Journal of Mathematical Modeling
Vol. 14, No. 1, 2026, pp. 177-190. Research Article

JMM

Proximal policy optimization with adaptive generalized advantage estimate: critic-aware refinements

Naemeh Mohammadpour[†][*], Meysam Fozi[‡], Mohammad Mehdi Ebadzadeh[‡], Ali Azimi[†], Ali Kamali[†]

[†] *Department of Mechanical Engineering, Amirkabir University of Technology, Tehran, Iran*
[‡] *Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran*

Email(s): {*naemeh.mpr, meysam.fozi, ebadzadeh, aliazimi, alikamalie*}*@aut.ac.ir*

Abstract. Proximal Policy Optimization (PPO) is one of the most widely used methods in reinforcement learning, designed to optimize policy updates while maintaining training stability. However, in complex and high-dimensional environments, maintaining a suitable balance between bias and variance poses a significant challenge. The λ parameter in Generalized Advantage Estimation influences this balance by controlling the trade-off between short-term and long-term return estimations. In this study, we propose a method for adaptive adjustment of the λ parameter, where λ is dynamically updated during training instead of remaining fixed. The updates are guided by internal learning signals such as the value function loss and Explained Variance—a statistical measure that reflects how accurately the critic estimates target returns. To further enhance training robustness, we incorporate a Policy Update Delay mechanism to mitigate instability from overly frequent policy updates. The main objective of this approach is to reduce dependence on expensive and time-consuming hyperparameter tuning. By leveraging internal indicators from the learning process, the proposed method contributes to the development of more adaptive, stable, and generalizable reinforcement learning algorithms. To assess the effectiveness of the approach, experiments are conducted in four diverse and standard benchmark environments: Ant-v4, HalfCheetah-v4, and Humanoid-v4 from the OpenAI Gym, as well as Quadruped-Walk from the DeepMind Control Suite. The results demonstrate that the proposed method can substantially improve the performance and stability of PPO across these environments. Our implementation is publicly available at https://github.com/naempr/PPO-with-adaptive-GAE.

Keywords: Reinforcement learning, proximal policy optimization, generalized advantage estimate, bias-variance trade-off
AMS Subject Classification 2010: 68T40, 68T05, 68T07.

[*]Corresponding author

Received: 01 February 2025/ Revised: 23 September 2025/ Accepted: 30 September 2025
DOI: 10.22124/jmm.2025.29704.2654

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful tool for solving complex decision-making problems through interaction with environment. Policy gradient methods [10], which adjust policy parameters to maximize expected rewards, are particularly effective in continuous action spaces [2, 4–7]. To address the limitations of traditional policy gradient methods, Proximal Policy Optimization (PPO) was introduced as an efficient and robust alternative. The PPO improves upon earlier methods by using a clipping mechanism in policy updates, which prevents drastic changes and enhances learning stability. This approach, combined with its computational simplicity, has made PPO one of the most widely used RL algorithms today. Additionally, techniques such as Generalized Advantage Estimate (GAE) have been developed to address the bias-variance trade-off in RL, further improving the performance of policy gradient algorithms.

In this paper, we first explain PPO and GAE, then we propose a dynamic adjustment of λ parameter in GAE to achieve a better balance between bias and variance, which is crucial for the stability of the learning process. Also a policy update delay is added to PPO to enhance the control of updates, helping to increase the algorithm's stability. Finally, we evaluate our approach by conducting experiments in two popular environments, OpenAI Gym and DeepMind, to demonstrate the benefits of dynamic parameter tuning on RL performance.

2 Related work

Policy gradient methods optimize policies by directly adjusting parameters to maximize expected rewards. Instead of learning a value function, they update the policy in the direction that increases the probability of rewarding actions. This approach is effective for stochastic policies and continuous action spaces, using function approximators like neural networks.

The PPO [9], Algorithm 1, is a type of policy gradient method designed to optimize policies in RL while maintaining stability and preventing large, destructive policy updates. The central idea of PPO is to maximize a "surrogate objective function", allowing multiple epochs of optimization on the same batch of data. Unlike standard policy gradient methods, PPO introduces a mechanism to limit the extent to which the policy is updated, using either a clipping method or a Kullback-Leibler (KL) penalty.

