Improving the Representation Learning Ability of PPO Algorithm by Interleaving States and Actions

Tianze Luo
IIIS, Tsinghua University
2022012084

Xingchen Miao
IIIS, Tsinghua University
2022012203

Abstract

In this work, we significantly improve the data efficiency and performance of the Proximal Policy Optimization (PPO) algorithm on HumanoidBench, a benchmark for robot learning in high-dimensional simulated environments. HumanoidBench features a humanoid robot with dexterous hands and presents a variety of challenging tasks involving both whole-body manipulation and locomotion. According to the original HumanoidBench paper, the PPO algorithm fails to learn any practical policy within 10 million sample steps, even on relatively simple tasks like walking. To enhance the sample efficiency and performance of PPO, we improve its representation learning capability by sequentially modeling the policy-value network with interleaved states and actions as inputs. We also implement several beneficial techniques such as state normalization, reward scaling, Generalized Advantage Estimation (GAE), and carefully tune the hyperparameters, including loss function scales and the training schedule. Our improved PPO algorithm successfully learns effective policies and significantly outperforms the model-free methods reported in the original HumanoidBench paper, including the baseline PPO and Soft Actor-Critic (SAC) algorithms. Moreover, our method achieves performance similar to model-based approaches like DreamerV3 and TD-MPC2, demonstrating its potential for solving complex tasks in humanoid robot learning.

1 Introduction

Humanoid robots, with their human-like morphology, hold great potential in a wide range of applications, from assisting with daily human tasks to functioning autonomously in dynamic environments. However, the development of effective controllers for humanoid robots has been hampered by the high costs and fragility of real-world hardware setups. To mitigate these challenges, HumanoidBench[15] was introduced as a high-dimensional simulated benchmark for robot learning, designed to test algorithms in environments that simulate complex humanoid robot tasks. Built on the MuJoCo[17] physics engine, HumanoidBench features a humanoid robot (Unitree H1) equipped with dexterous hands, and supports a variety of locomotion and manipulation tasks. These tasks span from basic walking and reaching to more intricate whole-body manipulations like unloading packages, cleaning windows, and playing basketball. With these features, HumanoidBench serves as both a testbed for evaluating robot learning algorithms and a platform for advancing research in humanoid robotics, offering the research community a controlled, safe, and cost-effective means to benchmark algorithms in complex, long-horizon tasks. The challenges faced by algorithms on this platform include not only the complexity of robot dynamics but also the need for coordination across multiple body parts and the intricacies of high-dimensional action spaces.

Previous research on robotic simulation has predominantly concentrated on tasks characterized by simpler dynamics and shorter time horizons. These tasks are often limited to single-arm manipulation or relatively straightforward locomotion problems. While benchmarks such as Robosuite and other bimanual manipulation environments have made significant progress in addressing coordination challenges, they still fall short of capturing the intricate full-body coordination required by humanoid robots. Some benchmarks have incorporated long-horizon tasks, like block stacking or furniture assembly, but these remain constrained in their ability to allow robots to engage with their environment as holistic, fully integrated systems. To address these complexities, hierarchical reinforcement learning (HRL)[11] has been explored as a strategy to separate low-level control from high-level decision-making, which can aid in managing long-horizon tasks. However, scaling HRL to humanoid robots presents additional challenges, as low-level RL typically requires extensive parallel pretraining, while high-level RL demands intricate coordination across a multitude of degrees of freedom.

One of the core challenges in humanoid robot learning lies in the efficient utilization of data. The high-dimensional nature of humanoid systems, coupled with the complexity of their environments, often results in learning processes that are both slow and data-intensive. Model-based approaches, such as TD-MPC2[9] and DreamerV3[8], have demonstrated progress in addressing these issues by learning and leveraging world models. In contrast, model-free methods, such as Soft Actor-Critic (SAC)[6] and Proximal Policy Optimization (PPO)[14], are constrained by their inability to efficiently reuse past data, as they lack a learned world model and rely solely on real samples collected during training.

