# Bridging Physics-Informed Neural Networks with Reinforcement Learning: Hamilton-Jacobi-Bellman Proximal Policy Optimization (HJBPPO)

**Amartya Mukherjee** [*] [1] **Jun Liu** [1]

## Abstract

This paper introduces the Hamilton-Jacobi-Bellman Proximal Policy Optimization (HJBPPO) algorithm into reinforcement learning. The Hamilton-Jacobi-Bellman (HJB) equation is used in control theory to evaluate the optimality of the value function. Our work combines the HJB equation with reinforcement learning in continuous state and action spaces to improve the training of the value network. We treat the value network as a Physics-Informed Neural Network (PINN) to solve for the HJB equation by computing its derivatives with respect to its inputs exactly. The Proximal Policy Optimization (PPO)-Clipped algorithm is improvised with this implementation as it uses a value network to compute the objective function for its policy network. The HJBPPO algorithm shows an improved performance compared to PPO on the MuJoCo environments.

## 1. Introduction

In discrete-time Reinforcement Learning (RL), the value function estimates returns from a given state as a sum of the returns over time steps. This value function is obtained by solving the Bellman Optimality Equation. On the other hand, in continuous-time RL, the value function estimates returns from a given state as an integral over time. This value function is obtained by solving a partial differential equation (PDE) known as the Hamilton-Jacobi-Bellman (HJB) equation (Munos, 1999). Currently existing algorithms in the RL literature such as Proximal Policy Optimization (PPO) update the value function using the Bellman Optimality Equation to estimate the discrete-time returns for each state. We discovered that this value function, when trained on MuJoCo environments, does not show convergence towards the optimal value function as described by the HJB equation (see Figure 2 in Appendix B). This shows that information is lost when the value function is trained using discrete-time methods.

Physics-informed neural networks (PINNs) was introduced by Raissi et al. (2019) and leverage auto-differentiation to compute derivatives of neural networks with respect to their inputs and model parameters exactly. This enables the laws of physics (described by ODEs or PDEs) governing the dataset of interest to act as a regularization term for the neural network. PINNs outperform regular neural networks on such datasets by exploiting the underlying physics of the data.

To the best of our knowledge, this paper is the first to examine the intersection between PINNs and RL. We utilize PINNs to encode the HJB equation to train the value function and bridge the information gap between returns computed over discrete time and continuous time. We propose the Hamilton-Jacobi-Bellman Proximal Policy Optimization (HJBPPO) algorithm, which demonstrates superior performance in terms of higher rewards, faster convergence, and greater stability compared to PPO on MuJoCo environments, making it a significant improvement.

## 2. Preliminaries

Consider a controlled dynamical system modeled by the following equation:

$$\dot{x} = f(x,u), \quad x(t_0) = x_0, \tag{1}$$

where $x(t)$ is the state and $u(t)$ is the control input. In control theory, the optimal value function $V^*(x)$ is useful towards finding a solution to control problems (Munos et al., 1999):

$$V^*(x) = \sup_u \int_{t_0}^{\infty} \gamma^t R(x(\tau; t_0, x_0, u(\cdot)), u(\tau)) d\tau, \tag{2}$$

where $R(x,a)$ is the reward function and $\gamma$ is the discount factor. The following theorem introduces a criteria for

[*]Equal contribution [1]Department of Applied Mathematics, University of Waterloo, Waterloo, Ontario, Canada. Correspondence to: Amartya Mukherjee <a29mukhe@uwaterloo.ca>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023. Copyright 2023 by the author(s).

assessing the optimality of the value function [(Liberzon, 2012), (Kamalapurkar et al., 2018)].

**Theorem 2.1.** *A function $V(x)$ is the optimal value function if and only if:*

1. *$V \in C^1(\mathbb{R}^n)$ and $V$ satisfies the Hamilton-Jacobi-Bellman (HJB) Equation*

$$V(x)\ln\gamma + \sup_{u\in U}\{R(x,u) + \nabla_x V^T(x) f(x,u)\} = 0 \tag{3}$$

*for all $x \in \mathbb{R}^n$.*

2. *For all $x \in \mathbb{R}^n$, there exists a controller $u^*(\cdot)$ such that:*

$$\begin{aligned} &V(x)\ln\gamma + R(x,u^*(x)) + \nabla_x V^T(x) f(x,u^*(x)) \\ &= V(x)\ln\gamma + \sup_{\hat{u}\in U}\{R(x,\hat{u}) + \nabla_x V^T(x) f(x,\hat{u})\}. \end{aligned} \tag{4}$$