The surrogate objective in PPO is represented as:

$$L_{\text{CPI}}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right], \tag{1}$$

where the superscript CPI refers to conservative policy iteration [3], $\pi_\theta(a_t|s_t)$ is the new policy, $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the old policy, $\hat{A}_t$ is the advantage estimate, $\hat{\mathbb{E}}_t$ is the empirical expectation over time-steps. This objective aims to encourage the policy to increase the probability of actions that are deemed advantageous by $\hat{A}_t$. The PPO avoids large, destabilizing policy updates by clipping the probability ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. The clipped objective is

$$L_{\text{clip}}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t) \right], \tag{2}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio. ε is the clipping hyper-parameter, typically set by a number in the interval $[0.1, 0.3]$. The clip function ensures that if the probability ratio, $r_t(\theta)$, deviates too far from 1 (either below $1-\varepsilon$ or above $1+\varepsilon$), it gets clipped. This mechanism prevents the policy from changing too drastically in one update, thereby stabilizing training. $r_t(\theta)\hat{A}_t$ is the normal policy gradient. $\text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t$ limits the update size. The use of the min function ensures that the objective is only reduced when the change would lead to an excessively large update, keeping the training stable. This combination of clipping and multiple mini-batch updates leads to efficient and robust policy learning, striking a balance between performance and stability.

Algorithm 1: The PPO

```
1: Initialize $\theta$ for policy network
2: for each iteration do
3:     for actor = $1, 2, \dots, N$ do
4:         Run policy $\pi_{\theta_{\text{old}}}$ for $T$ timesteps.
5:         Compute advantage estimates $\hat{A}_1, \dots, \hat{A}_T$.
6:     end for
7:     Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \le NT$.
8:     $\theta_{\text{old}} \leftarrow \theta$.
9: end for
```

The $\hat{A}_t$ is calculated using GAE [8], which helps balance the bias-variance trade-off in RL. The GAE adds flexibility to the way future rewards are accounted for in policy updates. The advantage is calculated using

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+1}^V, \tag{3}$$

where γ is the discount factor, which controls the weight given to future rewards, λ is a hyper-parameter, which adjusts the balance between bias and variance, δ_t represents the Temporal Difference (TD) error, which measures the difference between the value of a state and predicted value of the next state. The TD error is calculated as

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{4}$$

where r_t is the reward at time step t, and $V(s_t), V(s_{t+1})$ are the value estimates for the current and next state, respectively.

The structure of GAE is similar to $TD(\lambda)$, a method used for value function estimation. However, $TD(\lambda)$ focuses on estimating the value function, while GAE is designed to estimate the advantage function, which helps in policy optimization.

Parameter λ plays a crucial role in managing the bias-variance trade-off in PPO:

- **High λ values (close to 1)**: This setting puts more weight on future rewards, reducing bias but increasing variance. In this case, the advantage estimate sums many future TD errors:

$$\hat{A}_t \approx \sum_{l=0}^{\infty} \gamma^l \delta_{t+1}. \tag{5}$$

- **Low λ values (close to 0)**: This setting focuses more on immediate rewards, reducing variance but increasing bias. In this case, the advantage estimate mainly reflects the current TD error:

$$\hat{A}_t \approx \delta_t. \tag{6}$$

The parameters γ and λ both help in managing the bias-variance trade-off when working with an approximate value function. However, they serve distinct purposes and are optimal in different value ranges. The discount factor γ mainly influences the scaling of the value function V^{π}, and operates independently of λ. Setting $\gamma < 1$ introduces bias in the policy gradient estimate, regardless of how accurate the value function is. In contrast, $\lambda < 1$ result in bias only if the value function is not accurate.

3 Proposed methods

3.1 The PPO with adaptive GAE

To enhance the performance of PPO, we dynamically adjust λ based on changes in value loss during training. Adjusting λ helps the model adapt to variations in the learning process and a better bias-variance trade-off. When λ is high (close to 1), the advantage estimate, $\hat{A}_t$, heavily depends on future advantages. In this scenario, the network relies more on predictions of future rewards, which can introduce noise and uncertainty, especially if the critic network is not properly trained to provide accurate value predictions.

Algorithm 2: The PPO with adaptive GAE

```
 1: Initialize $v_{\text{loss}}^{\text{old}} \leftarrow \infty$.
 2: Initialize total time steps $T$ and global step $\leftarrow 0$.
 3: Initialize $\lambda \leftarrow 0.95$ for GAE
 4: for each iteration do
 5:     ▷ Dynamically adjust $\lambda$.
 6:     for each episode do
 7:         Compute value loss $v_{\text{loss}}$.
 8:         $m \leftarrow 0.09 - (0.085 \cdot \text{global step}/T)$.
 9:         if $v_{\text{loss}} < v_{\text{loss}}^{\text{old}}$ then
10:             $\lambda \leftarrow \max(0.7, \lambda - m)$.
11:         else if $v_{\text{loss}} > v_{\text{loss}}^{\text{old}}$ then
12:             $\lambda \leftarrow \min(0.99, \lambda + m)$.
13:         end if
14:         $v_{\text{loss}}^{\text{old}} \leftarrow v_{\text{loss}}$.
15:     end for
16: end for
```

In Algorithm 2, parameter λ is adaptively adjusted based on value loss:

- When value loss *decreases*, indicating the critic network is becoming more accurate, λ is reduced, which results in a lower variance. This reduces reliance on future advantages and focuses more on current information, i.e., immediate TD errors.

