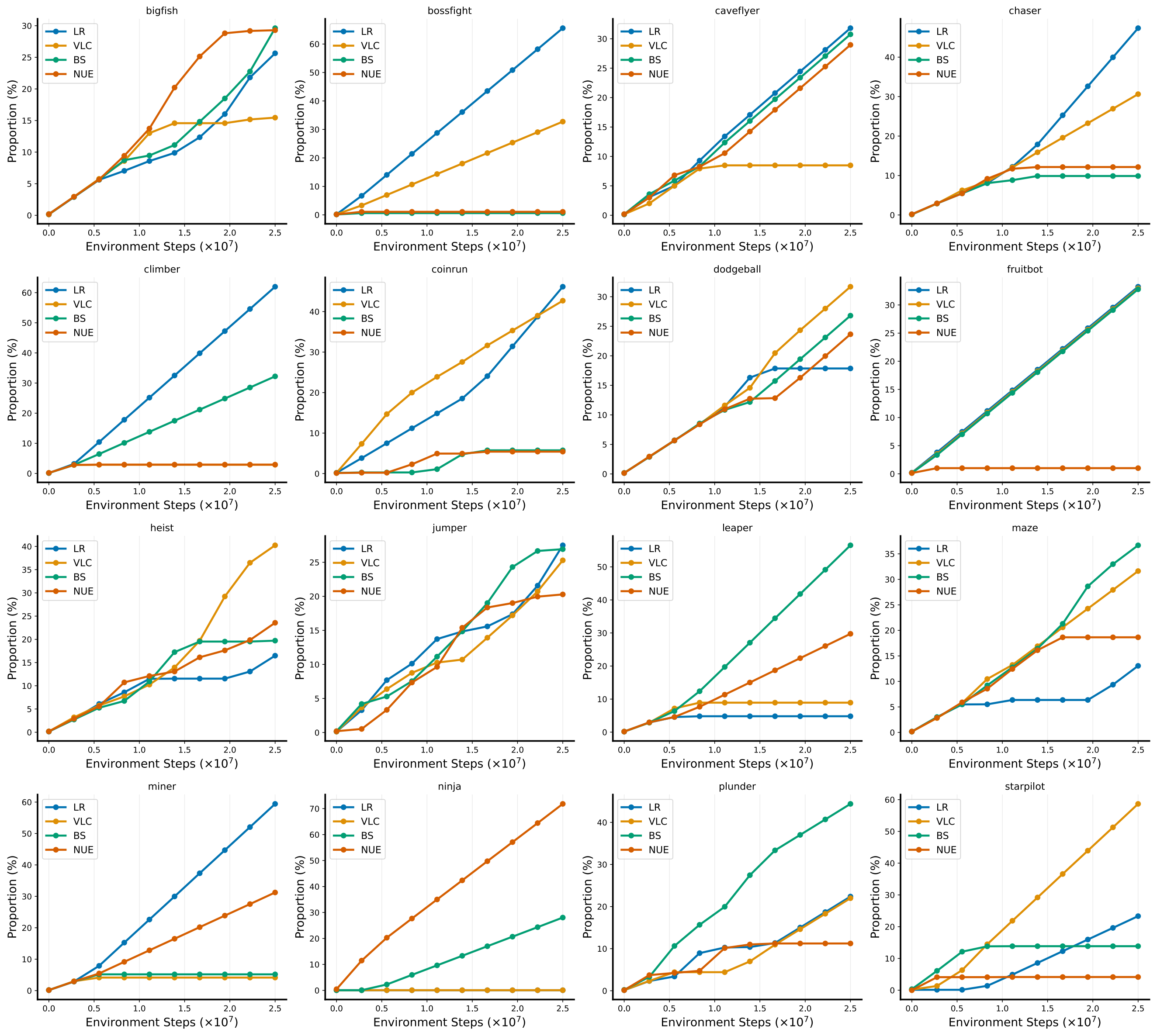

Figure 24. Detailed decision processes of PPO+ULTHO on the Procgen benchmark.