The limitations of PPO are particularly pronounced due to its on-policy nature, which prevents the algorithm from repeatedly reusing data stored in a replay buffer – a capability that off-policy methods like SAC employ effectively. This inability to make multiple passes over the same data exacerbates the challenges of scaling PPO to complex tasks. On benchmarks such as HumanoidBench, which demand both precise high-dimensional control and robust long-horizon decision-making, these deficiencies render PPO insufficient for achieving practical performance. Indeed, PPO often fails even on relatively simple tasks, such as basic walking, further highlighting its limitations in addressing the unique challenges posed by humanoid robots.

In this work, we address the shortcomings of PPO on HumanoidBench by improving its data efficiency and performance. We propose a novel approach to PPO by incorporating a sequential policy-value network, where states and actions are interleaved as inputs, thereby enhancing the network's representation learning capability. This approach allows the model to capture the complex dependencies between states and actions in long-horizon tasks, improving both stability and efficiency in learning. By tuning key hyperparameters and integrating techniques such as state normalization, reward scaling, and Generalized Advantage Estimation (GAE)[13], our method significantly outperforms existing model-free methods including SAC and origin PPO, which achieves similar performances to state-of-the-art model-based algorithms like DreamerV3 and TD-MPC2.

2 Related Work

Proximal Policy Optimization (PPO) is a widely adopted reinforcement learning algorithm, renowned for its simplicity and effectiveness across a wide range of tasks. PPO improves upon traditional policy gradient methods by introducing a surrogate objective function that constrains policy updates to remain within a stable range. The key innovation of PPO lies in its "clipped" objective, which prevents the new policy from deviating excessively from the old one during updates. By bounding the ratio of new to old action probabilities, PPO mitigates instability often associated with large policy shifts. Despite its widespread success, PPO's on-policy, model-free nature limits its ability to reuse past experiences, resulting in inefficiencies when tackling complex, high-dimensional tasks, such as those found in the HumanoidBench benchmark. These limitations become particularly problematic in scenarios where sample efficiency is crucial. The pseudocode for the PPO algorithm is described in Algorithm 1.

PPO has several strengths that have contributed to its popularity. Its clipped objective ensures stable training by constraining updates, making it particularly effective in environments with high reward variance. Moreover, PPO is relatively easy to implement compared to alternatives like Trust Region Policy Optimization (TRPO), which requires a more complex constraint handling mechanism. PPO's versatility has also been demonstrated across a wide variety of tasks, ranging from robotic control to video games. However, PPO is not without its limitations. Its on-policy nature requires fresh data at every iteration, which limits the reuse of past experiences and leads to high sample complexity. Additionally, PPO is highly sensitive to hyperparameter choices, such as clipping thresholds, learning

Algorithm 1 Proximal Policy Optimization (PPO)

```
1: Initialize: Policy $\pi$, value function $V$, and hyperparameters such as clipping threshold $\epsilon$.
2: for each iteration do
3:     Collect Rollouts: Interact with the environment using policy $\pi$, collecting $N$ samples of $(s_t, a_t, r_t)$.
4:     Advantage estimation (GAE can be replaced by other method):
5:     for each sample $t$ in the rollout do
6:         Compute reward-to-go: $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$.
7:         Estimate advantage $A_t$ using Generalized Advantage Estimation (GAE):
               $A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$
           where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$.
8:     end for
9:     for each minibatch in the sampled data do
10:        Compute new action log probabilities: $\log \pi(a_t|s_t)$.
11:        Compute probability ratio:
               $\text{ratio} = \frac{\pi_{\text{new}}(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} = \exp(\log \pi_{\text{new}} - \log \pi_{\text{old}}).$
12:        Clip the probability ratio:
               $\text{clipped\_ratio} = \min(\max(\text{ratio}, 1-\epsilon), 1+\epsilon).$
13:        Calculate policy loss:
               $\text{policy\_loss} = -\mathbb{E}_t\left[\min(\text{ratio} \cdot A_t, \text{clipped\_ratio} \cdot A_t)\right].$
14:        Calculate value loss (TD-target can be replaced):
               $\text{value\_loss} = \mathbb{E}_t\left[(R_t + \gamma V(s_{t+1}) - V(s_t))^2\right].$
15:        Optionally, calculate entropy loss:
               $\text{entropy\_loss} = -\mathbb{E}_t\left[\pi(a_t|s_t) \cdot \log \pi(a_t|s_t)\right].$
16:        Combine losses into a total objective:
               $\text{total\_loss} = \text{policy\_loss} + c_1 \cdot \text{value\_loss} - c_2 \cdot \text{entropy\_loss}.$
17:        Update $\pi$ and $V$ via backpropagation using the total loss.
18:    end for
19: end for
```

rates, and parameters for advantage estimation, which can significantly affect performance if not carefully tuned. Furthermore, PPO struggles with tasks requiring long-term credit assignment due to its reliance on shorter-horizon advantage estimates.