Currently existing algorithms in RL such as PPO do not focus on solving the HJB equation to maximize the total reward for each episode. To show this, we define the HJB loss at each episode as the following:

$$\begin{aligned} &\widehat{MSE_f} \\ &= \frac{1}{T}\sum_{t=0}^{T-1} |V(x_t)\ln\gamma + R(x_t,a_t) + \nabla_x V^T(x_t) f(x_t,a_t)|^2, \end{aligned} \tag{5}$$

where $T$ is the number of timesteps in the episode, $x_t$ is the state of the environment at timestep $t$, and $a_t$ is the action taken at timestep $t$. The gradient $\nabla_x V^T(x_t)$ is computed exactly using auto-differentiation. We approximate $f(x_t, a_t)$ using finite differences:

$$\begin{aligned} MSE_f = \frac{1}{T}\sum_{t=0}^{T-1} &|V(x_t)\ln\gamma + R(x_t,a_t) \\ &+ \nabla_x V^T(x_t)\left(\frac{x_{t+1}-x_t}{\Delta t}\right)|^2, \end{aligned} \tag{6}$$

where $\Delta t$ is the time step size used in the environment. We have plotted the HJB loss for each environment using PPO in Figure 2 in Appendix B. The mean HJB loss for each environment takes extremely high values and does not show convergence in 6 out of 10 of the environments, thus showing that the value function does not converge to the optimal value function as shown by the HJB equation.

As a comparison, we have plotted the graphs for the value network loss in Figure 3 in Appendix B. The Bellman optimality loss shows convergence in 8 out of the 10 environments. This shows that information is lost when we solve the Bellman optimality equation for a discrete-time value function compared to a continuous-time value function. It also shows that convergence of the value function does not necessarily lead to convergence in the HJB loss.

To solve this problem as shown in Figure 2, we treat the value network as a PINN and use gradient-based methods to reduce the HJB loss.

## 3. Related Work

The use of HJB equations for continuous RL has sparked interest in recent years among the RL community as well as the control theory community and has led to promising works. Kim et al. (2021) introduced an alternate HJB equation for Q Networks and used it to derive a controller that is Lipschitz continuous in time and shows improved performance over Deep Deterministic Policy Gradient (DDPG) without the need for an actor-network. Wiltzer et al. (2022) introduced a distributional HJB equation to train the FD-WGF Q-Learning algorithm. This models return distributions more accurately compared to Quantile Regression TD (QTD) for a particle-control task.

The use of neural networks to solve the HJB equation has been an area of interest across multiple research projects. Jiang et al. (2016) uses a structured Recurrent Neural Network to solve for the HJB equation and achieve optimal control for the Dubins car problem. Tassa & Erez (2007) uses the Pineda architecture (Pineda, 1987) to estimate partial derivatives of the value function with respect to its inputs. They used iterative least squares method to solve the HJB equation. Sirignano & Spiliopoulos (2018) develops the DGM algorithm to solve PDEs. They use auto-differentiation to compute first-order derivatives and Monte Carlo methods to estimate higher-order derivatives. This algorithm was used to solve the HJB equation to achieve optimal control for a stochastic heat equation and achieved an error of $0.1\%$. Nakamura-Zimmerer et al. (2020) used a PINN to solve the HJB equation in an optimal feedback control problem setting. The paper achieves results similar to that of the true optimal control function in high-dimensional problems.

## 4. HJBPPO

To our knowledge, our work is the first to combine the HJB equation with a currently existing RL algorithm, PPO. It is also the first to use a PINN to solve the HJB equation in an RL setting. The PPO-Clipped algorithm is improvised with this implementation because it uses a value network to compute advantages used to update its policy network (Schulman et al., 2017). PPO is an actor-critic method that limits the update of the policy network to a small trust region at every iteration. It shows state-of-the-art performance in the RL literature.

Our work combines the HJB equation with reinforcement learning in continuous state and action spaces to improve the training of the value network. We treat the value network as a PINN to solve for the HJB equation by computing its derivatives with respect to its inputs exactly. Note that the term

$$\sup_{\hat{u}\in U}\{R(x,\hat{u}) + \nabla_x V^T(x) f(x,\hat{u})\}$$

in the HJB equation cannot be determined without exploration of the agent in its environment. From Theorem 2.1, the optimal policy $\pi^*(a|x)$ and the optimal controller $u^*(x) = \text{argmax}_a \pi^*(a|x)$ satisfies equation (4). The optimal policy is modeled by the policy network $\pi_\theta$ parameterized by $\theta$ and the optimal controller can be approximated using $u(x) = \text{argmax}_a \pi_\theta(a|x)$. As a result, we can use the following approximation:

$$\begin{aligned} &V(x)\ln\gamma + R(x,u(x)) + \nabla_x V^T(x) f(x,u(x)) \\ &\approx V(x)\ln\gamma + \sup_{\hat{u}\in U}\{R(x,\hat{u}) + \nabla_x V^T(x) f(x,\hat{u})\}. \end{aligned} \quad (7)$$