- When value loss *increases*, indicating poor performance of the critic network, λ is increased, leading to higher variance that can help to improve the algorithm.

The parameter m controls the adjustment rate, initially set to 0.09 and decreased over time. Additionally, this adjustment of λ is more intense at the beginning and gradually decreases over time, allowing the algorithm to have sufficient opportunity to test different λ values with larger steps initially. This approach ensures that the algorithm can explore a wider range of values early on before refining its choices. This approach dynamically tunes λ, aiming to optimize GAE depending on how well the value function is being learned, making it more adaptive to environment's needs. Parameter $\lambda = 0.95$ has been adapted from the original PPO paper, and parameter m is set to linear decay with respect to total timestep. This leads to adaptation to training progress, ensuring more efficient exploration in early stages and refined updates as training stabilizes.

3.2 The PPO with adaptive GAE and policy update delay

The concept of Policy Update Delay (PUD) after several updates to the critic was introduced in Twin Delayed Deep Deterministic (TD3) [1] to reduce approximation errors and improve learning stability. This concept can also be applied to enhance the control of updates in PPO, presented in Algorithm 3, helping to mitigate large fluctuations in the policy and increase the algorithm's stability. In our approach, considering the dependency of λ update on the value loss, it is recommended to perform policy updates after several updates to the critic to ensure higher accuracy in learning. Frequent updates to the critic networks compared to the actor provide the critic with more opportunities to minimize value errors effectively.

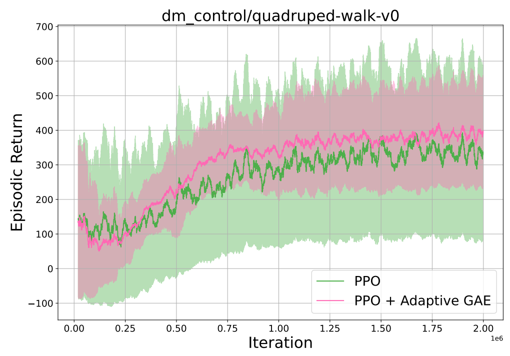
Algorithm 3: The PPO with adaptive GAE and PUD

```
1:  Initialize policy_frequency ← 2.
2:  for each iteration do
3:      ▷ Policy Update Delay
4:      for each epoch do
5:          if epoch % policy_frequency == 0 then
6:              Update actor policy $L_{\text{actor}}$.
7:          end if
8:      end for
9:      Update critic policy $L_{\text{critic}}$.
10: end for
```

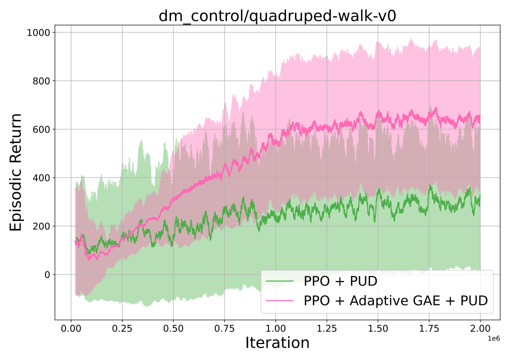
4 Results

4.1 Empirical evaluation

We evaluated the proposed methods in four different benchmarks, Quadruped-Walk-v0 from Google DeepMind [12], and Ant-v4, Humanoid-v4, HalfCheetah-v4 from OpenAI Gym [11], each with 5 seeds and compared them with PPO as the baseline.



(a) PPO with Adaptive GAE



(b) PPO with Adaptive GAE and PUD

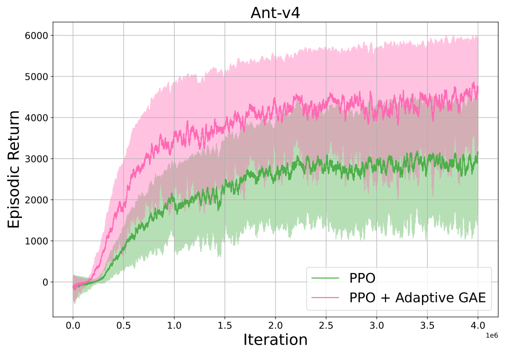
Figure 1: Simulation results for DeepMind Quadruped-Walk-v0 benchmark