Soft Actor-Critic (SAC) is a model-free, off-policy reinforcement learning algorithm designed to maximize a reward function while simultaneously encouraging exploration by incorporating a maximum entropy objective. The algorithm utilizes a soft Q-learning approach with two Q-value functions and a target network to ensure stability and avoid overestimation in value estimation. SAC employs a stochastic policy modeled as a Gaussian distribution, which facilitates exploration in continuous action spaces. The entropy term in the objective function serves to maintain stochasticity in the policy, ensuring robust exploration and reducing the likelihood of premature convergence to suboptimal deterministic policies.

The main advantages of SAC lie in its sample efficiency, as it reuses past experiences due to its off-policy nature, and its effectiveness in environments with high-dimensional action spaces. The entropy-based exploration encourages diverse policy behaviors, helping the algorithm excel in complex tasks. Additionally, the use of twin Q-functions and a target network enhances training stability. However, SAC has notable limitations, including high computational costs due to the maintenance of two Q-functions and target networks, sensitivity to hyperparameter tuning such as the entropy coefficient and learning rates, and slower convergence in environments with sparse rewards. These factors can make SAC challenging to implement effectively in certain scenarios.

TD-MPC2 is a state-of-the-art model-based reinforcement learning algorithm that combines learned environment models with online planning to achieve superior decision-making capabilities. It builds a predictive model of the environment and uses this model to perform rollouts and evaluate potential future trajectories. The predictions from the learned model are utilized to compute optimal actions through trajectory optimization methods, allowing the algorithm to perform planning directly in the latent space. TD-MPC2 integrates temporal difference learning with model-based predictions, enabling the agent to balance planning accuracy and adaptability in dynamic environments.

The strengths of TD-MPC2 include its high sample efficiency due to the ability to simulate and evaluate trajectories using the learned environment model, and its suitability for tasks requiring long-term planning and reasoning. Additionally, its capacity for real-time adaptation makes it effective in dynamic or partially observable environments. However, TD-MPC2 heavily depends on the accuracy of the learned model, and any errors in the model can propagate through the planning process, leading to suboptimal decision-making. Furthermore, the computational demands of maintaining and updating the model, as well as performing online planning, can make it resource-intensive. Its performance may also degrade in highly stochastic or complex environments where precise modeling is challenging.

DreamerV3 is a model-based reinforcement learning algorithm that leverages latent space dynamics models to perform learning and planning from imagined trajectories. The algorithm builds a world model using a compact latent representation of the environment, which is trained to predict future states, rewards, and terminal conditions. Imaginary rollouts generated by the world model allow DreamerV3 to optimize its policy without requiring direct interaction with the environment for every learning step. By combining a learned model and model-free policy optimization techniques, DreamerV3 achieves sample-efficient learning and is well-suited for high-dimensional tasks.

The advantages of DreamerV3 include its exceptional sample efficiency, as it relies heavily on model-generated rollouts to reduce interaction with the environment. This efficiency enables it to tackle tasks requiring long-term planning, such as humanoid manipulation, by using its latent dynamics to simulate future outcomes effectively. Additionally, DreamerV3's ability to operate in compact latent spaces reduces computational demands compared to directly modeling raw observations. However, DreamerV3 is limited by the quality of its learned world model; inaccuracies in predictions can significantly impact policy performance. Training the latent dynamics model and optimizing the policy simultaneously introduces complexity, making the algorithm sensitive to hyperparameters and model architecture choices. Moreover, transferring policies from simulation to real-world scenarios remains a challenge due to discrepancies between the simulated and real environments.

