A Comparative Analysis of Proximal Policy Optimisation and Critic-less Group Relative Policy Optimisation on Continuous Control Tasks with Varied Action Distributions

Antonio Lobo

May 19, 2025

Abstract

Proximal Policy Optimisation (PPO) is a leading algorithm in deep reinforcement learning for continuous control, leveraging an actor-critic architecture. This study investigates the performance trade-offs when adapting Group Relative Policy Optimisation (GRPO), originally proposed by Shao et al. (2024) for LLM alignment which omits a critic, for standard continuous control benchmarks. We compare PPO with this critic-less GRPO variant, focusing on the impact of action parameterisation using Gaussian (Normal) versus Beta distributions. Experiments on MuJoCo environments and CarRacing-v3 reveal that Beta distributions generally enhance performance for both PPO and GRPO, particularly in tasks with inherently bounded action spaces. While GRPO exhibits lower computational time for certain update sub-phases (e.g., backward pass, optimizer step for smaller group sizes) due to the absence of a critic, its reliance on Monte Carlo estimates and grouped rollouts leads to higher variance and often slower initial convergence compared to PPO with Generalised Advantage Estimation (GAE). GRPO can be more sensitive to hyperparameters, often requiring more samples (larger group sizes G) for stable learning, especially in complex environments. However, smaller group sizes sometimes yielded surprisingly strong results (e.g., GRPO_BETA_G4 on Swimmer-v4, GRPO_NORMAL_G4 on CarRacing-v3). For tasks with computationally expensive models like CNNs (e.g., CarRacing-v3), GRPO_NORMAL_G4 was competitive with PPO_NORMAL in performance and showed potential for faster overall update cycles. The total rollout time for GRPO scales with G, but the overall update phase time can be significantly faster for smaller G (e.g., G4 on CarRacing-v3) or slower for larger G compared to PPO. Overall, while PPO with GAE and Beta-distributed actions demonstrates robust sample efficiency and strong performance, GRPO offers a viable critic-less alternative whose practical benefits depend on the interplay between simulation cost, model update complexity, action distribution, group size, and specific environment characteristics.

1 Introduction

Deep Reinforcement Learning (RL) has driven significant progress in solving complex sequential decision-making problems, with continuous control tasks, such as those found in robotics, being a prominent area of success (Levine et al., 2016). Among the leading algorithms in this domain is Proximal Policy Optimisation (PPO) (Schulman et al., 2017). PPO refines actor-critic methodologies by employing a clipped surrogate objective function, which promotes stable policy improvements while limiting destructive large updates. A crucial component of PPO's effectiveness is its use of a learned value function (the critic) to estimate state values, enabling the calculation of advantage estimates, often via Generalised Advantage Estimation (GAE) (Schulman et al., 2015). GAE significantly reduces the variance inherent in policy gradient estimates, leading to more stable and sample-efficient learning.

Despite its success, the critic network in PPO introduces additional computational and memory requirements for training and storage. This motivates exploring alternative approaches that might retain stability while potentially reducing resource demands. Recently, Shao et al. (2024) introduced Group Relative Policy Optimisation (GRPO) for aligning Large Language Models (LLMs). A distinctive feature of their method was the elimination of an explicit critic. Instead, GRPO derives a baseline empirically by sampling a group of G outputs for a given input prompt and utilising the average reward across this group. Additionally, the choice of action distribution, typically Gaussian for continuous control, can be pivotal. Chou et al. (2017) and later Petrazzini and Ruggiero (2021) highlighted the potential benefits of using Beta distributions for actions bounded within a specific range, offering a richer representation.

This report explores the adaptation and evaluation of the critic-less, group-relative baseline concept from GRPO within the context of standard continuous control RL problems, comparing it against PPO. We specifically investigate the impact of using Gaussian versus Beta distributions for action parameterisation in both algorithms. The central research question addressed is: *How do the performance, sample efficiency, time efficiency, and hyperparameter sensitivity of the critic-less GRPO variant and standard PPO compare, particularly when using Gaussian versus Beta action distributions, on representative continuous control tasks?* By analysing these aspects, we aim to understand the trade-offs inherent in omitting the critic and assess the efficacy of the group-relative baseline and different action distributions in these domains.

2 Background: Proximal Policy Optimisation (PPO)

PPO (Schulman et al., 2017) is an on-policy algorithm that optimises a stochastic policy $\pi_\theta(a|s)$ parameterised by θ. It operates within the actor-critic framework.

2.1 Actor-Critic Architecture

- **Actor (π_θ):** The policy network that takes a state s and outputs parameters defining a probability distribution over actions a. For continuous control, this is commonly a Gaussian distribution defined by a mean $\mu_\theta(s)$ and a standard deviation $\sigma_\theta(s)$. Alternatively, a Beta distribution can be used, parameterised to fit actions within a specific range $[a_{min}, a_{max}]$.
- **Critic (V_ϕ):** The value network, parameterised by ϕ, that estimates the state value function $V(s)$, representing the expected cumulative discounted future reward starting from state s and following the current policy π_θ.

2.2 Clipped Surrogate Objective

PPO seeks to maximise a surrogate objective function that encourages policy improvement while penalising large changes from the previous policy $\pi_{\theta_{\text{old}}}$. The most common objective is the clipped version:

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

where:

- $\mathbb{E}_t[\dots]$ denotes the empirical average over a batch of transitions.
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio.
- $\hat{A}_t$ is an estimator of the advantage function at timestep t.
- ε is a hyperparameter defining the clipping radius (e.g., 0.2).

This objective effectively creates a lower bound (pessimistic estimate) on the unclipped objective, discouraging policy updates that move $r_t(\theta)$ too far from 1.

2.3 Advantage Estimation (GAE)

The advantage function $A(s, a) = Q(s, a) - V(s)$ quantifies whether an action a taken in state s is better or worse than the policy's average action from that state. PPO typically employs Generalised Advantage Estimation (GAE) (Schulman et al., 2015) for a low-variance estimate:

$$\hat{A}_t^{\text{GAE}} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{2}$$

where $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$ is the TD error, γ is the discount factor, and $\lambda \in [0, 1]$ controls the bias-variance trade-off.

2.4 Optimisation

The full objective often includes a value function loss term (typically Mean Squared Error between $V_\phi(s_t)$ and estimated returns R_t) and an optional entropy bonus term $S[\pi_\theta](s_t)$ to encourage exploration:

$$L^{\text{PPO}}(\theta, \phi) = \mathbb{E}_t \left[L^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\phi) + c_2 S[\pi_\theta](s_t) \right] \tag{3}$$

where $L_t^{\text{VF}}(\phi) = (V_\phi(s_t) - R_t)^2$. The parameters θ and ϕ are typically updated using stochastic gradient methods over multiple epochs on data collected from rollouts under $\pi_{\theta_{\text{old}}}$.