Results on Google DeepMind Control Environment, as shown in Figure 1a, over two million iterations of the Quadruped-Walk-v0 benchmark, mean episodic return increased from 350 to 400, while the variance was significantly reduced. We first introduced policy update delay to PPO (`PPO + PUD`). Then, we took adaptive GAE into account (`PPO + PUD + Adaptive GAE`). Comparing Figure 1a and Figure 1b, the policy update delay alone resulted in weaker performance compared to PPO (`PPO` vs. `PPO + PUD`). However, when combined with adaptive GAE, it leads to clearly improved results over PPO (`PPO` vs. `PPO + Adaptive GAE + PUD`). Additionally, both the variance and mean episodic return increased; so the combination of both methods does not show significant progress compared to PPO with adaptive GAE (`PPO + Adaptive GAE` vs. `PPO + Adaptive GAE + PUD`). Therefore, policy update delay can, in some cases, lead to improved performance.

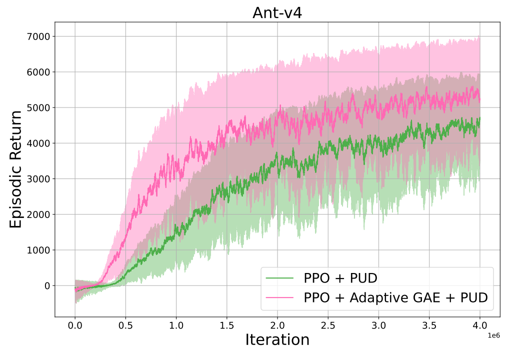
Results on OpenAI Gym MuJoCo Environment, as shown in Figure 2a, in four million iterations of the Ant-v4 benchmark, mean episodic return increased from 3000 to 4500 by keeping variance in a constant range. We first introduced the policy update delay to PPO. Then, we incorporated PPO with adaptive GAE. Comparing Figure 2a and Figure 2b, the policy update delay alone was able to improve PPO (`PPO` vs. `PPO+PUD`). However, when combined with adaptive GAE, as shown in Figure 2b, mean episodic return increased from 4500 to 5400 while maintaining the same variance (`PPO + PUD` vs. `PPO + Adaptive GAE + PUD`). The combination of these two methods resulted in beneficial outcomes.

Results on OpenAI Gym MuJoCo Environment for Humanoid-v4 benchmark, as shown in Figure 3a, in two million iterations, mean episodic return increased from 700 to 1100 by increasing variance as well but the increase in return mean is much greater than the increase in variance and the model outperforms well enough. Comparing Figure 3a and Figure 3b, the policy update delay alone was able to improve PPO (`PPO` vs. `PPO+PUD`). However, when combined with adaptive GAE, as shown in Figure 3b, mean episodic return increased from 1500 to 4000, and increasing variance as well, but the the increase in return mean is much greater than the increase in variance and the model outperforms (`PPO + PUD` vs. `PPO + Adaptive GAE + PUD`). The combination of these two methods resulted in beneficial outcomes in this benchmark as well.

Results on OpenAI Gym MuJoCo Environment for HalfCheetah-v4 benchmark, as shown in Figure 4a, in two million iterations, mean episodic return decreased from 2900 down to 1900 but also decreasing variance significantly. Comparing Figure 4a and Figure 4b, the policy update delay was not able to

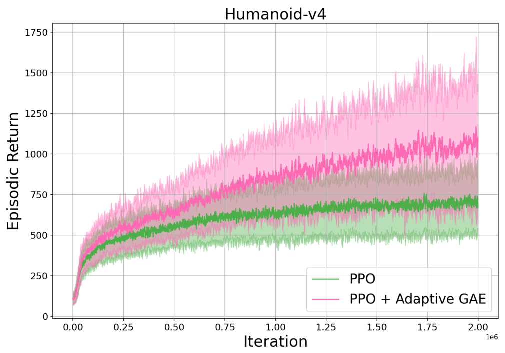


(a) PPO with Adaptive GAE

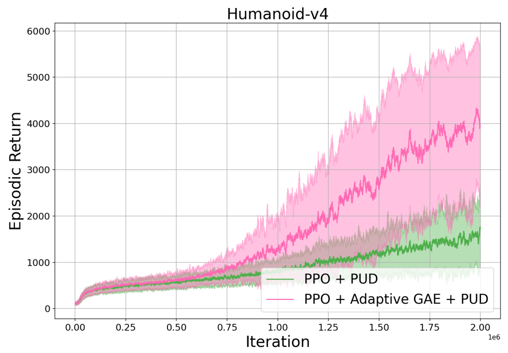


(b) PPO with Adaptive GAE and PUD

Figure 2: Simulation results for OpenAI Gym Ant-v4 benchmark



(a) PPO with Adaptive GAE.



