Enhancing Game Control Through Hybrid Reinforcement Learning

Danhua Yan [1]

Abstract

This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase. This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase. This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase. This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase. This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. 150 words.

1. Introduction

Training reinforcement learning (RL) agents typically demands substantial data and exploration to achieve optimal performance. In complex environments such as difficult retro arcade games, high-dimensional state spaces, sparse rewards, and intricate dynamics make pure exploration particularly inefficient. Moreover, situations with limited or costly exploration opportunities increase the risk of converging to suboptimal policies.

To address these challenges, hybrid RL (HRL) paradigms leveraging demonstrations and imitation learning (offline RL) combined with agent exploration (online RL) have emerged as effective solutions. Specifically, Behavior Cloning (BC) integrated with Proximal Policy Optimization (PPO) has shown considerable potential. These hybrid methods use human demonstrations to bootstrap initial policies, significantly accelerating learning, with the potential to converge to policies that surpass demonstration performance levels.

This study investigates approaches that integrate human-generated demonstrations into PPO training, focusing on techniques such as transfer learning from BC, KL divergence-based regularization, and assisted exploration strategies in a complex retro game environment.

2. Related Work

Bootstrapping RL agents from demonstrations or imitation learning has shown significant promise. Hybrid paradigms combining human guidance (offline RL) and agent exploration (online RL) can accelerate policy learning and achieve performance beyond demonstrations (Hester et al., 2017; Nair et al., 2018; Song et al., 2023; Ren et al., 2024; Coletti et al., 2023).

Behavior cloning (BC) effectively initializes reinforcement learning policies using human demonstrations to tackle challenging exploration problems. It often trains faster than RL agents, providing a solid starting point for explorations.

PPO weight initialization via transfer learning is a common strategy. A BC policy network is first trained through supervised learning on human demonstration data. These pre-trained weights are then used to initialize PPO training. The DQfD approach by (Hester et al., 2017) combines behavior cloning and DQN networks, demonstrating effective training and surpassing demonstration performance. (Laidlaw et al., 2024) shows BC weights initiated PPO using Atari-HEAD human datasets, indicating notable efficiency gains, especially for tasks with extended exploration phases and sparse rewards. (Coletti et al., 2023) demonstrates that PPO warm-started with expert demonstrations can achieve satisfactory results in the costly exploration field of controlling a fixed-wing unmanned aerial vehicle. This approach significantly reduces the initial exploratory behavior required by the agent, leading to substantial improvements in sample efficiency and faster convergence.

[1]Department of Computer Science, Stanford University. Correspondence to: Danhua Yan <dhyan@stanford.edu>.

CS234: Reinforcement Learning Winter 2025 - Final Project

Another common approach is to integrate behavior cloning into PPO by adding a divergence-based regularization term, typically the Kullback-Leibler (KL) divergence, between the current PPO policy and a reference BC policy. This regularization guides the PPO agent to remain close to behaviors suggested by human demonstrations, especially early in training. This method is similar to the Kickstarting framework by (Schmitt et al., 2018), where an auxiliary distillation loss encourages the student policy to follow the teacher (reference) policy initially, progressively allowing divergence as training advances to surpass human performance.

Assisted exploration leverages demonstrations to guide agent exploration explicitly, often through curriculum strategies that reset or start episodes from states sampled from human demonstrations. This approach does not require a BC network but uses human trajectories to shape the exploration process, enhancing learning efficiency and performance. The idea is to reset the agent to progressively more challenging starting points, encouraging the learning process (Florensa et al., 2018). (Salimans & Chen, 2018) illustrate this in Atari's Montezuma's Revenge, using demonstration states to reduce exploration complexity and improve learning outcomes. This form of assisted exploration allows the agent to efficiently encounter and practice critical, sparse-reward regions of the state space, enhancing PPO's capability to master challenging tasks that conventional random exploration methods struggle with.

In this study, we explore the efficacy of HRL paradigms in a retro game environment.

3. Data and Environement

In this project, we leverage the `stable-retro` library to create an OpenAI Gym environment for training an agent to play the NES game Super Mario Bros, level 1-1. The default integration of the environment encapsulates the game into the in-game visual frame as a matrix $I \in \mathbb{R}^{H\times W\times 3}$, where each element takes an integer value between 0 and 255, representing the RGB channels of the frame. The action of pressing 9 buttons on NES controllers is represented as a vector $\mathbf{a} = (a_1, a_2, \ldots, a_9) \in \{0,1\}^9$, where each button can be toggled independently, resulting in a total of 512 discrete action spaces. The default reward is the change in the x-axis position Δx moved by Mario. In-game metadata, including scores, time left, and positions of Mario, can also be retrieved for each timestep t.