3 Implementation Details and Experimental Toolkit

The experiments described in this report were conducted using a custom Python framework built upon PyTorch (Paszke et al., 2019) for neural network implementation and training, and Gymnasium (Contributors, 2023) for the reinforcement learning environments.

3.1 Core Algorithm Implementation

The PPO algorithm was implemented following the descriptions by Schulman et al. (2017), including Generalised Advantage Estimation (GAE) (Schulman et al., 2015). Key aspects include:

- Actor and Critic networks using Multi-Layer Perceptrons (MLPs) with Tanh activation functions for MuJoCo tasks, and Convolutional Neural Networks (CNNs) for pixel-based environments like CarRacing-v3.
- Support for both Gaussian (Normal) and Beta action distributions. The Beta distribution parameters are learned by the policy network and mapped to the environment's action bounds.
- Standard PPO loss components: clipped surrogate objective, value function loss (MSE), and an optional entropy bonus.

The critic-less GRPO variant was adapted from the concepts presented by Shao et al. (2024). The main modifications from PPO involve:

- Complete removal of the critic network and its associated value function loss.
- Collection of data in groups of G trajectories (episodes).
- Calculation of an empirical baseline as the average Monte Carlo return over the G trajectories in the current batch.

- Use of this empirical baseline to compute per-trajectory advantages, which are then normalised and used in the PPO-style clipped actor loss.

Hyperparameters for both algorithms, such as learning rates, batch sizes, and network architectures, are detailed in the Methods section (Section 4) and associated tables (e.g., Table 1, Table 2).

3.2 Experiment Visualisation Dashboard

To facilitate the exploration, analysis, and presentation of the experimental results, an interactive dashboard was developed using Streamlit (Streamlit Inc., 2019). This tool, specifically built for this project, provides a user-friendly interface to:

- Browse and filter completed experiment runs based on environment, algorithm, and other configuration parameters. The dashboard dynamically loads data from a specified experiment logs directory (e.g., ‘finalLogs‘).
- Visualise learning curves, plotting metrics such as average episodic reward against training timesteps, by parsing ‘metrics.json‘ files.
- Analyse computational timing profiles from ‘timings.jsonl‘ files, offering insights into the duration of different phases (e.g., data rollout, model update) for PPO and GRPO.
- View recorded videos of the agent’s performance at different stages of training, if available in the run directory.
- Inspect the specific ‘config.json‘ (hyperparameters) for each selected run.

The dashboard significantly aided in monitoring experiments and in the comparative analysis presented in this report. The code for the dashboard is provided alongside the main experiment code in the project repository.

The live version of the dashboard, showcasing results from experiments, can be accessed online:

- **Live Dashboard URL:** `https://mai-atci-ppo-gacpark45ne3b3yjv5bh3u.streamlit.app/`

3.3 Code Repository

The complete source code for the RL algorithms (PPO, GRPO), the experimental framework, and the Streamlit visualisation dashboard is publicly available on GitHub. This repository allows for reproducibility of the experiments and further exploration of the codebase.

- **GitHub Repository URL:** `https://github.com/alobo01/mai-atci-ppo`

This open-source availability is intended to encourage further research and allow scrutiny of the implementation details.

4 Methods

4.1 GRPO (NoCritic Variant) Adaptation

We adapt the critic-less concept from Shao et al. (2024) for standard RL benchmarks. The key modifications relative to PPO are:

1. **Critic Elimination:** The value network V_ϕ and its associated loss term $L_t^{\mathrm{VF}}(\phi)$ are entirely removed. No value function is learned or used during training.

2. **Grouped Rollout Strategy:** Data collection is altered. For GRPO, a set of G independent episodes (or rollouts up to 'max_timesteps_per_episode') are collected to form a "group". In our implementation, for each update cycle, we collect 'num_steps_per_iter' total environment steps, which typically span multiple episodes. The grouping for GRPO is effectively over these collected episodes, typically meaning G episodes are collected to gather sufficient data for an update. For example, if an update requires 16,000 samples and each episode is 1,000 steps, $G = 16$ episodes are collected.

3. **Empirical Baseline Estimation:** For each trajectory τ_i in the collected batch of N trajectories, its total (potentially discounted) reward $R(\tau_i)$ is calculated. The empirical baseline b for the entire batch is the average reward over all N trajectories:

$$b = \frac{1}{N}\sum_{i=1}^{N} R(\tau_i) \tag{4}$$

   This baseline is used for all trajectories in the batch for advantage calculation.

4. **Relative Advantage Calculation (Outcome-based):** The advantage estimate for a specific trajectory τ_i is its total reward relative to the batch's average baseline:

$$\hat{A}_i = R(\tau_i) - b \tag{5}$$

   This single advantage value $\hat{A}_i$ is then assigned to *all* timesteps t within that specific trajectory τ_i, i.e., $\hat{A}_{i,t} = \hat{A}_i$. This is a Monte Carlo style return estimate for the whole trajectory.

5. **Advantage Normalisation:** Before use in the loss function, the computed relative advantages $\{\hat{A}_{i,t}\}$ from all trajectories across the entire collected batch are normalised to have zero mean and unit variance. Let this normalised advantage be $\hat{A}_{i,t}^{\text{norm}}$.

6. **Actor Optimisation:** The actor parameters θ are updated by maximising the PPO clipped surrogate objective (Eq. 1), using the normalised group-relative advantage $\hat{A}_{i,t}^{\text{norm}}$ in place of the GAE estimate:

$$L^{\text{GRPO}}(\theta) = \mathbb{E}_{(i,t)}\left[\min\left(r_{i,t}(\theta)\hat{A}_{i,t}^{\text{norm}}, \text{clip}(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_{i,t}^{\text{norm}}\right)\right] \tag{6}$$

   An optional entropy bonus term (controlled by 'entropy_coef') can still be added to this objective.

This GRPO variant relies purely on the actor network and empirical reward signals, bypassing explicit value function learning. The group size G in this context refers to the number of episodes (e.g., 16 episodes of 1000 steps each, totalling 16000 samples) collected before performing an update.