(b) PPO with Adaptive GAE and PUD.

Figure 3: Simulation results for OpenAI Gym Humanoid-v4 benchmark

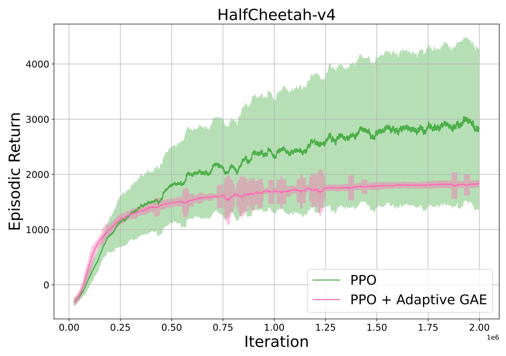
improve PPO.

4.2 Explained variance analysis

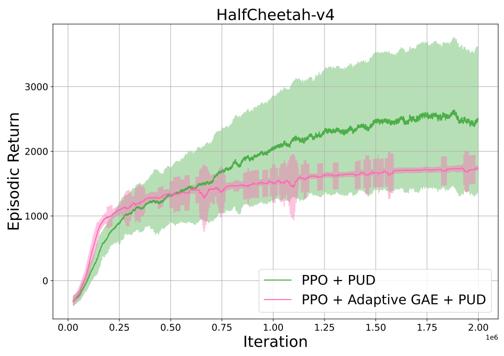
Explained Variance (EV) is an important metric in RL to assess how accurately the critic (value network) approximates the target returns. Formally, when $\mathrm{Var}(y_{\text{true}}) \neq 0$, it is defined as

$$\mathrm{EV} = 1 - \frac{\mathrm{Var}(y_{\text{true}} - y_{\text{pred}})}{\mathrm{Var}(y_{\text{true}})}, \tag{7}$$

where y_{true} is the target return (e.g., computed via GAE), y_{pred} is the predicted value from the critic network, and $\mathrm{Var}(\cdot)$ denotes the statistical variance. An EV close to 1 indicates that the critic network predicts target returns with high accuracy. An EV close to 0 implies that the predictions are no better than a constant average baseline. Negative EV values suggest performance worse than using a constant prediction.



(a) PPO with adaptive GAE



(b) PPO with adaptive GAE and PUD

Figure 4: Simulation results for OpenAI Gym HalfCheetah-v4 benchmark

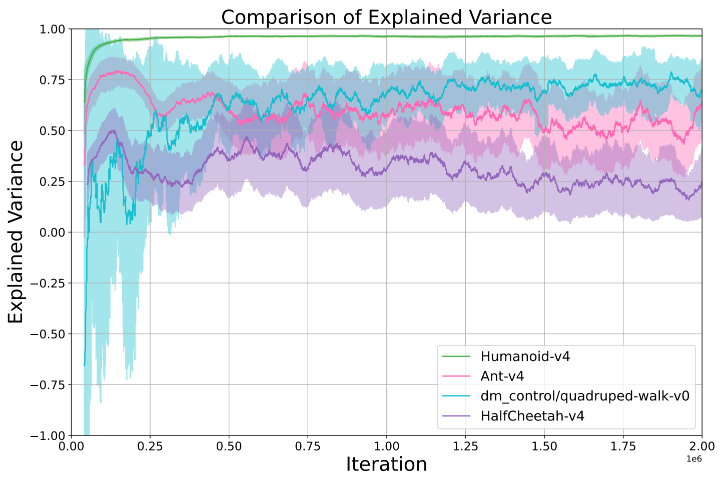
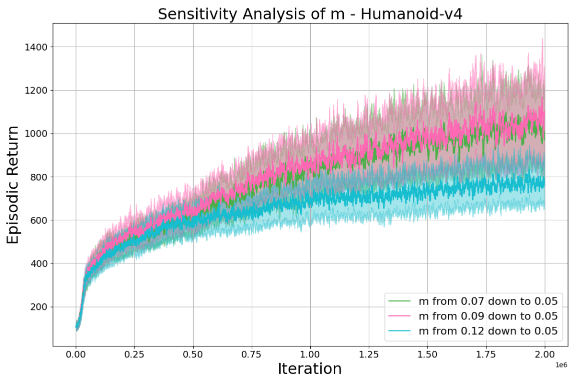