3.1. Human Demonstration Data

To record human demonstrations, we implemented scripts to save gameplay interactions with the environment via a game controller. We recorded a single gameplay by amateur players $\{\tau_{\text{HD}}\}$, each successfully completing Level 1-1 without losing a life. The episode is saved as $\tau_{\text{HD}} = \{(s_t, a_t, r_t, d_t, m_t)\}_{t=0}^{T}$, where each element represents the observation, action, reward, termination boolean, and metadata. Additionally, a single trajectory of game emulator states is saved every 50 steps, $\tau_{\text{ES}} = \{s_t \mid t \in \{50k \mid k \in \mathbb{N}, 50k \leq T\}\}$, used for resetting RL agents to start from a state along the human demonstrated trajectory.

3.2. Customized Game Environement

To frame the game as a solvable RL problem within a reasonable time, we made the following custom modifications to the default game integration:

Action Space The default 512 discrete action space includes all possible joystick button combinations, most of which are not meaningful for controlling Mario. We reduced the action space to 3 commonly used button combinations (see Appendix A.1).

Termination States The default game termination occurs when Mario exhausts all lives or the 400 second time limit for Level 1-1 is reached. We employ stricter termination conditions: 1) Mario has only one life, and the game terminates immediately if he loses it; 2) If Mario remains stuck at the same position without moving right for 10 seconds, the game is terminated.

Reward Function The game's scoring system provides sparse rewards for defeating enemies, collecting coins or power-ups, and successfully completing the level. We modify the reward function to provide dense rewards, incorporating scores, encouraging rightward movement with milestones, and penalizing time consumption and termination without success:

$$\begin{aligned}\mathcal{R} =& \Delta s + \beta_x \Delta x + \beta_t \Delta t + \mathbf{1}[d_{\text{milestones}} = 1]M \\ &+ \mathbf{1}[d_{\text{timeout}} = 1]T_t + \mathbf{1}[d_{\text{death}} = 1]T_d\end{aligned}$$

where Δs is the score earned since the last state, Δx is the movement, Δt is the time spent, β are coefficients, M is the milestone score at 10%, 20%, etc., of the level, and T_t and T_d are penalties for timeout and death terminations.

Sampling Rate To ensure smooth rendering, the game runs at 60 fps. However, consecutive frames exhibit minimal differences. Following (Feng et al., 2024), we reduced the sampling rate to 15 fps to reduce state spaces.

4. Approach

In this section, we describe baseline and three different hybrid reinforcement learning approaches in controlling Mario for completing level 1-1.

4.1. Policy Training Architectures

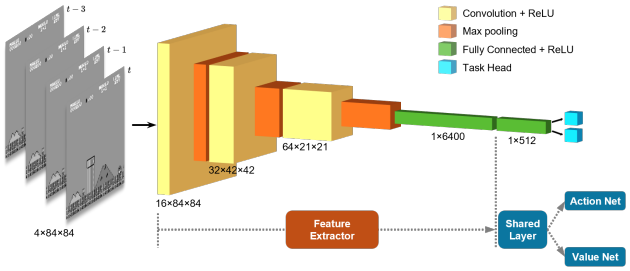

Figure 1. PPO agent architecture for playing Super Mario Bros. Four downsampled gameplay frames are stacked to represent the current state s_t, as input to the CNN actor-critic PPO networks.

State Representation Due to the presence of moving enemies (*e.g.*, Goombas, Koopa Troopas) and power-ups, *Super Mario Bros.* exhibits non-Markovian dynamics. As a result, incorporating temporal structure in the state representation is crucial for accurately estimating enemy movements and other dynamic changes. To capture these temporal dependencies, we define the state s_t at time t as a stack of four consecutive gameplay frames, $\{f_{t-3}, \ldots, f_t\}$, preserving local temporal dynamics (see Figure 1). Each frame is downsampled from its original RGB representation to a single-channel grayscale image and resized to 84×84 pixels.

Network Architectures All baselines and HRL approaches use the same architecture (Figure 1). The feature extractor is a CNN $\phi_s = \mathcal{F}(s;\phi)$ that maps the stacked state representation $4 \times 84 \times 84$ to a dense feature representation. The CNN has three convolutional layers with 16, 32, and 64 channels, respectively. Each convolution uses a 3×3 kernel with stride 1 and padding 1, followed by ReLU activation and 2×2 max pooling, halving the spatial dimensions. The PPO actor-critic policy is a multi-head MLP network $a, V = \mathcal{M}(\phi_s;\theta)$ with a shared fully connected hidden layer of 512 units and ReLU activation, and two linear heads: the action net $\pi_\theta(a|s) = M_a(\phi_s;\theta_a)$, and the value net $\hat{V}(s) = M_V(\phi_s;\theta_V)$. For BC, it matches the PPO with just action net: $\pi_{\text{BC}}(a|s) = \mathcal{M}_{\text{BC}}(\mathcal{F}(s;\phi_{\text{BC}});\theta_{\text{BC}})$.