This, as a result, justifies the use of equation 6 as the HJB loss used to update the value function at each episode. The loss function is computed as

$$J(\phi) = 0.5MSE_u + \lambda_{HJB} MSE_f,$$

where $MSE_f$ is defined in equation (6) and $MSE_u$ is the standard loss function for the value network used in PPO, and is used to improve the discrete-time estimate of returns for the value function:

$$MSE_u = \frac{1}{T}\sum_{t=0}^{T-1} |V(x_t) - (R(x_t,a_t) + \gamma V(x_{t+1}))|^2, \quad (8)$$

where $\{x_t\}_{t=1}^T$ is a batch of states explored in a single episode, and $\{a_t\}_{t=1}^T$ is a batch of actions executed at time step $t$ following the policy $\pi_\theta$. The hyperparameter $\lambda_{HJB}$ is determined by hand as the magnitude of the ratio of the Bellman optimality loss to the HJB loss so that both losses are given equal weight. Similar to PINNs, $MSE_f$ is the physics loss that forces the value function to converge to the solution of the HJB equation and $MSE_u$ gives data points that correspond to initial conditions for the value network.

### 4.1. Algorithm

The HJBPPO algorithm is provided in Algorithm 1. The policy update and the minimization of $MSE_u$ for the value network is identical to PPO. In order to satisfy $V(x) \in C^1(\mathbb{R}^n)$ as stated in Theorem 2.1, we use the infinitely differentiable $tanh$ activation function for the value network.

Lines 3–7 in the algorithm are identical to the PPO algorithm. The advantage term $A_t$ is computed as: $A_t = \sum_{n=t}^{T-1}(\gamma\lambda)^{n-t}\delta_n$, where $\delta_t = R_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$, $\gamma$ is the discount factor, and $\lambda$ is the Generalized Advantage Estimation (GAE) parameter. $\alpha_1$ and $\alpha_2$ are learning rates for the policy network optimizer and value network optimizer respectively. $\epsilon$ is the clipping parameter. Lines 8–9 in the algorithm are our modifications to the PPO algorithm. We treat the value network as a PINN and add a $MSE_f$ term into its loss function. This way, the HJB equation is used as a regularization term for the value network.

**Algorithm 1** HJBPPO

```
1:  Initiate policy network parameter $\theta$ and value network
    parameter $\phi$
2:  for $iteration = 1, 2, ...$ do
3:      Run the policy $\pi_\theta$ in the environment for $T$ timesteps
        and observe samples $\{(s_t, a_t, R_t, s_{t+1})\}_{t=1}^T$.
4:      Compute the advantage $A_t$
5:      Compute $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
6:      Compute the objective function of the policy network:
        $L(\theta) = \frac{1}{T}\sum_{t=0}^{T-1}\min[r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t]$
7:      Update $\theta \leftarrow \theta + \alpha_1 \nabla_\theta L(\theta)$
8:      Compute the value network loss as: $J(\phi) = 0.5MSE_u + \lambda_{HJB} MSE_f$ described in equations
        (8) and (6)
9:      Update $\phi \leftarrow \phi - \alpha_2 \nabla_\phi J(\phi)$
10: end for
```

## 5. Results

The HJBPPO algorithm was implemented by modifying the code for PPO in the Stable Baselines 3 library by Raffin et al. (2021). To ensure the reproducibility of our results, we have posted our code at https://github.com/amartyamukherjee/stable-baselines3 and provided our hyperparameters in Appendix C. The code was run on the Béluga cluster in Compute Canada. The cluster provided the MuJoCo environments for training. Training each algorithm over 1 million time steps took seven hours, and training over 10 million time steps took three days. The multiprocessing library from Python was used to train each algorithm over multiple environments at the same time.

The reward graphs have been plotted in Figure 1, comparing HJBPPO to PPO on all the MuJoCo environments over a million time steps or ten million time steps. The line shows the total reward, averaged over 50 consecutive episodes, and the shaded area indicated the standard deviation of the total reward over 50 consecutive episodes. HJBPPO shows a significant improvement in Ant-v4 and HalfCheetah-v4. It shows faster convergence and stability in Reacher-v4, Swimmer-v4, and InvertedDoublePendulum-v4. And it shows a slight improvement in HumanoidStandup-v4, Hopper-v4, and Walker2d-v4. And it performs equally as well as PPO in the