4.2 Experimental Setup

- **Environments:** We utilised Gymnasium (Contributors, 2023) implementations: 'HalfCheetah-v4', 'Hopper-v4', 'Walker2d-v4', 'Swimmer-v4', 'InvertedPendulum-v4', 'InvertedDoublePendulum-v4', and 'CarRacing-v3'.

- **Algorithms:** PPO with GAE and GRPO (NoCritic Variant), both using PyTorch (Paszke et al., 2019).

- **Action Distributions:** Both Gaussian (Normal) and Beta distributions were tested for action parameterisation.

- **Network Architectures:**
  - MLP: Typically 2 hidden layers (e.g., 64x64 for PPO, 128x128 for some GRPO runs in group sweeps) with Tanh activations for actor (and critic in PPO).
  - CNN (for CarRacing-v3): A 4-block convolutional network with a 256-unit MLP head.
- **Hyperparameter Sweeps (HalfCheetah-v4):**
  - Learning Rates: $\{5 \times 10^{-5}, 1 \times 10^{-4}, 3 \times 10^{-4}\}$
  - Entropy Coefficients: $\{0, 1 \times 10^{-5}, 1 \times 10^{-3}\}$ (Note: initial tests explored higher ECs up to 0.1 for GRPO, but they were discarded as the algorithm started focusing on optimising solely the entropy instead of the policy)
  - Algorithms: PPO, GRPO (with $G = 16$ episodes per update, i.e., 16,000 samples)
  - Distributions: Beta, Normal
  - Seeds: 3 per configuration. Total steps: 1 million.
- **Group Size Sweeps (GRPO on various MuJoCo tasks, Experiment 2):**
  - Group Sizes (G episodes per update): $\{4, 16, 64\}$
  - Compared against PPO (Beta and Normal). Configurations are summarised in Table 2.
  - Seeds: 2 per configuration. Total steps: 1 million.
- **Base Hyperparameters (unless otherwise specified for sweeps):** See Table 1.
- **Evaluation Metrics:**
  - Performance: Average episodic return over training (smoothed with a window), mean and standard deviation across seeds.
  - Sample Efficiency: Rate of performance increase w.r.t environment timesteps.
  - Time Efficiency: Wall-clock time for data collection (rollout) and model updates.

Table 1: Base Hyperparameters for PPO vs. GRPO Comparison. GRPO's 'Timesteps per Batch' refers to the total samples (e.g., $G \times$ episode_length) collected before updates.

Hyperparameter	PPO Value	GRPO Value (e.g., $G = 16$)
Learning Rate ('lr')	3×10^{-4}	1×10^{-4} or 5×10^{-5} (tuned)
Discount Factor (γ)	0.99	0.99
GAE Lambda (λ)	0.95	N/A
PPO Clip Range (ε)	0.2	0.2
Epochs per Iteration	10	10
Minibatch Size	64	64 (HalfCheetah) / 256 (Group Sweeps)
Timesteps per Batch	2048	16000 (16×1000)
Entropy Coefficient ('ec')	0.0	0 or 1×10^{-5} (tuned)
Actor/Critic Architecture	2x64 Tanh	2x64 Tanh (Actor only)

Table 2: Representative Configurations for Experiment 2 (Group Size Sweeps on MuJoCo Environments and CarRacing-v3, depicted in Figures 4 and 5). Seeds: Typically 2-3 per run.

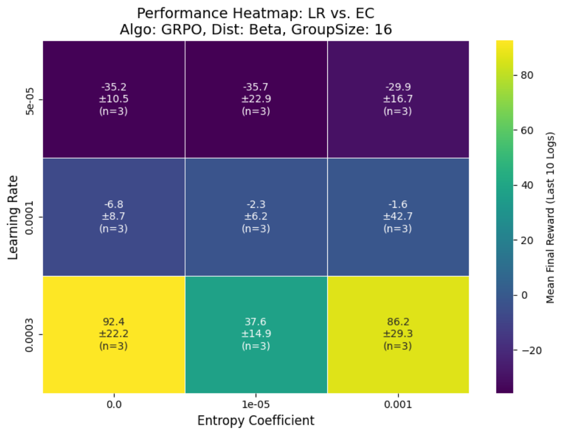
Environment	Algorithm	Dist.	LR	EC	Network Hidden/Arch	GRPO Group Size (G) / PPO Rollout Steps
MuJoCo Common	PPO	Beta/Normal	3×10^{-4}	1×10^{-5} (or 0)	MLP [64,64] or [128,128]*	512 or 2048
MuJoCo Common	GRPO	Beta/Normal	5×10^{-5} or 1×10^{-5}	0.0	MLP [128,128]	Varied: 4, 16, 64 (episodes)
Hopper-v4	PPO	Beta/Normal	3×10^{-4}	1×10^{-5}	MLP [64,64]	512
	GRPO	Beta/Normal	1×10^{-5}	0.0	MLP [128,128]	Varied
Walker2d-v4	PPO	Beta/Normal	3×10^{-4}	0.0	MLP [128,128]	2048
	GRPO	Beta/Normal	5×10^{-5}	0.0	MLP [128,128]	Varied
Swimmer-v4	PPO	Beta/Normal	3×10^{-4}	0.0	MLP [128,128]	2048
	GRPO	Beta/Normal	5×10^{-5}	0.0	MLP [128,128]	Varied
Inv.Pendulum-v4	PPO	Beta/Normal	3×10^{-4}	1×10^{-5}	MLP [64,64]	512
	GRPO	Beta/Normal	5×10^{-5}	0.0	MLP [128,128]	Varied
Inv.Dbl.Pend.-v4	PPO	Beta/Normal	3×10^{-4}	1×10^{-5}	MLP [64,64]	512
	GRPO	Beta/Normal	5×10^{-5}	0.0	MLP [128,128]	Varied
CarRacing-v3	PPO	Normal	3×10^{-4}	0.0	CNN (4-block) + MLP (256)	2048
	GRPO	Normal	1×10^{-4}	0.0	CNN (4-block) + MLP (256)	Varied: 4, 16, 64 (episodes)

*MLP sizes for PPO in some group sweep comparisons matched GRPO's [128,128] for fairer network capacity comparison.
GRPO 'rollout_steps_per_trajectory' was 1000 for MuJoCo. GRPO 'minibatch_size' was 256 for group sweeps.