3 Methods

3.1 Overview And Model Design

PPO is a well-known reinforcement learning algorithm that has achieved significant success across a variety of tasks. However, on the HumanoidBench benchmark, it struggles to learn any practical policy. We identify two key factors contributing to this challenge. First, PPO is a model-free, on-policy algorithm, which limits its ability to leverage data efficiently. Unlike off-policy methods such as Soft Actor-Critic (SAC) or model-based methods such as DreamerV3, PPO cannot take advantage of trajectory data from a world model, nor can it fully utilize previously sampled data, thereby hindering its performance. Second, the complexity of the HumanoidBench environment exacerbates this issue. The H1 robot in HumanoidBench has an action space with 61 dimensions, while the observations consist of hundreds of dimensions. Additionally, the action frequency is 50Hz, meaning that the relationships between states and actions are long-term, highly complex, and difficult to capture. This makes learning an effective policy with PPO even more challenging, as it requires the model to handle and make decisions based on this intricate temporal and spatial information.

Based on these observations, we propose modeling the policy-value network in a sequential manner, where recent states and actions are interleaved as inputs. This design choice is inspired by the GLM-4-Voice[18], a recent large language model for speech-to-text tasks, which interleaves text and speech tokens during the pretraining phase to enhance the model's multimodal understanding and generative capabilities. By borrowing this approach, we aim to capture the intricate dependencies between states and actions in a more effective and dynamic way. The structure of our policy-value network is shown below, and it stands in stark contrast to conventional architectures such as multi-layer perceptrons (MLPs) and recurrent neural networks (RNNs), offering a novel approach to sequential learning in reinforcement learning environments. To begin with, we utilize only the most recent 8

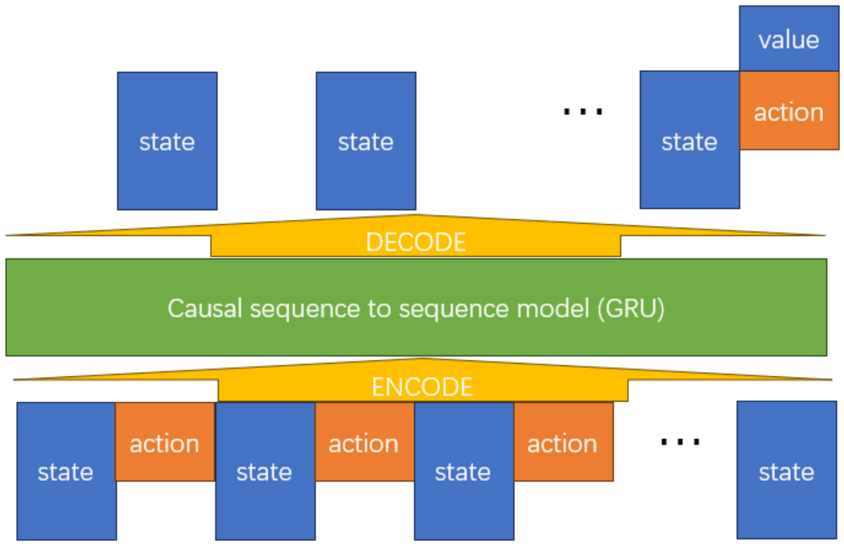

Figure 1: The sequential policy-value network

states and actions to predict the current action and value, striking a good balance between the full Markov prediction typically implemented by multi-layer perceptrons (MLPs) and the full sequential prediction used in recurrent neural networks (RNNs). In our model, both the encoder and decoder are implemented as MLPs, while the sequential modeling component is a two-layer Gated Recurrent Unit (GRU)[2]. The most recent state represents the current state from which we make predictions, and the actions are generated by an MLP utilizing the Beta distribution reparameterization trick. The reason we choose a linearly scaled Beta distribution over a clipped Gaussian distribution is that, in HumanoidBench, all actions are normalized to the range [-1, 1]. The Beta distribution naturally provides bounded outputs, making it a more suitable choice for this setting. The value function is predicted by a separate MLP, which takes the output from the final time step of the GRU as its input.This architecture efficiently combines the benefits of both Markovian and sequential dependencies for enhanced performance.