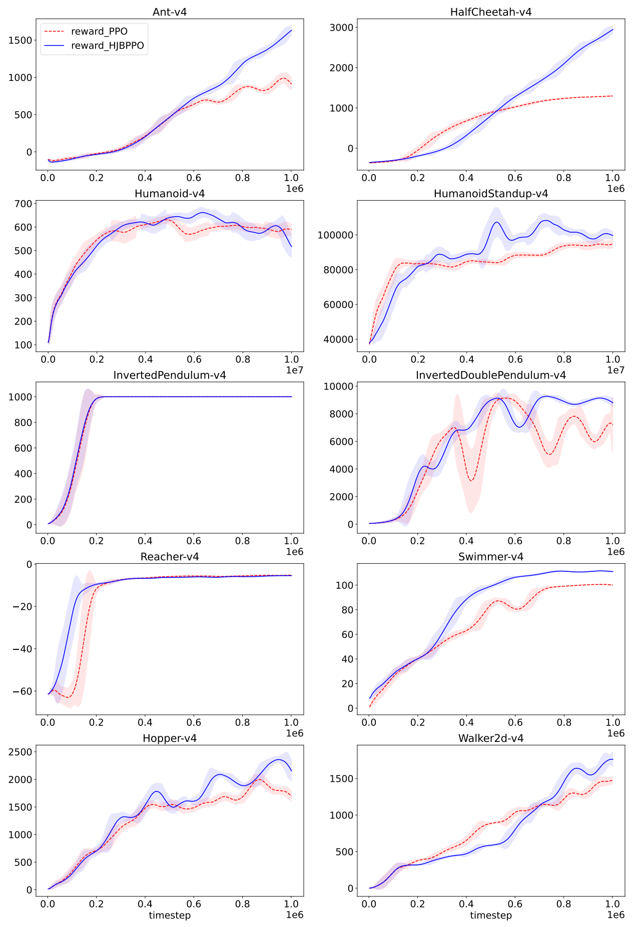

*Figure 1.* Comparison of learning curves for PPO (Red) compared to HJBPPO (Blue)

remaining environments. For the two remaining environments (Humanoid-v4 and InvertedPendulum-v4), it shows equal performance to PPO.

As a result, the graphs show that incorporating the HJB equation into the PPO algorithm to train the value function leads to an improved learning curve for the agent. This is because HJBPPO uses a PINN to exploit the physics in the environment. It uses finite differences to approximate the underlying governing equation $f(x, u)$ of the environment and uses auto-differentiation to solve the HJB equation to achieve optimal control.

The HJB loss for each environment has been plotted in Figure 4 in Appendix C. HJBPPO shows a significant decrease in the HJB loss compared to PPO. And the HJB loss shows convergence in 8 out of the 10 environments, including Ant-v4 and HalfCheetah-v4 where it performs significantly better than PPO in terms of rewards, thus showing that the value function converges to the optimal value function as shown in the HJB equation.

The HJB loss does not converge for Humanoid-v4 and HumanoidStandup-v4, even though the HJB loss for these environments is significantly lower than that as shown in Figure 2. In both of these environments, the reward curve for HJBPPO shows similar performance to PPO. This shows that HJBPPO does not show significantly improved performance compared to PPO in general if the HJB loss does not show convergence.

The value loss for each environment has been plotted in Figure 5 in Appendix C. The value network shows convergence in every environment, including Ant-v4 and Walker2d-v4, where PPO doesn't show convergence. This shows that convergence of the value function in the continuous-time HJB equation also improves its convergence in the discrete-time Bellman optimality equation, while the converse may not necessarily be true.

## 6. Conclusion

In this paper, we have introduced the HJBPPO algorithm that improves the PPO algorithm to solve the HJB equation. This paper is the first of its kind to combine PINNs with RL. We treat the value function as a PINN to solve the HJB equation in an RL setting. The HJBPPO algorithm shows an overall improvement in performance compared to PPO due to its ability to exploit the physics of the environment as well as optimal control to improve the learning curve of the agent. This paper also shows that convergence of the value function in the continuous-time HJB equation also improves its convergence in the discrete-time Bellman optimality equation.

## 7. Future Research

Despite showing an overall improvement in the reward curves, the HJBPPO leaves room for improved RL algorithms using PINNs.

The loss function $MSE_f$ used in training the value network was derived as a result of the approximation used in equation 7. So this does not guarantee convergence of the policy network towards the optimal policy such that $u(x) = \sup_{\hat{u} \in U}\{R(x, \hat{u}) + \nabla_x V^T(x) f(x, \hat{u})\}$ where the controller $u(x)$ is derived from the policy $\pi_\theta(a|x)$. Holzleitner et al. (2021) proves the convergence of the policy network parameters in PPO to a local optimum but it does not guarantee global convergence. Thus, a potential area of further research could involve alternate choices of HJB loss functions for the value network that relaxes this approximation.

Additionally, finite difference approximations become less accurate in environments with high dimensions (Sirignano & Spiliopoulos, 2018). This makes the HJB loss less reliable in environments such as Humanoid-v4 and HumanoidStandup-v4 where the state is a 376-dimensional vector. The finite difference approximation does not compute $f(x,u)$ exactly because the environment uses semi-implicit Euler integration steps rather than Euler's method (Tassa et al., 2012). Thus, a potential area for further research could be combining HJBPPO with model-based RL so that $f(x,u)$ can be estimated with a smaller error.