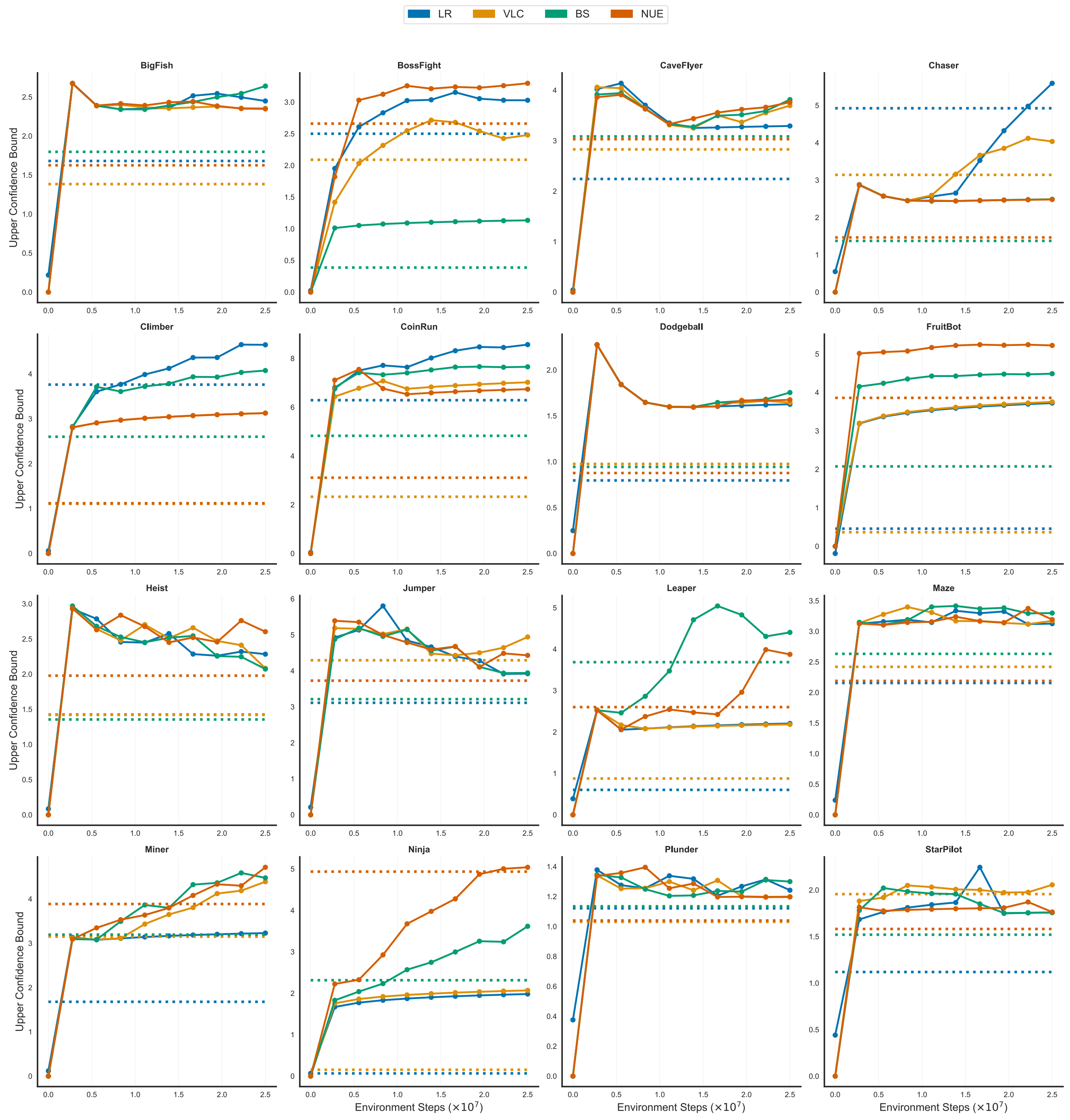

Figure 25. The variation of confidence intervals of PPO+ULTHO on the Procgen benchmark. Here, the solid line represents the mean value, and the dashed line represents the final upper confidence bound.