The core design principle of our approach is to enhance the representation learning capability of the network in order to improve both data efficiency and overall performance. However, we aim to avoid the world-model bias that can arise from inaccurate trajectory predictions, as seen in model-based methods. To mitigate this, we treat state prediction loss solely as an auxiliary loss, helping the model grasp the underlying physical laws of the environment, rather than using it to sample trajectories, which could introduce bias into the policy-value network. Furthermore, model-based methods typically rely on predicting rewards to make decisions. However, in our experiments, we observed a performance drop when our model was tasked with predicting rewards. This degradation may be due to the simultaneous prediction of the value function, where the gradients of the reward predictions could interfere with the model's learning process. Additionally, it is worth noting that, unlike typical sequence modeling, our model does not predict actions. This is because, in reinforcement learning, actions are generated by the model itself, not from the data distribution, making action predictions an unreliable way to guide the network's learning. We also experimented with using the average of previous action predictions to output the final state, but did not observe significant improvements compared to the approach we currently employ.

It is important to note that we combine the tasks of state, action, and value prediction into a single network, which effectively enhances both data efficiency and the representation learning capability of our model. However, this also introduces a challenge in balancing the associated loss functions. To encourage exploration, we include an action entropy loss. Thus, the complete loss function for our network consists of four components in total. After conducting several experiments, we empirically set the coefficients for the state prediction loss, PPO-clip loss, value prediction loss, and action entropy loss to 0.1, 1, 0.3, and 0.05, respectively, in order to stabilize the learning process.

In our view, our method offers three key advantages over the original PPO algorithm. First, our model leverages the environment's dynamics to assist in decision-making and value prediction, while avoiding the world-model bias that can affect model-based approaches. Second, by using the 8 most recent states and actions to predict the current action, our model is better able to learn more consistent actions over long-horizon tasks. Finally, our model achieves much higher data efficiency, as all predictions are made by a unified network with stronger representation power (compared to MLPs). This significantly alleviates the challenges faced by the original PPO algorithm in terms of data efficiency and learning stability.

3.2 Implementation Details

Simply having a working algorithm is not sufficient; we also incorporate several auxiliary techniques to stabilize the learning process. First, we use Generalized Advantage Estimation (GAE) with a decay factor of 0.95 to estimate the advantages in the PPO-clip loss function, and the advantages are normalized again in every batch. Additionally, the reward is scaled by a factor of 0.25, and the discount factor is set to 0.98. Empirically, we found that the target for the value network should not exceed 15, and the reward scaling coefficient of 0.25 is effective for tasks with rewards around 1000 over 1000 steps. It is worth mentioning that, in many cases, reward scaling is performed by standardizing the variance of rewards, allowing the model to automatically adapt to different environments. However, we found that this approach can reduce the model's data efficiency. After normalization, the discrepancies between rewards become dynamically smaller, which can hinder the model's learning efficiency. Since our goal is for the model to learn effectively with only 10 million samples, we found that multiplying by a moderate constant, such as 0.25, offers better data efficiency. For different environments, this constant should be adjusted according to the discount factor and the average reward level to ensure that the value network's target does not exceed 15. The learning rate is initially set to 4e-5 and gradually decays to 1e-5 in a linear fashion during training.

Additionally, due to the significant variation in environments across different tasks in HumanoidBench, we employ state normalization to stabilize the input level of the network and reduce the impact of such environmental differences. Specifically, we maintain a running estimate of the mean and standard deviation of the states during the sampling process. These statistics are then used to normalize the state data before feeding it into the network, ensuring that the input features remain within a consistent range. To further improve training stability and convergence, we adopt orthogonal initialization for the network weights, which helps maintain healthy gradients throughout training. Moreover, we carefully tune the hyperparameters to stabilize the optimization process: the PPO-clip scale is set to 0.2, and the gradient clipping scale is set to 0.5.

All these choices mentioned above ensure that the learning process remains smooth and prevent excessively large updates that could destabilize training. Additionally, for the action predictor, we use an MLP with Beta distribution reparameterization. The alpha and beta parameters are predicted using MLPs, with the Softplus function (+1.0) applied to ensure that these parameters are strictly greater than one. The Beta distribution is then scaled and shifted to the range [-1, 1] to generate the normalized actions. To avoid potential numerical instability, we clamp the values at the boundaries. For further details, please refer to our code.