In the MuJoCo environments, the HJBPPO algorithm showed an improvement compared to PPO due to the fact that $f(x,u)$ could be estimated through finite differences, thus allowing for the physics of the environment to be exploited. The environments give all the details of the state needed to choose an action. Another limitation of HJBPPO is that it may not perform well in partially observable environments because the estimate of $f(x,u)$ may be inaccurate. Deep Transformer Q Network (DTQN) was introduced by Esslinger et al. (2022) and achieves state-of-the-art results in many partially observable environments. A potential area for further research may be the introduction of an HJB equation that facilitates partial observability. The DTQN algorithm may be improvised by incorporating this HJB equation using PINNs.

## References

Esslinger, K., Platt, R., and Amato, C. Deep transformer q-networks for partially observable reinforcement learning. *Preprint arXiv:2206.01078*, 2022.

Holzleitner, M., Gruber, L., Arjona-Medina, J., Brandstetter, J., and Hochreiter, S. Convergence proof for actor-critic methods applied to ppo and rudder. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLVIII*, pp. 105–130. Springer, 2021.

Jiang, F., Chou, G., Chen, M., and Tomlin, C. J. Using neural networks for fast reachable set computations. *Preprint arXiv:611.03158*, 2016.

Kamalapurkar, R., Rosenfeld, J., Dixon, W. E., and Walters, P. *Reinforcement Learning for Optimal Feedback Control: A Lyapunov-Based Approach*. Springer Cham, 2018.

Kim, J., Shin, J., and Yang, I. Hamilton–jacobi deep q-learning for deterministic continuous-time systems with lipschitz continuous controls. *Journal of Machine Learning Research*, 22:1–34, 2021.

Liberzon, D. *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2012.

Munos, R. A Study of Reinforcement Learning in the Continuous Case by the Means of Viscosity Solutions. *Machine Learning*, 40:265–299, 1999.

Munos, R., Baird, L., and Moore, A. Gradient descent approaches to neural-net-based solutions of the hamilton-jacobi-bellman equation. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 3, pp. 2152–2157 vol.3, 1999.

Nakamura-Zimmerer, T., Gong, Q., and Kang, W. A causality-free neural network method for high-dimensional hamilton-jacobi-bellman equations. In *2020 American Control Conference (ACC)*, pp. 787–793, 2020.

Pineda, F. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, 1987.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Raissi, M., Perdikaris, P., and Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *Preprint arXiv:1707.06347*, 2017.

Sirignano, J. and Spiliopoulos, K. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018. ISSN 0021-9991.

Tassa, Y. and Erez, T. Least squares solutions of the hjb equation with neural network value-function approximators. *IEEE Transactions on Neural Networks*, 18(4):1031–1041, 2007.

Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.

Wiltzer, H. E., Meger, D., and Bellemare, M. G. Distributional Hamilton-jacobi-Bellman equations for continuous-time reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 23832–23856. PMLR, 2022.

## A. Hyperparameters

| Hyperparameter | Value |
|---|---|
| Horizon (T) | 2048 |
| Adam stepsize | 3e-04 |
| Num. epochs | 10 |
| Minibatch size | 64 |
| Discount ($\gamma$) | 0.99 |
| GAE parameter ($\lambda$) | 0.95 |

*Table 1.* HJBPPO hyperparameters

| Environment | Value |
|---|---|
| Ant-v4 | 0.1 |
| HalfCheetah-v4 | 0.1 |
| Humanoid-v4 | 1e-04 |
| HumanoidStandup-v4 | 1.0 |
| InvertedPendulum-v4 | 1e-04 |
| InvertedDoublePendulum-v4 | 1e-03 |
| Reacher-v4 | 1.0 |
| Swimmer-v4 | 1e-04 |
| Hopper-v4 | 0.1 |
| Walker2d-v4 | 0.1 |