4.2. Baselines

Here we shall establish performance of offline-only and online-only RL approaches as baselines to compare against future HRL approaches and human demonstration trajectories.

4.2.1. Behavior Cloning (BC)

BC is an offline approach using supervised learning to map state-action pairs from human demonstrations. We learned a BC policy $\pi_{\text{BC}}(a|s) = \mathcal{M}_{\text{BC}}(\mathcal{F}(s;\phi_{\text{BC}});\theta_{\text{BC}})$ using (s,a) pairs from human demonstration data τ_{HD} as mentioned in section 3.1.

4.2.2. Proximal Policy Optimization (PPO)

PPO serves as an online baseline. We extended the `stable-baselines3` PPO implementation, utilizing the CNN feature extractor $\mathcal{F}(s;\phi)$ and the two-head MLP policy network $a, V = \mathcal{M}(\phi_s;\theta)$. Parameters ϕ and θ are updated via gradient descent per rollout iteration. Here we denote the policy learned by PPO as $\pi_\theta(a|s) = M_a(\mathcal{F}(s;\phi);\theta_a)$.

4.3. Hybrid Reinforcement Learning (HRL)

We explore three HRL paradigms, leveraging pre-trained BC policies or directly using human demonstration data in PPO policy learning.

4.3.1. BC-Initialized PPO

This approach initializes PPO explorations using pre-trained behavior cloning weights. These weights provide a biased prior for state and action distributions, giving the agent a reasonable initial policy for exploration. Instead of initializing the model with random parameters ϕ_0 and θ_0, i.e., $a, V = \mathcal{M}_0(\mathcal{F}_0(s;\phi_0);\theta_0)$, we warm-start the feature extractor $\phi_0 \leftarrow \phi_{\text{BC}}$ and/or the policy network $\theta_0 \leftarrow \theta_{\text{BC}}$ in this approach.

4.3.2. BC-Constrained PPO

This approach constrains the PPO policy to remain close to the pre-trained BC policy by adding a KL-divergence term to the loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{PPO}}(\theta) + \lambda \mathbb{E}_{s\sim\mathcal{D}}\left[\sum_a \pi_\theta(a|s) \log \frac{\pi_\theta(a|s)}{\pi_{\text{BC}}(a|s)}\right],$$

where $\mathcal{L}_{\text{PPO}}$ is the default PPO loss, and λ is a hyperparameter controlling the strength of the divergence loss. This parameter is a function of training steps, allowing the penalty to be relaxed in later stages of training to encourage surpassing the BC policy.

4.3.3. Assisted Explorations

This approach aims to shape the exploration process using human trajectories. The key idea is to reset the rollout to progressively more challenging starting points for the agent, encouraging the learning process (Florensa et al., 2018). (Salimans & Chen, 2018) proposed reversing the human gameplay trajectory as resets to effectively encourage learning. Inspired by their approach, we propose a simpler version of resets using an exponential decay schedule,

instead of running indefinitely until each reset reaches a certain performance threshold.

The exponential decay schedule for PPO resets works as follows: given a target training iteration count N and k resets along the trajectory, we want the i-th state to be reset for rollout $n(i)$ times, such that $n(i)$ follows a discrete exponential distribution $n(i) \sim r^i$, and $\sum_{i=1}^{k} n(i) \approx N$. Here, r is the decay factor, where $0 < r < 1$, that smaller r decays faster. The PPO rollouts start from the k-th human state from τ_{ES} (see section 3.1) $n(k)$ times, then move to the $(k-1)$-th state $n(k-1)$ times, and so on, until exhausting all states in τ_{ES}. If there are still training steps remaining, the rollouts start from the initial state s_0.

The intuition behind this approach is to distribute resets strategically within a given training iteration count. States closer to the winning state (later states) are easier and thus get less practice than the earlier, more difficult states. This schedule effectively distributes learning along good state trajectories within limited training time, aiding efficient explorations.

5. Experiments

In this section, we present experimental results for the aforementioned approaches.

5.1. Experimental Details

All experiments are trained on a single Nvidia GeForce RTX 4070 Ti SUPER 16GB GPU. Random seed is set to 12345 for consistency.