Additionally, to accelerate the training process, we utilize 32 parallel processes for data sampling, leveraging the SubprocVecEnv() for efficient parallelization. The total training consists of 10 million sample steps, with each round involving the collection of 32 parallel environments, each sampling 1000 steps. This setup provides 32,000 samples per training period. During training, the batch size is set to 500, and each round consists of 10 epochs, meaning the network is updated 640 times per round. Given that our model requires the 8 most recent states and actions to make decisions, our sampling process is more complex compared to the original MLP-based PPO model. To handle this complexity, we maintain a trajectory collector for each parallel environment. This collector ensures that data is formatted correctly for the network from the raw trajectories. Additionally, once a trajectory is completed, the collector facilitates the accumulation of data for the training process. For further details, please refer to the codes.

4 Experiments

(a) Sit-Hard (b) Bookshelf-Simple (c) Truck (d) Walk

Figure 2: Some Visual Results on Four Environments

To evaluate the effectiveness of our improved PPO algorithm, we conducted experiments across four distinct environments: Sit-Hard, Bookshelf-Simple, Truck, and Walk. These environments were chosen to represent a variety of tasks, ranging from basic manipulation to more complex locomotion and coordination challenges. We did not try other environments, as training both the baseline models and our model took too long.

It is worth noting that we do not include the original PPO algorithm in our training curves, as it consistently fails to demonstrate stable progress on these tasks. This observation aligns with the findings presented in the HumanoidBench paper, where PPO's performance was notably suboptimal, particularly on high-dimensional environments like HumanoidBench.

4.1 Sit-Hard Environment

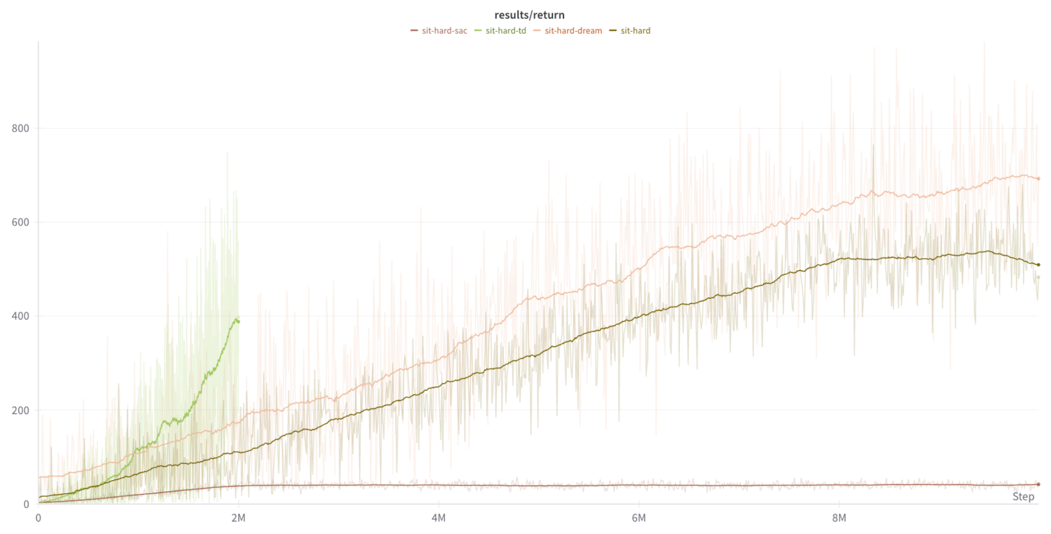

Figure 3: Reward Curves on Sit-Hard

In the Sit-Hard environment, our improved PPO model outperforms the model-free Soft Actor-Critic (SAC) algorithm by a significant margin, with SAC failing to maintain a stable standing or sitting posture. The performance of our algorithm reaches a loss level comparable to that of model-based methods such as TD-MPC2 and DreamerV3. This suggests that, despite being model-free, our PPO variant can achieve near-model-based performance in this environment. Both our model and

TD-MPC2 are occasionally able to sit on the unfixed chair, although they usually tend to squat near it due to the risk of knocking it over. In contrast, DreamerV3 demonstrates a slightly more stable sitting posture in this environment.