*Table 2.* $\lambda_{HJB}$ hyperparameter for each environment

## B. PPO Loss Curves

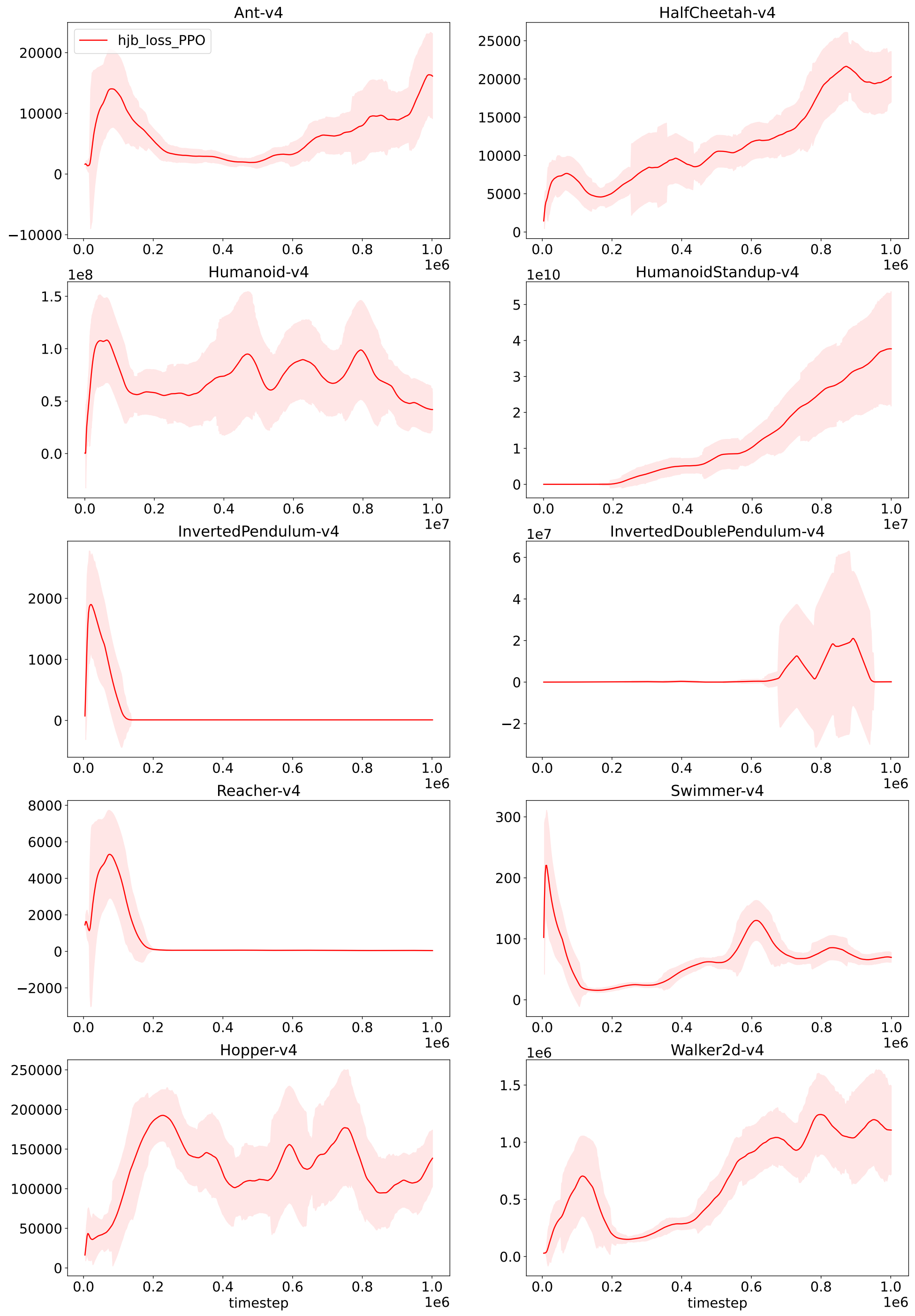

*Figure 2.* HJB loss curves for PPO on MuJoCo environments