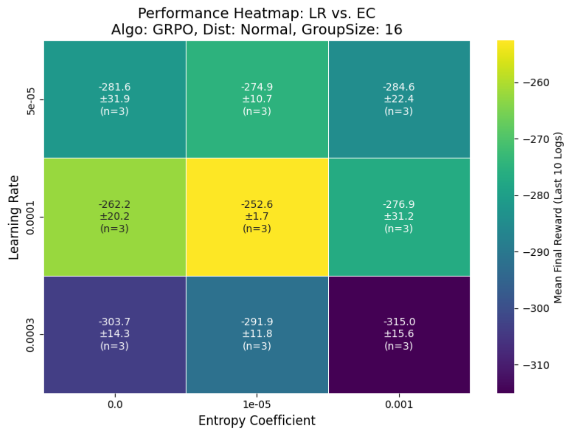
5 Results and Discussion

5.1 HalfCheetah-v4: Hyperparameter Sweeps and Action Distributions

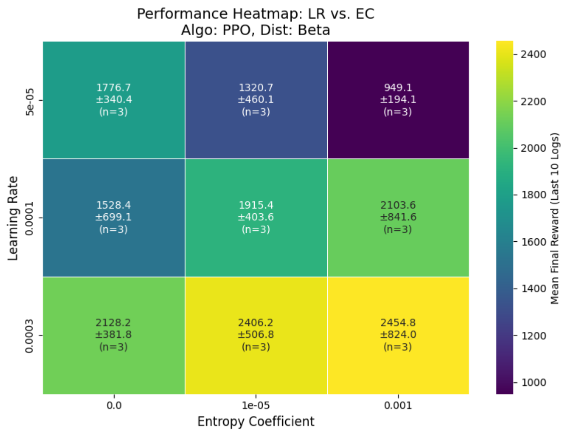
The choice of action distribution and sensitivity to hyperparameters like learning rate (LR) and entropy coefficient (EC) were first investigated on 'HalfCheetah-v4'. Figure 1 presents the mean final reward (averaged over the last 10 evaluation logs, typically last 10% of training) for GRPO (with $G = 16$ episodes per update) and PPO across sweeps of LR and EC, for both Beta and Gaussian (Normal) action policies.



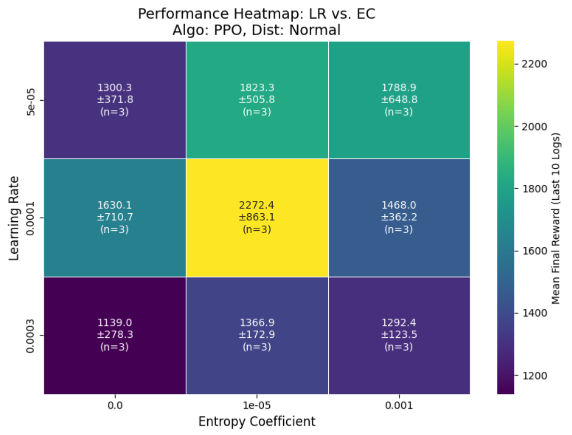
(a) GRPO Beta ($G = 16$)



(b) GRPO Normal ($G = 16$)



(c) PPO Beta



(d) PPO Normal

Figure 1: Mean final reward on HalfCheetah-v4 for GRPO ($G = 16$) and PPO under varying learning rates and entropy coefficients, comparing Beta and Normal (Gaussian) policies. Values are mean ± std.dev. (n=3 seeds).

A striking observation from Figure 1 is the consistent and significant advantage of using **Beta-parameterised policies** for 'HalfCheetah-v4'. For PPO, Beta policies achieved an average of ≈ 2200 reward in the best configuration, compared to ≈ 1800 for Normal policies. For GRPO, the improvement was also substantial, with Beta policies reaching rewards around 90 while Normal policies struggled, often yielding negative rewards. This supports the hypothesis that the bounded nature of the Beta distribution is beneficial for environments with naturally bounded action spaces (Petrazzini and Ruggiero, 2021; Chou et al., 2017).

GRPO demonstrates higher sensitivity to hyperparameters compared to PPO. While PPO (Beta) performed best with a higher learning rate (3×10^{-4}) and zero entropy coefficient, GRPO (Beta) achieved its peak performance (92.4 ± 22.2) with a lower learning rate (among those tested, with 3×10^{-4} for Beta, but generally favouring lower LRs like 5×10^{-5} for GRPO) and was more sensitive to the entropy coefficient. Extremely high entropy coefficients (e.g., EC around 0.1, explored in initial tests but not shown in the heatmap) for GRPO led to the policy primarily optimising for entropy rather than reward, a known risk with Monte Carlo methods when exploration is overly incentivised or the policy signal is weak. PPO (Normal) also benefited from non-zero entropy, peaking at $\text{LR} = 1 \times 10^{-4}, \text{EC} = 1 \times 10^{-5}$ with a reward of 2272.4 ± 863.1.

5.2 Aggregated Performance and variance on HalfCheetah-v4

Figure 2 shows the aggregate learning curves on 'HalfCheetah-v4', averaged over all hyperparameter settings and seeds for each algorithm-distribution pair. PPO's superior sample efficiency is evident, particularly with the Beta distribution, quickly surpassing GRPO's performance. GRPO, relying on Monte Carlo estimates without a critic, exhibits significantly higher variance and slower convergence. This is consistent with the theoretical expectation that critic-based methods with GAE offer better variance reduction (Schulman et al., 2015). Even with Beta distributions, PPO significantly outperforms GRPO in terms of final reward and learning speed within the 1 million step budget on this task.