Figure 5: Explained variance analysis

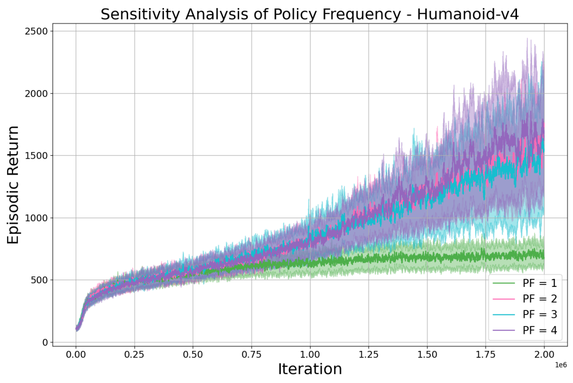
As illustrated in Figure 5, HalfCheetah-v4 exhibits the lowest EV among the four tested environments, fluctuating between 0.2 and 0.4 with significant instability. This highlights a critical limitation: the critic network fails to capture the structure of the returns effectively in this environment. In contrast, the Humanoid-v4 benchmark shows substantially higher EV values, indicating better critic performance. However, the cumulative reward remains suboptimal, which may result from convergence to a local optimum rather than global success.

The adaptive λ method relies on the critic's accuracy to dynamically balance bias and variance. In environments such as HalfCheetah-v4, where the critic underperforms, this dependency can hinder the method's effectiveness. Our findings indicate that relying on an unstable critic may degrade policy performance.

To mitigate this, we can use a conditional version of the adaptive λ method based on EV metric.



(a) Paramater m



(b) Parameter $\mathtt{PF}$

Figure 6: Sensitivity analysis for OpenAI Gym Humanoid-v4 benchmark

Specifically, the algorithm could activate λ adjustment only when

$$\mathrm{EV} > \delta, \quad \text{with } \delta \in [0.5, 1.0]. \tag{8}$$

This threshold ensures that adaptive adjustment is employed only when the critic provides sufficiently reliable estimates. By introducing this conditional logic, the method becomes more robust and avoids deteriorating performance in environments with weak value approximation.

4.3 Sensitivity analysis

To more accurately assess the robustness of the proposed method, a comprehensive sensitivity analysis is performed on two key hyperparameters: the policy update frequency and the parameter adjustment rate m, within Humanoid-v4 environment. Due to its high dimensionality and complex dynamics, this environment serves as a challenging and appropriate benchmark for evaluating the stability and performance of RL algorithms. It is worth noting that, to improve the clarity of the plots and enable more effective comparisons, the variance in episodic returns was normalized by a factor of 0.5 to reduce visual noise.

4.3.1 Parameter adjustment rate m

Next, three different linear decay schedules for the parameter m—which controls the adaptive adjustment rate λ—are evaluated as follows:

- $m \in [0.07, 0.005]$,
- $m \in [0.09, 0.005]$, (default configuration)
- $m \in [0.12, 0.005]$.

In all three cases, the value of m was linearly decreased over the course of training. This allowed larger adjustments to λ in the early stages and progressively finer tuning in later stages.

As shown in Figure 6a, all tested values of m yielded better performance than the standard PPO. These findings indicate that the adaptive adjustment mechanism for λ is *not* highly sensitive to reasonable changes in the m parameter and maintains stable performance across a range of values. Moreover, none of the tested configurations led to a drop in performance, further validating the robustness and reliability of the proposed method.

4.3.2 Policy update frequency

In the default configuration of the proposed algorithm, Policy Frequency (PF) is set to 2, meaning the actor network is updated once every two updates of the critic network. To evaluate sensitivity, the following values are tested:

$$\texttt{policy\_frequency} \in \{1,2,3,4\}. \tag{9}$$

A value of `PF = 1` corresponds to the standard PPO algorithm, where the actor and critic are updated simultaneously in each iteration. As shown in Figure 6b, all values greater than 1 (i.e., 2, 3, and 4) resulted in higher episodic returns compared to the baseline configuration. The results also showed that the performance across values 2 to 4 was relatively stable and similar, suggesting that even a slight delay in policy updates (e.g., a frequency of 2) can improve learning stability. While the standard PPO implementation achieved an episodic return of around 750, all modified versions surpassed this value.

4.4 Complexity analysis

We provide a theoretical analysis of the time and space complexity of the proposed method, compared against standard PPO. Let N denote the number of parallel environments, T the number of steps per environment in each rollout, and $B = N \cdot T$ the total batch size. Each update is optimized for K epochs, with M minibatches per epoch. Denote the state and action dimensions by d_s and d_a, respectively, and the average episode length by $L \leq T$. Let $|\theta_\pi|$ and $|\theta_V|$ be the number of parameters for actor network and critic network, respectively. We abstract per-sample forward and backward computational costs by constants $c_\pi^f, c_\pi^b, c_V^f, c_V^b$. Also let $\mathscr{T}$ and $\mathscr{S}$ denote time and space complexities, respectively. The environment simulation cost is denoted by c_{env} per step, but as it is independent of the algorithm, we report it separately from the algorithmic complexity.