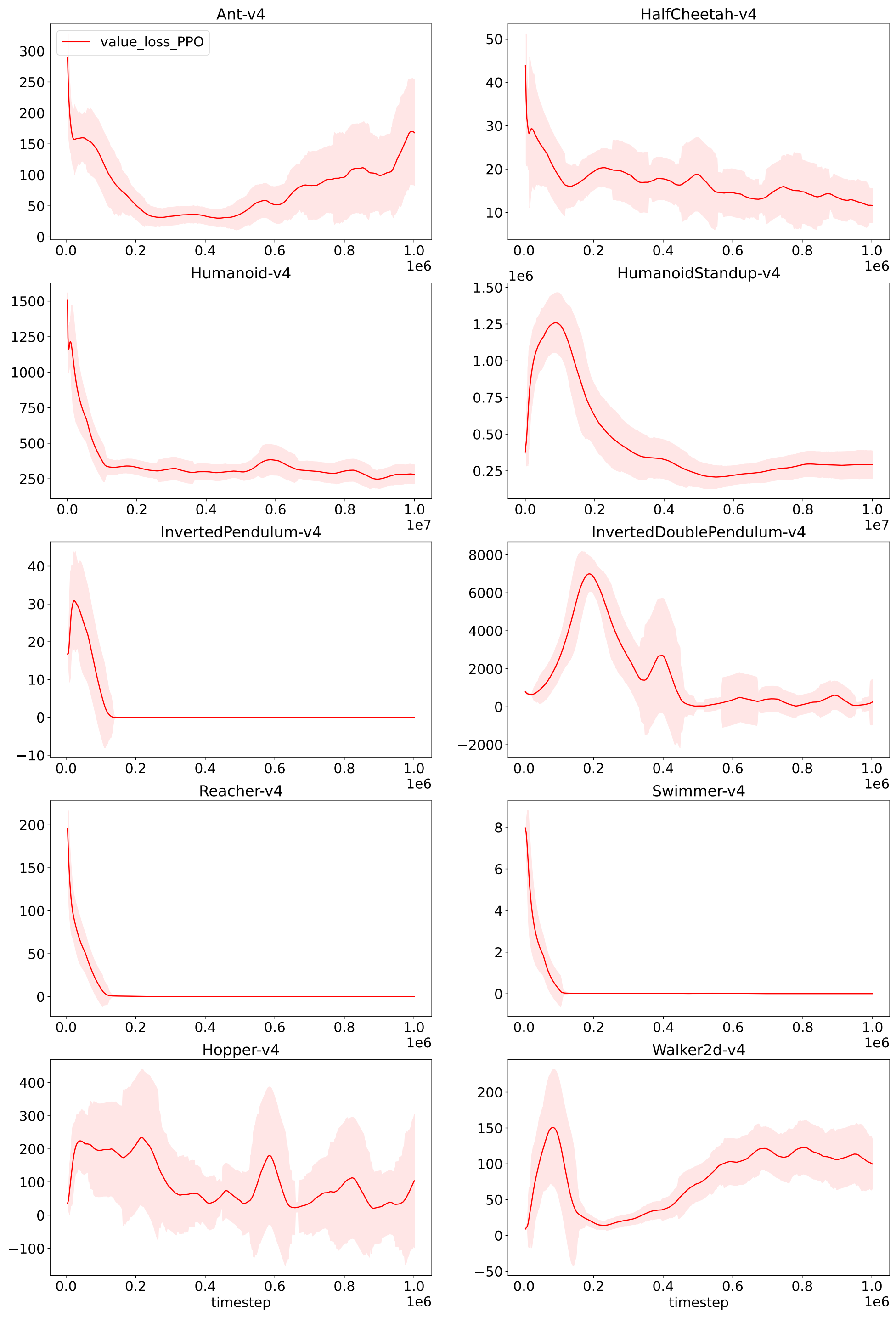

*Figure 3.* Bellman optimality loss curves for PPO on MuJoCo environments

## C. HJBPPO Loss Curves

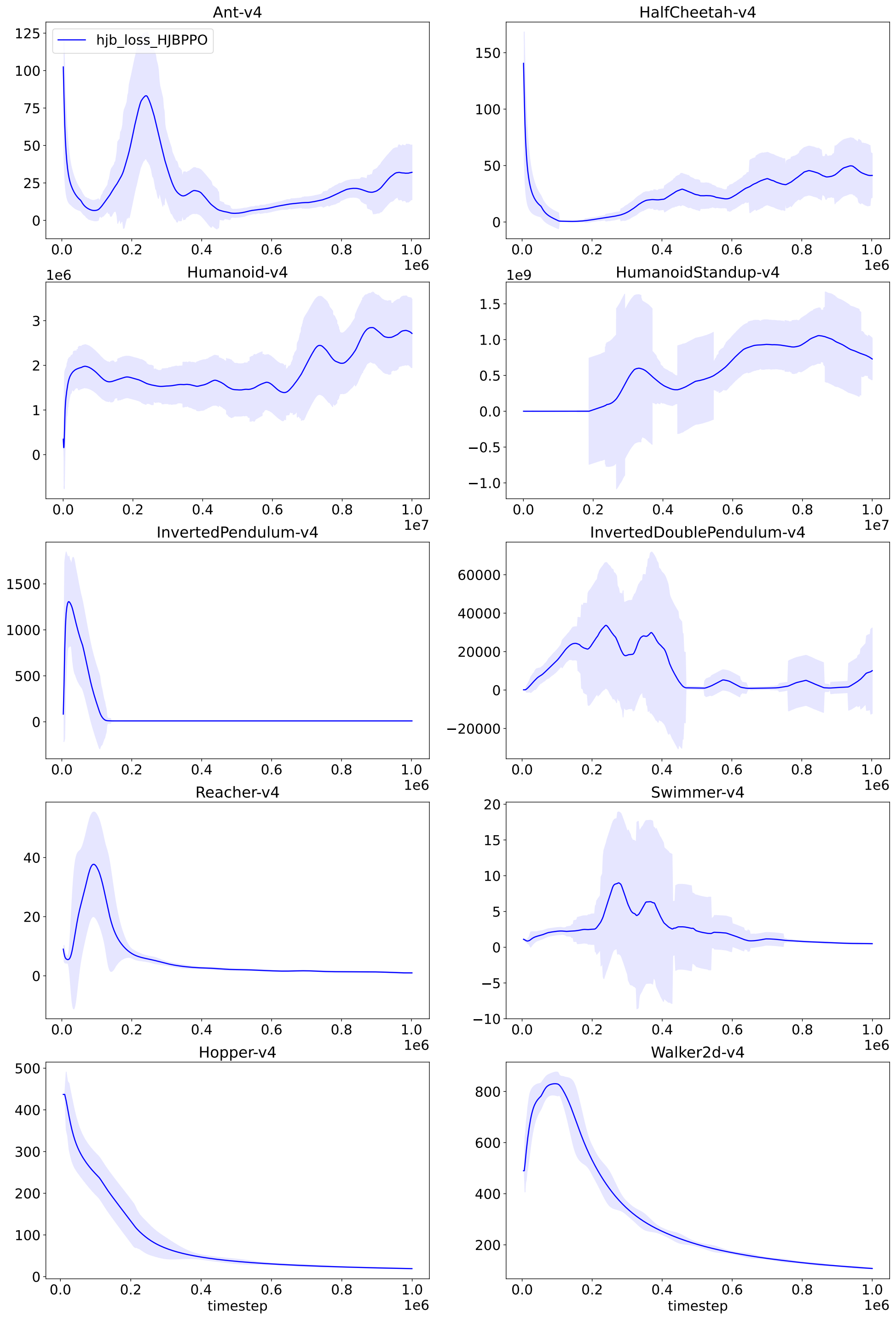