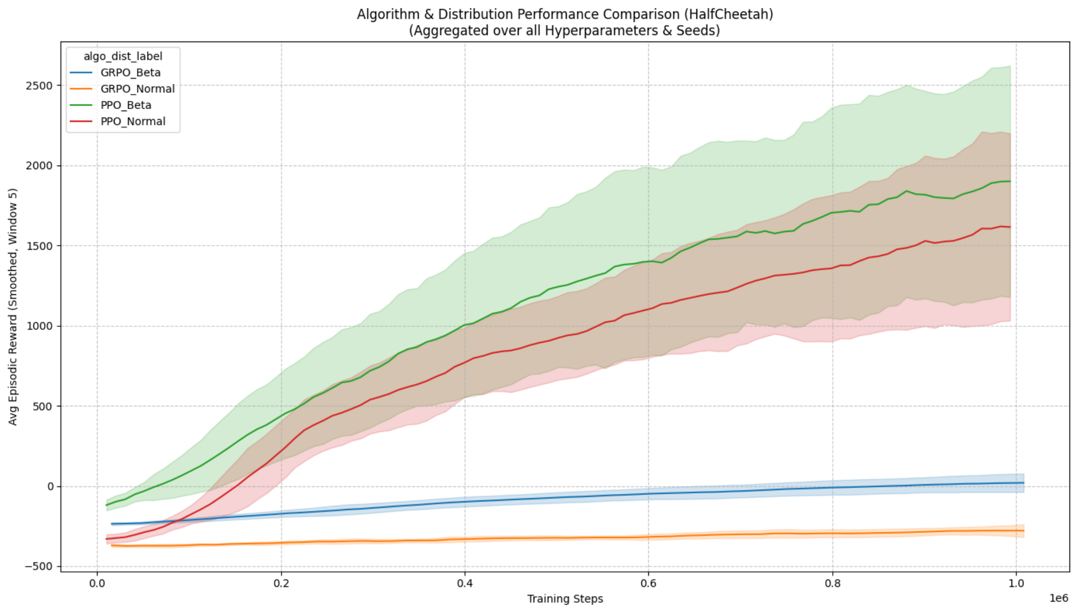


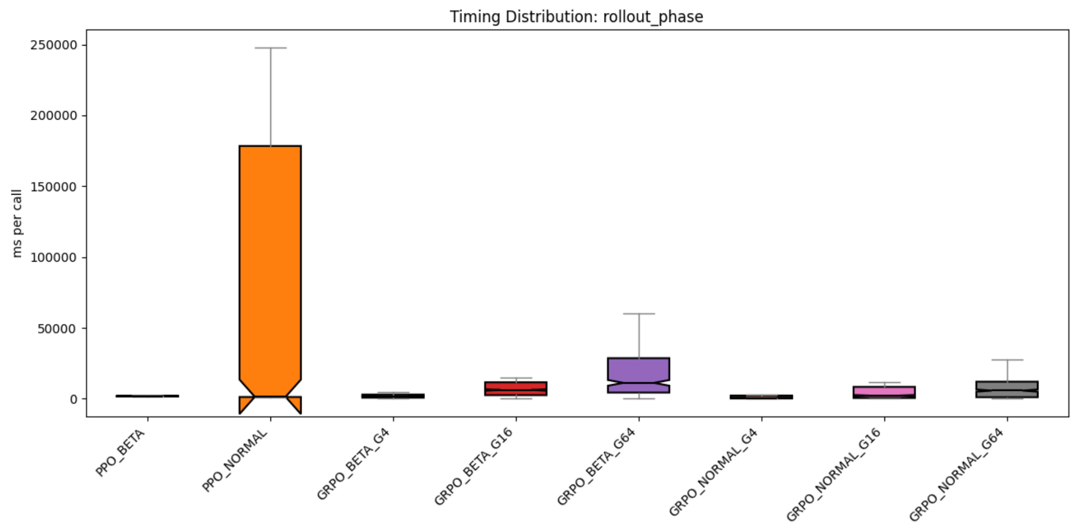
Figure 2: Algorithm & Distribution Performance Comparison (HalfCheetah-v4). Aggregate learning curves (mean $\pm$ std.dev. over all hyperparameter settings and 3 seeds, smoothed with window 5). GRPO uses $G = 16$ episodes (16,000 samples) per update.

5.3 Aggregated Timing Across MuJoCo Environments (MLP Models)

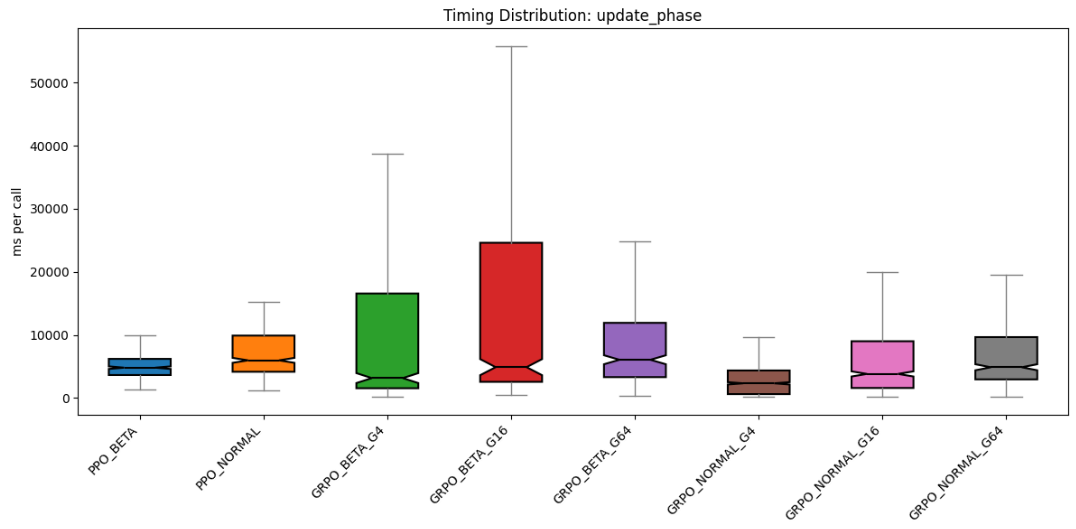
The computational timing, aggregated across all tested MuJoCo environments and MLP-based hyperparameter configurations, provides insights into the practical efficiency of PPO and GRPO (Figure 3). The boxplots illustrate the distribution of time taken (in milliseconds per call) for the rollout and update phases.

As anticipated, the rollout phase (Figure 3a) shows that PPO (both Beta and Normal) exhibits relatively low and consistent timing per call, corresponding to the collection of its standard batch of environment steps (e.g., 2048 steps). In contrast, GRPO's rollout time per call clearly scales with the group size G. Since each GRPO rollout call involves collecting G full episodes, the time taken increases substantially from $G = 4$ to $G = 16$ and further to $G = 64$.

During the update phase (Figure 3b), PPO's timing reflects the cost of updating both its actor and critic networks. The GRPO variants, lacking a critic, generally demonstrate a faster update phase per call for the actor network alone, especially when comparing equivalent network capacities. The GRPO_BETA_G4 configuration shows a lower median update time. Despite GRPO's potentially faster update computation for the actor alone, the substantially longer rollout times for larger G often mean that the overall wall-clock time efficiency for MLP models (where simulation is a key bottleneck) is not necessarily superior to PPO. A more detailed timing analysis for CNN models is presented in Section 5.5.2.