4.4.1 Baseline PPO

Using our notation, the computational complexity of PPO will be decomposed to the following:

1. *Data collection:* Each transition requires one actor forward and one critic forward pass:
$$\mathscr{T}_{\text{collect}} = \mathscr{O}\big(B(c_\pi^f + c_V^f)\big). \tag{10}$$
2. *Advantage computation:* The backward recursion for GAE is linear in the batch size:
$$\mathscr{T}_{\text{GAE}} = \mathscr{O}(B). \tag{11}$$
3. *Optimization:* For each of K epochs, all B samples are used across minibatches. Each sample requires one forward and one backward pass through both actor and critic networks:
$$\mathscr{T}_{\text{opt}} = \mathscr{O}\big(KB(c_\pi^f + c_\pi^b + c_V^f + c_V^b)\big). \tag{12}$$
4. *Space:* Storing trajectories $(s, a, r, \hat{A}, V, \log \pi_{\theta_{\text{old}}}(a|s))$ gives:
$$\mathscr{S}_{\text{PPO}} = \Theta\big(B(d_s + d_a + 1)\big). \tag{13}$$

4.4.2 Additional cost of adaptive λ

Adjusting the GAE parameter λ per episode requires only a few scalar operations (e.g., updating a running statistic), with no extra gradient computations or network passes. Thus

$$\mathcal{T}_{\text{adaptive-}\lambda} = \mathcal{O}\big(B/L\big) \subseteq \mathcal{O}(B) \tag{14}$$

with negligible constant factors, and space overhead $\mathcal{O}(1)$.

4.4.3 Effect of PUD

If the policy is updated once every PF critic updates, then for each training iteration consisting of K epochs over B samples:

$$\mathcal{T}_{\text{opt}}^{\text{PUD}} = \mathcal{O}\Big(KB(c_V^f + c_V^b) + \tfrac{K}{\text{PF}}B(c_\pi^f + c_\pi^b)\Big). \tag{15}$$

Relative to PPO, this reduces the actor optimization cost by approximately a factor of $1/\text{PF}$, while leaving critic computation unchanged.

4.4.4 Overall complexity

The proposed method (adaptive λ and PUD) therefore has complexity

$$\mathcal{T}_{\text{ours}} = \mathcal{O}\big(B(c_\pi^f + c_V^f)\big) + \mathcal{O}(B) + \mathcal{O}\Big(KB(c_V^f + c_V^b) + \tfrac{K}{\text{PF}}B(c_\pi^f + c_\pi^b)\Big) \tag{16}$$

with space

$$\mathcal{S}_{\text{ours}} = \Theta\big(B(d_s + d_a + 1)\big). \tag{17}$$

Hence, the algorithm preserves the same asymptotic time and space complexity as PPO, while adaptive λ introduces only negligible overhead, and PUD can reduce actor optimization by a constant factor depending on PF.

4.5 Wall-clock runtime and memory (empirical analysis)

To complement the theoretical analysis in the previous section, we measured empirical runtime and memory usage on the Humanoid-v4 environment under three different settings: standard PPO, PPO with PUD, and PPO with adaptive λ adjustment (Table 1). The results indicate that applying PUD reduced execution time by approximately 2.7% relative to PPO, due to fewer actor updates. In contrast, adaptive λ adjustment introduced only a negligible 0.9% increase in execution time, as it involves lightweight scalar operations. Peak memory consumption remained unchanged across all methods.

Overall, these empirical measurements confirm that the proposed refinements introduce only marginal runtime differences, while maintaining asymptotic complexity equivalent to PPO.

4.6 Hyperparameters

In this study, all hyperparameters used in the proposed PPO variant are defined in accordance with the standard settings of the original PPO algorithm. A comprehensive summary of these configurations is provided in Table 2. In addition to the baseline parameters, two key modifications are introduced:

Table 1: Wall-clock runtime relative change compared to PPO on Humanoid-v4

Method	%Δ Runtime vs PPO
PPO	–
PPO + PUD	-2.7%
PPO + Adaptive λ	+0.9%
PPO + Adaptive λ + PUD	-1.8%