4.2 Bookshelf-Simple Environment

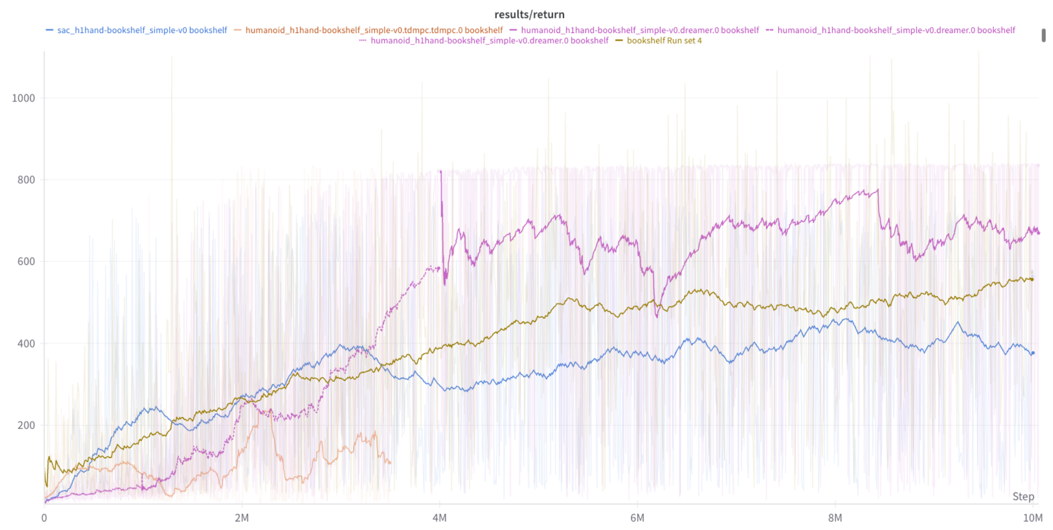

Figure 4: Reward Curves on BookShelf_Simple

In the Bookshelf-Simple environment, our model, SAC, and DreamerV3 exhibit similar behavior—they all stay close to the bookshelf, although none can complete the manipulation task. We attribute this phenomenon to the inadequate reward design of the HumanoidBench environment, which is too imprecise to effectively guide the robot. In contrast, TD-MPC2 behaves differently, deviating from the bookshelf and straying from the intended task. This highlights that our PPO model, SAC, and DreamerV3 maintain more stable and goal-directed behavior, while TD-MPC2 struggles with consistency in this environment.

4.3 Truck Environment

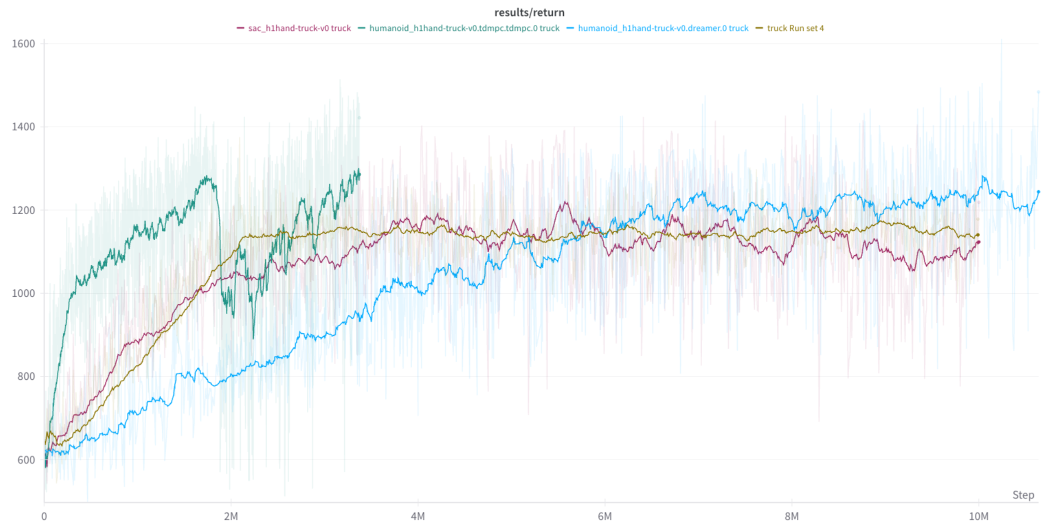

Figure 5: Reward Curves on Truck

The truck environment presents a case where reward design has a significant impact on learning performance. In this setting, all models—including ours—fail to make meaningful progress, instead remaining stationary, lying on the ground, or rolling and trembling. The poorly designed rewards lead to a plateau in performance for all algorithms, emphasizing the crucial role of well-tuned reward structures in reinforcement learning tasks.