*Figure 4.* HJB loss curves for HJBPPO on MuJoCo environments

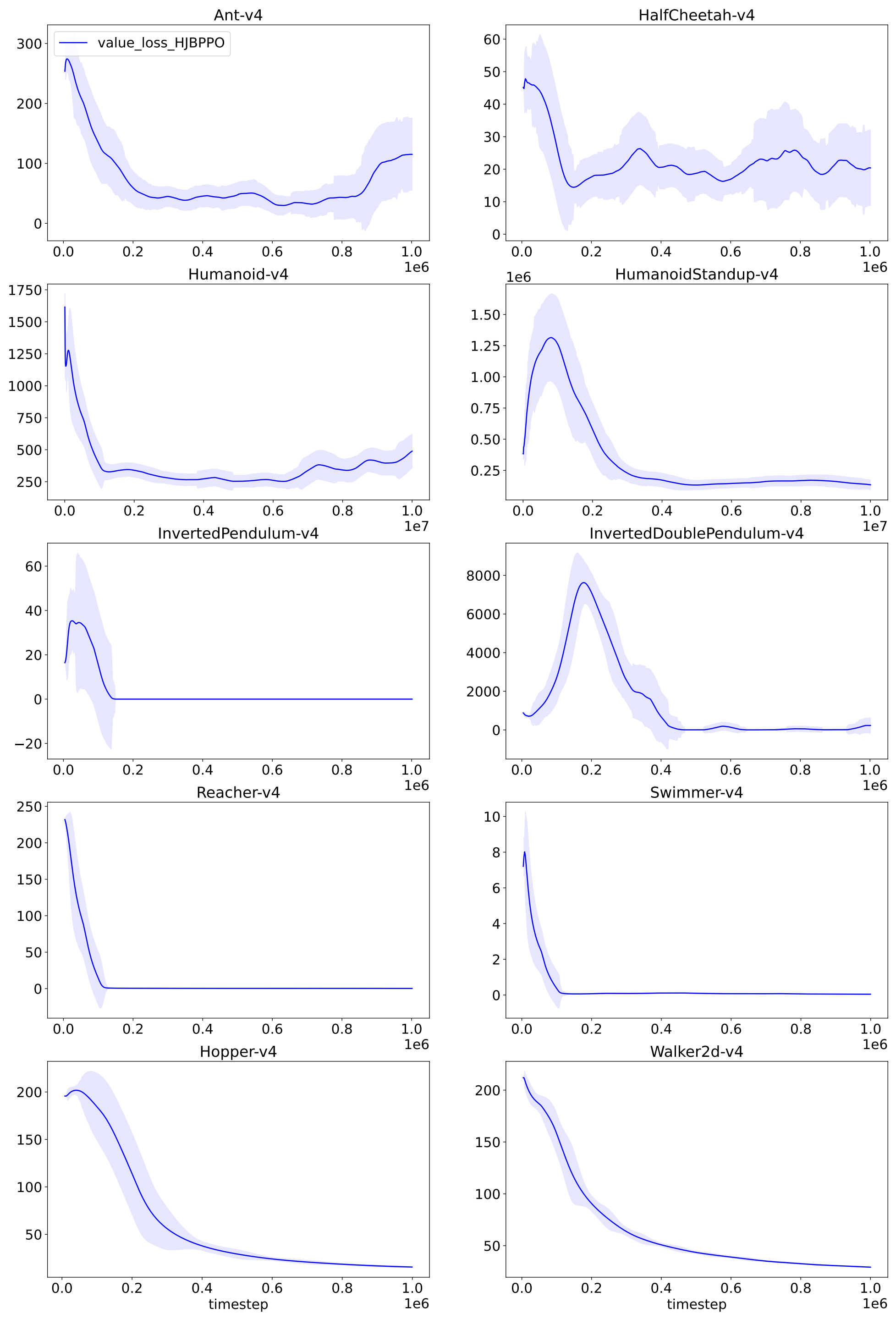

*Figure 5.* Bellman optimality loss curves for HJBPPO on MuJoCo environments