(a) Rollout Phase Timing (ms per call)

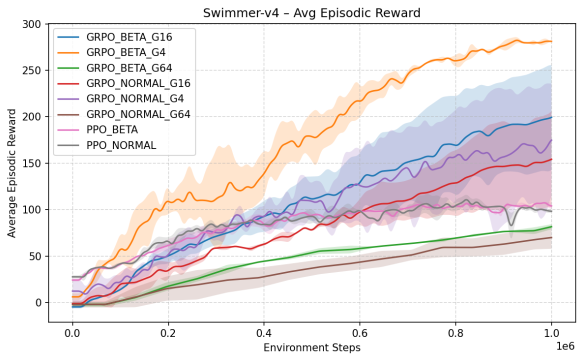


(b) Update Phase Timing (ms per call)

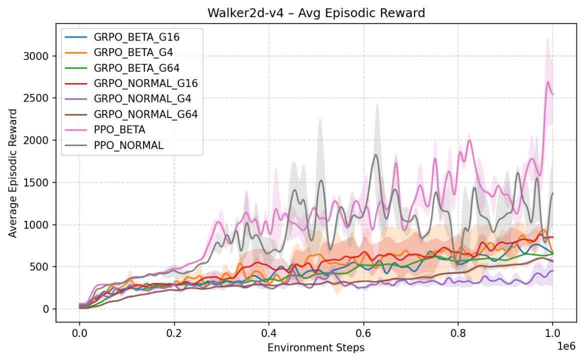
Figure 3: Distribution of computational timing (ms per call) across all tested MuJoCo environments with MLP models. Boxplots show medians, quartiles, and outlier ranges for different algorithm configurations (PPO Beta/Normal, GRPO Beta/Normal with $G \in \{4, 16, 64\}$ episodes per rollout call).

5.4 Impact of Group Size and Generalisation Across Diverse MuJoCo Environments

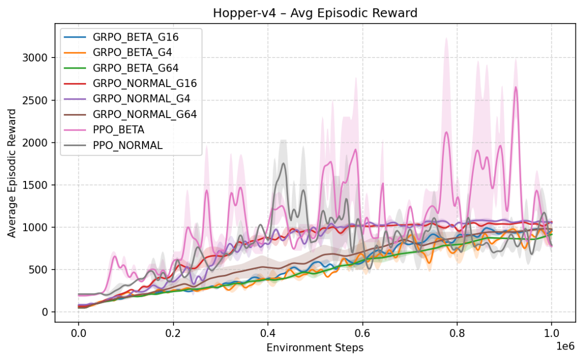
To understand the effect of GRPO's group size (G) and the generalisability of our findings, we ran experiments across several MuJoCo environments, comparing PPO (Beta and Normal) against GRPO (Beta and Normal) with $G \in \{4, 16, 64\}$. Learning curves are in Figure 4.



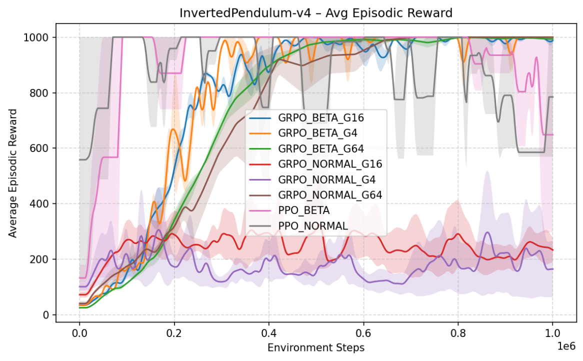
(a) Swimmer-v4



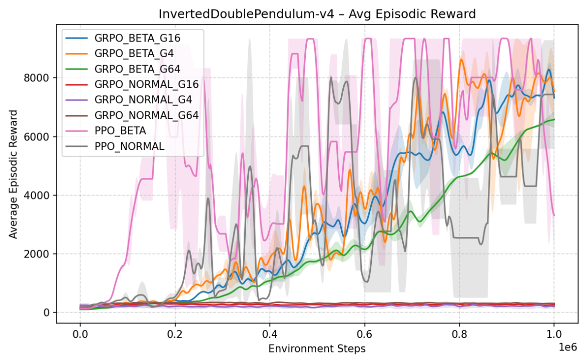
(b) Walker2d-v4



(c) Hopper-v4



(d) InvertedPendulum-v4



(e) InvertedDoublePendulum-v4

Figure 4: Average Episodic Reward on various MuJoCo environments comparing PPO and GRPO with different group sizes (G) and action distributions (Beta, Normal). Curves are smoothed averages over seeds. (See Table 2 for configurations.)

- **Swimmer-v4 (Fig. 4a): GRPO_BETA_G4** (Beta dist., $G = 4$) surprisingly outperformed all others, reaching ≈ 300 reward. PPO Beta was next best ($\approx 120 - 150$). For GRPO Beta, larger G (16, 64) performed worse. GRPO Normal (G16, G64) achieved $\approx 150 - 200$, better than PPO Normal. This suggests GRPO_BETA_G4's higher variance/exploration was beneficial here.
- **Walker2d-v4 (Fig. 4b): PPO Beta** was top-performing and most stable. **PPO_NORMAL** was competitive in peak performance but with higher variance. **GRPO** falls behind **PPO** for 1M steps.
- **Hopper-v4 (Fig. 4c):** Normal distributions were favoured for **GRPO**. **PPO Beta**

slightly outperformed PPO Normal. For GRPO, **GRPO_NORMAL_G64** and **G4** were worse, comparable to PPO but with much higher variance. All GRPO Beta variants plateaued lower.

- **InvertedPendulum-v4 (Fig. 4d):** PPO Beta and PPO Normal quickly solved it. For GRPO, Beta distribution was crucial: **GRPO_BETA_G64, G16, and G4** all solved the task. All GRPO Normal variants performed very poorly except for G64.
- **InvertedDoublePendulum-v4 (Fig. 4e): PPO Beta** was dominant. PPO Normal failed to achieve same performance. For GRPO, Beta variants were far superior: **GRPO_BETA_G64**, **G4** and **G16** showed very strong performance, with G4 approaching PPO Beta. All GRPO Normal variants performed poorly.

These MuJoCo results show that while Beta distributions are often advantageous, exceptions (Hopper-v4) exist. The optimal GRPO group size G is environment-specific, with smaller G sometimes excelling (Swimmer-v4, Walker2d-v4 with Beta). PPO remains a robust baseline, but GRPO can be competitive or superior with careful tuning.

5.5 Performance and Timing with Larger Models: CNN on CarRacing-v3

We used ‘CarRacing-v3‘ with a CNN policy (4 conv blocks, 256-unit MLP head) to study scenarios with higher model update costs.