4.4 Walk Environment

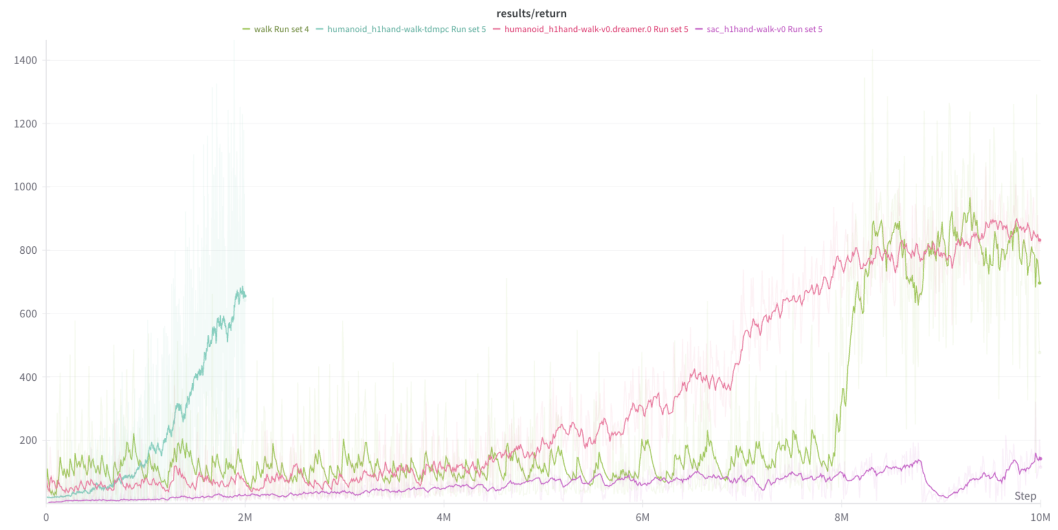

Figure 6: Reward Curves on Walk

In the Walk environment, SAC struggles to learn a valid walking policy and fails to make meaningful progress, often unable to stand firmly. In contrast, both the model-based methods, including TD-MPC2 and DreamerV3, and our improved PPO algorithm successfully learn to walk. Initially, our PPO model exhibits more oscillatory behavior, which can be attributed to the relatively large action entropy term we incorporate to encourage exploration during the early stages of training. While this introduces some instability at the start, our PPO model quickly converges to a successful walking policy after sufficient exploration.

Overall, the experiments demonstrate that our method significantly improves PPO's performance in high-dimensional, complex humanoid robot learning tasks. We believe that, fundamentally, both model-based and model-free models rely on sampled data for learning. By improving the representation learning of model-free models and enhancing their ability to understand the data, their data efficiency can be effectively increased. In the original paper of HumanoidBench, the performance of model-free methods was quite poor, which may be due to the fact that the 10 million samples used during training are too few for model-free methods that haven't been specially optimized.

5 Conclusion

In this paper, we present significant improvements in the data efficiency and performance of the PPO algorithm on the HumanoidBench benchmark, which involves a humanoid robot tackling complex tasks requiring both locomotion and manipulation. By enhancing the representation learning capability of PPO through sequentially modeling the policy-value network with interleaved states and actions, and incorporating techniques such as state normalization, reward scaling, and Generalized Advantage Estimation, we address the limitations highlighted in the original HumanoidBench paper. Our enhanced PPO algorithm not only surpasses traditional model-free methods, including the baseline PPO and Soft Actor-Critic algorithms, but also achieves performance levels similar to state-of-the-art model-based approaches such as DreamerV3 and TD-MPC2. This work offers valuable insights into improving the sample efficiency and representation learning capabilities of PPO, which are critical for deploying PPO in real-world, complex environments.

References

[1] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

[2] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[3] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[4] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[6] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[7] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[8] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[9] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

[10] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

[11] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022.

[12] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[13] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[15] Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoid-bench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *arXiv preprint arXiv:2403.10506*, 2024.

[16] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[17] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[18] Aohan Zeng, Zhengxiao Du, Mingdao Liu, Kedong Wang, Shengmin Jiang, Lei Zhao, Yuxiao Dong, and Jie Tang. Glm-4-voice: Towards intelligent and human-like end-to-end spoken chatbot, 2024.