Reward Function We set coefficients of reward function described in section 3.2 as time coefficient $\beta_t = -1$, position coefficient $\beta_x = 0.1$, $M = 1000m$, where $m = \{0.1, 0.2, \cdots, 1\}$ is the game completion percentage. Timeout termination $T_t = -1000$, death termination $T_d = -1000$.

Behavior Cloning All (s_t, a_t) pairs from human demonstration τ_{HD} are split into `train` and `dev` sets in a 7:3 ratio. With a batch size of 32 and cross-entropy loss, the full network $\pi(\mathcal{F}(s;\phi);\theta)$ is trained for 500 epochs using AdamW optimizer with a learning rate of 10^{-4}. Early termination occurs after 50 epochs based on `dev` data accuracy.

PPO and HRL Extensions In each iteration, the agent rolls out for 512 steps and updates the networks for 10 epochs with a batch size of 32, over 200 iterations. We set an entropy loss coefficient of 0.01. For BC-constrained PPO, with λ for KL-divergence loss regularization starting at 0.1 and linearly decreasing to 0 by the end of training. For baseline PPO, BC-constrained PPO, and assisted exploration methods, we use a learning rate of 10^{-4}, a clip range of 0.1. For BC-initialized PPO (or any extensions involving loading pre-trained BC weights), we use a smaller learning rate of 10^{-5} and a linear clip range schedule starting from 0.05 to 0.15. This stabilizes PPO near the BC policy initially and later encourages policy updates to potentially surpass demonstration strategies. For assisted explorations, we have $k = 43$ resets along demonstration τ_{ES}, with $r = 0.9$ and $N = 100$. Other hyperparameter are kept the same as `stable-baselines3` default values.

5.2. Evaluations

Each policy is evaluated as a stochastic policy, where the action at state s_t is sampled from a categorical distribution based on predicted logits. Each policy plays level 1-1 fifty times, starting with the `RIGHT` action to prevent stalling. We measure cumulative rewards and game completion percentage, defined as the termination x position divided by the full canvas length. Both the mean and distribution of rewards and completion percentages are analyzed to compare the performance of each policy. Given that all PPO-based policies are trained for a fixed 200 iterations, the performance of each policy can serve as a proxy for evaluating the efficacy of each method.

5.3. Baselines

We compare the performance of two baseline policies over 50 gameplays using mean cumulative rewards (μ_R) and mean game completion percentage (μ_C). Table 1 summarizes the gameplay statistics. Two-sample t-tests show that for μ_R, $p = 0.07$, and for μ_C, $p = 0.04$. These results suggest statistical significance at the 10% level, indicating that BC performs better than PPO, despite BC's training time being only 0.3% of PPO's. This highlights the inefficiency of PPO in complex game environments and the effectiveness of BC, even when trained on a single human demonstration trajectory only. However, neither policy completes the game. Observations reveal common failures in avoiding enemies (Goombas), indicating that the current state representation and model architecture may lack the complexity needed to predict enemy movements and avoid them effectively.

Experiment	μ_R	σ_R	μ_C	σ_C	Train Time (s)
BC	-648.3	368.5	22.4%	13.6%	14
PPO	-757.3	197.8	18.0%	5.9%	4,464

Table 1. Performance of baseline policies.

5.4. BC-Initialized PPO (BCI)

For this paradigm, we experimented with three different policies: `MLP` initializes the PPO policy with $\theta_0 \leftarrow \theta_{\text{BC}}$ for the policy MLP network $\mathcal{M}(\phi_s;\theta)$ while keeping the feature extractor $\mathcal{F}(s;\phi)$ randomly initialized; `FEAT` initializes the feature extractor with $\phi_0 \leftarrow \phi_{\text{BC}}$ while keeping the

MLP policy network randomly initialized; ALL transfers both $\phi_0 \leftarrow \phi_{\text{BC}}$ and $\theta_0 \leftarrow \theta_{\text{BC}}$.

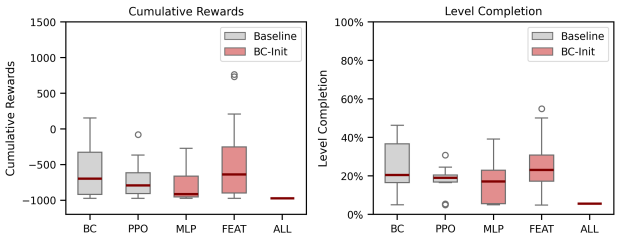

Figure 2. Performance of BC-Initialized policies vs. baseline policies.