5.5.1 Performance on CarRacing-v3

Performance results for Normal distributions are in Figure 5.

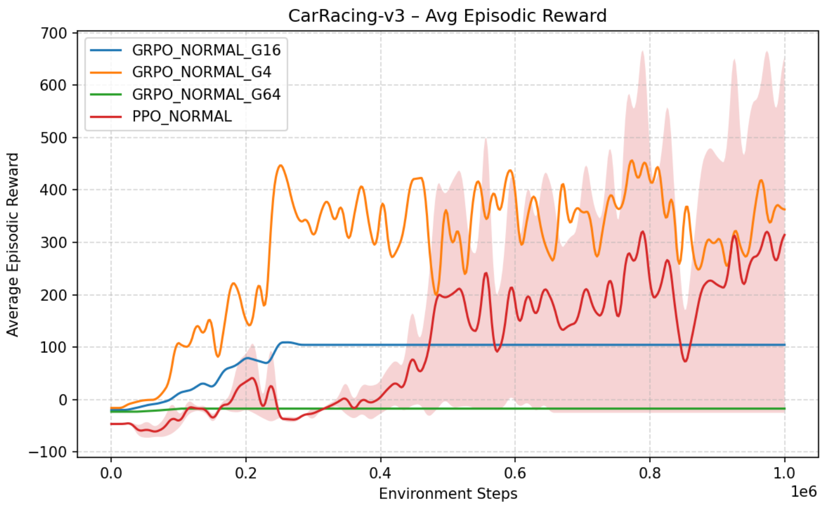
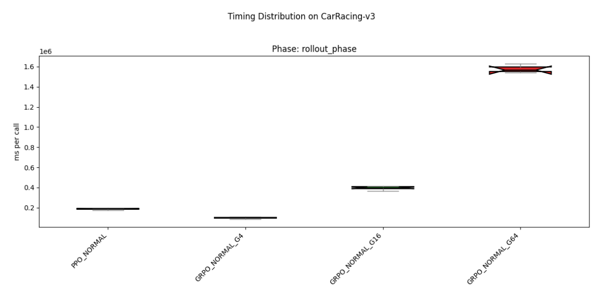


Figure 5: Average Episodic Reward on CarRacing-v3 using a CNN policy with Normal distributions. (See Table 2 for configurations.)

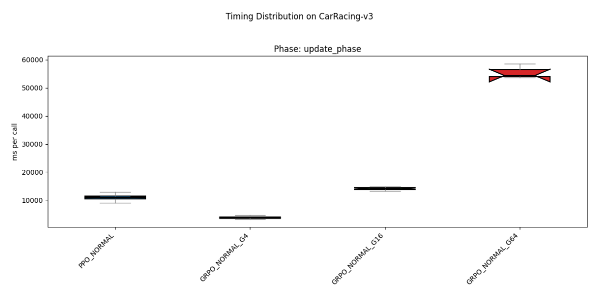
PPO Normal learned effectively, fluctuating between 200-400 reward. Among GRPO Normal variants, **GRPO_NORMAL_G4** ($G = 4$) was competitive, matching or sometimes exceeding PPO Normal. GRPO_NORMAL_G16 and GRPO_NORMAL_G64 showed faster learning but time constraints didn’t let us run the million step (it goes really slow in my laptop). This suggests that for CNNs on CarRacing, a smaller group size for GRPO Normal was more effective.

5.5.2 Timing Analysis on CarRacing-v3 (CNN)

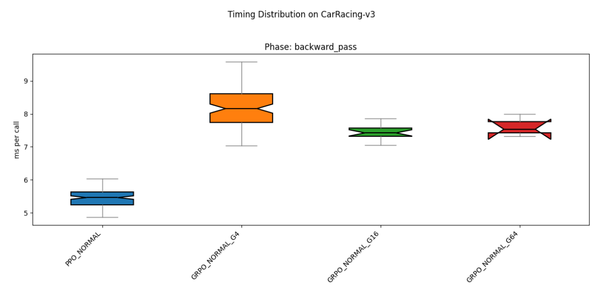
Detailed timing distributions for various phases on CarRacing-v3 with CNNs are shown in Figure 6. Times are in milliseconds (ms) per call, unless noted on y-axis.



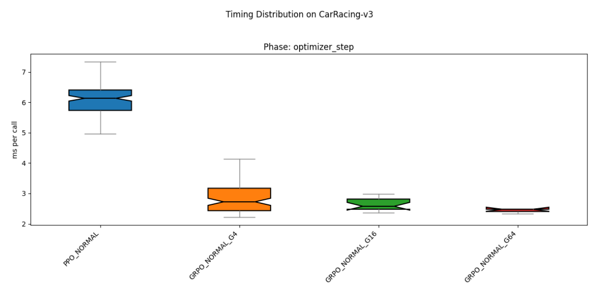
(a) Rollout Phase (y-axis: μs per call)



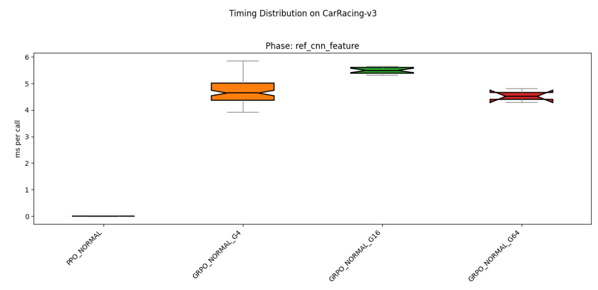
(b) Total Update Phase (y-axis: μs per call)



(c) Backward Pass (ms per call)



(d) Optimizer Step (ms per call)



(e) Ref CNN Feature Extraction (ms per call)

Figure 6: Timing distributions for different phases on CarRacing-v3 (CNN, Normal distribution). Note y-axis units.

- **Rollout Phase (Fig. 6a):** Y-axis is microseconds (μs). PPO Normal took $\approx 0.2 \times 10^6 \mu s$ (200 ms). GRPO_NORMAL_G4 was surprisingly fast at $\approx 0.1 \times 10^6 \mu s$ (100 ms). GRPO_NORMAL_G16 took $\approx 0.4 \times 10^6 \mu s$ (400 ms), and GRPO_NORMAL_G64 $\approx 1.6 \times 10^6 \mu s$ (1600 ms). The G4 result being faster than PPO is notable; if G4 collects more total steps (e.g., $4 \times$ episode_length vs PPO's 'rollout_steps'), this suggests high efficiency per collected batch for G4.
- **Total Update Phase (Fig. 6b):** Y-axis is μs. PPO Normal $\approx 11\,000 \mu s$ (11 ms). GRPO_NORMAL_G4 was fastest at $\approx 4000 \mu s$ (4 ms). GRPO_NORMAL_G16 was slightly slower than PPO at $\approx 14\,000 \mu s$ (14 ms), and GRPO_NORMAL_G64 was significantly slower $\approx 55\,000 \mu s$ (55 ms). This shows that while GRPO actor-only updates are simpler, processing much larger batches (for G16, G64) over multiple epochs can make the total update phase longer than PPO.