Table 2: Key hyperparameters used in PPO and proposed variants

Hyperparameter	**Value / Range**	**Description**
`learning_rate`	3×10^{-4}	Learning rate for the Adam optimizer
`num_envs`	1	Number of parallel simulation environments
`num_steps`	2048	Number of steps collected per rollout
`gamma`	0.99	Discount factor for future rewards
`gae_lambda`	[0.70, 0.99]	λ for GAE, dynamically updated based on value loss
`m`	$0.09 \to \sim 0.005$	Step-size for updating λ over training
`policy_frequency`	2	Number of critic updates per actor update (TD3-inspired)
`update_epochs`	10	Number of epochs per policy update
`num_minibatches`	32	Number of minibatches per update cycle
`norm_adv`	True	Whether to normalize advantages before policy loss calculation
`ent_coef`	0.0	Coefficient for entropy regularization
`vf_coef`	0.5	Coefficient for value function loss
`clip_coef`	0.2	Clipping parameter for PPO surrogate loss
`clip_vloss`	True	Enables clipping for value function updates
`max_grad_norm`	0.5	Gradient clipping threshold
`target_kl`	None	Optional KL-divergence threshold for early stopping

- *Policy update frequency:* Unlike classical PPO, where the actor and critic networks are updated simultaneously, the proposed version adopts the update strategy from TD3. Specifically, for every policy (actor) update, the critic network is updated twice. This adjustment aims to enhance the stability of value estimation.

- *Range and scheduling of λ:* The λ parameter in GAE is dynamically adjusted within the range $[0.70, 0.99]$ throughout training. The rate of change (denoted as m in the table) starts from an initial value of 0.09 and gradually decreases to around 0.005.

This unified setup ensures fairness and reliability in the comparison between the proposed method and the reference benchmarks. It is worth noting that all four primary benchmark tasks are conducted using exactly the same set of hyperparameters.

5 Conclusion

Based on the explanations provided and the experimental results obtained, we conclude that the adaptive adjustment of the λ parameter, compared to using a fixed value, can lead to a more effective and dynamic balance between bias and variance.

In addition, the concept of PUD has been incorporated into the proposed method to reduce approximation errors and improve training stability. Our findings indicate that applying a delay in policy updates can be beneficial, particularly when the critic network is still in the process of learning and adapting.

A key contribution of this study is the idea that RL algorithms should be capable of automatically adjusting sensitive hyperparameters such as λ, without requiring manual tuning for each specific task — a process that is often time-consuming and costly, especially in high-dimensional environments. To achieve this, our method dynamically adjusts λ based on internal learning signals such as the value loss. This process is further supported by monitoring EV, a statistical measure that reflects how well the critic captures the variations in the target returns.

When EV is low, the critic lacks the accuracy needed to provide reliable predictions, and the use of adaptive λ may lead to suboptimal performance. Conversely, when EV is high, the critic provides more trustworthy feedback, allowing adaptive λ to significantly improve learning outcomes. This highlights the potential of using indicators like EV as effective guidance for determining when and how to apply adaptive strategies.

In summary, this work represents a step toward automated bias-variance control in reinforcement learning, aiming to reduce the dependency on manual hyperparameter tuning and to enhance algorithmic stability across a range of tasks and environments.

Conflicts of interest

The authors declare that there are no conflicts of interest.

References

[1] S. Fujimoto, H. Hoof, D. Meger, *Addressing function approximation error in actor-critic methods*, In International conference on machine learning (2018) 1587–1596.

[2] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, Y. Tassa, *Learning continuous control policies by stochastic value gradients*, Adv. Neural Inf. Process. Syst. **28** (2015) 2944–2952.

[3] S. Kakade, J. Langford, *Approximately optimal approximate reinforcement learning*, In Proceedings of the nineteenth international conference on machine learning (2002) 267–274.

[4] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, *Continuous control with deep reinforcement learning* (2015) arXiv:1509.02971.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, *Human-level control through deep reinforcement learning*, Nature **518** (2015) 529–533.

[6] N. Mohammadpour, M. Fozi, M.M. Ebadzadeh, A. Azimi, A.K. Iglie, *Proximal policy optimization with adaptive generalized advantage estimate*, In proceedings of the first international conference on machine learning and knowledge discovery (2024) 453–458.

[7] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, *Trust region policy optimization*, In International conference on machine learning (2015) 1889–1897.

[8] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, *High-dimensional continuous control using generalized advantage estimation* (2015) arXiv:1506.02438.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, *Proximal policy optimization algorithms* (2017) arXiv:1707.06347.

[10] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Second edition, 2018.

[11] E. Todorov, T. Erez, Y. Tassa, *MuJoCo: A physics engine for model-based control*, In 2012 IEEE/RSJ international conference on intelligent robots and systems, IEEE (2012) 5026–5033.

[12] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, Y. Tassa, *dm_control: Software and tasks for continuous control*, Softw. Impacts **6** (2020) 10022.