Figure 2 shows game statistics for BCI policies compared with baselines. The FEAT policy slightly outperforms both baselines in terms of reward and completion percentage distributions. As seen in Figure 3, the FEAT policy's rewards are still trending upward, indicating potential for further improvement with extended training. Conversely, the MLP and ALL policies perform worse than the baselines, particularly ALL, where transferring all BC weights results in a nearly deterministic policy that gets stuck in suboptimal states.

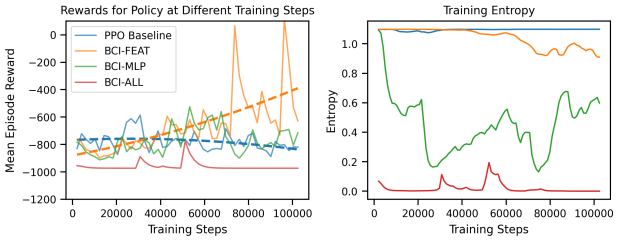

Figure 3. BC-Initialized policies rewards and entropy over time during training.

These behaviors correlate strongly with the entropy of the policy's actions. Figure 3 shows the entropy curve over time. The ALL policy starts with low entropy, severely limiting PPO exploration. This suggests the BC policy overfits to a single trajectory, making it sensitive to errors in the dynamic, stochastic game environment. Similarly, the MLP policy's strong action prior leads to lower entropy during training, and the mismatch with a randomized feature extractor worsens performance. In contrast, the FEAT policy shows a gradual decrease in entropy and an increase in rewards, indicating it learns better strategies with growing confidence, compared to the almost random actions of the PPO baseline. Feature weights ϕ_{BC} provides a state distribution prior while maintaining action randomness, allowing PPO to explore bootstrapped states. With a relaxed clip range near the end of training, the FEAT policy surpasses the BC policy.

References

Coletti, C. T., Williams, K. A., Lehman, H. C., Kakish, Z. M., Whitten, D., and Parish, J. Effectiveness of warm-start ppo for guidance with highly constrained nonlinear fixed-wing dynamics. *2023 American Control Conference (ACC)*, pp. 3288–3295, 2023. URL https://api.semanticscholar.org/CorpusID:259338376.

Feng, Y., Subramanian, S., Wang, H., and Guo, S. Reinforcement learning (ppo) with super mario bros, 2024. URL https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html. Accessed: 2025-03-15.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning, 2018. URL https://arxiv.org/abs/1707.05300.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z., and Gruslys, A. Deep Q-learning from Demonstrations, November 2017. URL http://arxiv.org/abs/1704.03732. arXiv:1704.03732 [cs].

Laidlaw, C., Russell, S., and Dragan, A. Bridging rl theory and practice with the effective horizon, 2024. URL https://arxiv.org/abs/2304.09853.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming Exploration in Reinforcement Learning with Demonstrations, February 2018. URL http://arxiv.org/abs/1709.10089. arXiv:1709.10089 [cs].

Ren, J., Swamy, G., Wu, Z. S., Bagnell, J. A., and Choudhury, S. Hybrid Inverse Reinforcement Learning, June 2024. URL http://arxiv.org/abs/2402.08848. arXiv:2402.08848 [cs].

Salimans, T. and Chen, R. Learning montezuma's revenge from a single demonstration, 2018. URL https://arxiv.org/abs/1812.03381.

Schmitt, S., Hudson, J. J., Zidek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Kuttler, H., Zisserman, A., Simonyan, K., and Eslami, S. M. A. Kickstarting deep reinforcement learning, 2018. URL https://arxiv.org/abs/1803.03835.

Song, Y., Zhou, Y., Sekhari, A., Bagnell, J. A., Krishnamurthy, A., and Sun, W. Hybrid RL: Using Both Offline and Online Data Can Make RL Efficient, March 2023. URL http://arxiv.org/abs/2210.06718. arXiv:2210.06718 [cs].

A. Appendix

A.1. Custom Environment

The default 512 discrete action space captures all possible joystick button combinations. However, most of these combinations are not meaningful for controlling Mario. From the human demonstration trajectories, we narrowed down the action space to 3 common used button combinations. Then the action vector is labeled as integers (0-9, following below orders) as discrete action space for the environment.

```
# List of meaningful button combinations used in gameplay
meaningful_actions = [
      [0, 0, 0, 0, 0, 0, 0, 0, 0],  # No action
      [0, 0, 0, 0, 0, 0, 0, 1, 0],  # Right
      [0, 0, 0, 0, 0, 0, 0, 1, 1],  # Right + A (Jump)
]
```