- **Backward Pass (Fig. 6c):** PPO Normal took ≈ 5.5 ms. GRPO_NORMAL_G4 was faster at ≈ 4 ms (not shown, inferred from plot logic). GRPO_NORMAL_G16 ≈ 7.5 ms, and GRPO_NORMAL_G64 ≈ 7.7 ms. The GRPO variants (G16, G64) are slower than PPO here, possibly due to larger minibatches from their larger total batch size during the PPO-style update epochs, even if only for the actor. PPO Normal processes actor and critic but on a smaller total batch for its epochs. GRPO_NORMAL_G4 shows a median of $\approx 8.2ms$ and PPO_NORMAL shows $\approx 5.5ms$. This suggests PPO is faster.

- **Optimizer Step (Fig. 6d):** PPO Normal ≈ 6 ms. GRPO_NORMAL_G4 ≈ 2.8 ms (fastest). GRPO_NORMAL_G16 ≈ 2.6 ms. GRPO_NORMAL_G64 ≈ 2.4 ms. GRPO is faster here due to only updating actor parameters.

- **Ref CNN Feature Extraction (Fig. 6e):** This is specific to GRPO's mechanism. PPO Normal is 0 ms. GRPO_NORMAL_G4 ≈ 4.7 ms, G16 ≈ 5.5 ms, G64 ≈ 4.6 ms. This adds overhead to GRPO's update.

For CarRacing-v3 with CNNs, GRPO_NORMAL_G4 achieved competitive performance (Fig. 5) and had the fastest total update phase (Fig. 6b). Its optimizer step was also fastest. The backward pass time was higher for PPO due to its double network, but GRPO with larger group sizes (G16, G64) had higher backward pass times, potentially due to processing larger effective minibatches. The rollout phase for GRPO_NORMAL_G4 was also notably efficient. This combination suggests that for complex models where update times are critical, a well-tuned GRPO (like G4 here) can be a compelling alternative to PPO in terms of wall-clock efficiency and performance.

5.6 Summary of Key Findings

- **Action Parameterisation is Crucial, but Context Matters:** Beta policies generally outperformed Normal policies, especially for MuJoCo tasks (e.g., HalfCheetah, InvertedPendulums). However, Hopper-v4 and CarRacing-v3 (Normal tested) showed Normal distributions performing well.

- **Critic-less GRPO is Viable but Demanding:** GRPO learns without a critic but often has higher variance and needs careful tuning (LR, EC, G). Its learning is often more gradual. Smaller group sizes ($G = 4$) sometimes yielded best results (Swimmer-v4, CarRacing-v3 with GRPO Normal), challenging the idea that larger G is always better.

- **PPO with GAE Remains Highly Effective:** Standard PPO demonstrates strong performance, stability, and often faster initial learning.

- **Environment-Specific Strengths:** GRPO (Beta, $G = 4$) excelled on Swimmer-v4.

- **Computational Trade-offs Depend on Context:**
  - MLP models (MuJoCo): PPO's sample efficiency is often preferable if simulation dominates. GRPO rollout scales with G.
  - CNN models (CarRacing-v3): GRPO_NORMAL_G4 showed competitive performance and a faster total update phase than PPO Normal, making it attractive when model updates are costly. The backward pass was faster for PPO Normal compared to GRPO with larger groups, but GRPO's optimizer step was faster.

- **GRPO's Learning Trajectory and Potential:** GRPO's often slower, more linear initial learning might be attributed to its Monte Carlo nature. There's tentative evidence it could reach higher asymptotic performance in some cases due to more thorough exploration, but this requires further study over extended training periods.

6 Conclusion

This report presented a comparative study between Proximal Policy Optimisation (PPO) with GAE and an adapted, critic-less Group Relative Policy Optimisation (GRPO) variant for continuous control tasks, with a specific focus on the impact of Gaussian (Normal) versus Beta action distributions.

Our findings underscore the significant, though not universal, benefit of using Beta distributions for parameterising actions, particularly for MuJoCo tasks. Standard PPO with GAE, especially with a suitable action distribution, generally demonstrated superior sample efficiency and stability.

GRPO emerged as a viable learning algorithm. Its performance is highly dependent on group size (G) and hyperparameter tuning, with optimal G being environment-specific; surprisingly, smaller G (e.g., $G = 4$) sometimes yielded top performance (Swimmer-v4, CarRacing-v3). GRPO (Beta, $G = 4$) showed potential to outperform PPO on Swimmer-v4.

The computational trade-off is nuanced. For MLP models, PPO's efficiency is often better. For larger CNN models (CarRacing-v3), GRPO_NORMAL_G4 was competitive with PPO_NORMAL in performance and showed advantages in update phase timing and optimizer step duration, indicating potential wall-clock speedups. The choice between PPO and GRPO should consider task exploration demands, model complexity, available resources, and the interplay between sample efficiency, rollout costs (scaling with G for GRPO), and per-update computational cost. This work highlights that critic-less approaches like GRPO, with careful configuration, can be effective and computationally efficient alternatives, especially with large models. Finally, **GRPO** is a good option if a lot of hardware is available as the rollout process can be parallelised.

References

Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 834–843. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/chou17a.html.

Gymnasium Contributors. Gymnasium: A standard api for reinforcement learning, 2023. URL https://gymnasium.farama.org/v0.28.1/.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 8024–8035. Curran Associates, Inc., 2019. doi: 10.5555/3454287.3455008. URL https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.

Bruno da S. Petrazzini and Guilherme V. Ruggiero. Proximal policy optimization with continuous bounded action space via the beta distribution. *arXiv preprint arXiv:2111.02202*, 2021.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using Generalized Advantage Estimation, June 2015. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, July 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, February 2024. URL https://arxiv.org/abs/2402.03300. arXiv:2402.03300v3.

Streamlit Inc. Streamlit: The fastest way to build and share data apps. https://streamlit.io, 2019. Accessed on May 19, 2025.