Universiteit Leiden
The Netherlands

DS & AI

Deepseek vs OpenAI

A comparison of GRPO and PPO in Reinforcement Learning environments

Alexander Cremer

Supervisor: Dr. Thomas Moerland
Second Reader: Koen Ponse

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

Abstract

This paper provides a systematic comparison between OpenAI's Proximal Policy Optimization (PPO) and Deepseek's Group-Relative Policy Optimization (GRPO) algorithms. The goal is to determine whether GRPO can serve as an alternative to the current state of the art, namely PPO. To investigate this, both algorithms were implemented and tested in two reinforcement learning environments. Their performance was compared across key metrics, such as stability, efficiency and overall results. Although, PPO generally performs marginally better than GRPO, GRPO yields competitive results. These findings highlight GRPO's strong potential, especially considering future work in hyperpapameter optimization and the fact that PPO is a well-tuned and studied algorithm, while GRPO may still need that attention.

Contents

1 Introduction

Deep Reinforcement Learning (DRL) combines two major sub-fields in Artificial Intelligence (AI), namely Deep Neural Networks (DNN) and Reinforcement Learning (RL)[iam25]. In DRL, an agent iteratively interacts with the environment, with the goal of maximizing cumulative rewards. DNNs are used to approximate the policy function, and potentially also the value function, while RL is utilized to efficiently update the models based on the agent's observations. DRL has a wide range of applications, as it is implemented in robots[ZQW20], gaming[LC17], healthcare [ZQW20]and various other fields[ADBB17].

In this paper, DRL agents are implemented in environments modeled as Markov Decision Processes (MDPs), where the future outcomes solely depend on the current state and action, therefore satisfying the Markov property. To train agents in such environments, the policy gradient method called Proximal Policy Optimization (PPO), a very popular method in DRL, was introduced by the team at OpenAI in 2017 [SWD+17]. PPO is an on-policy RL algorithm that improves upon previous policy gradient methods by using a clipped surrogate objective function to limit large updates to the policy, thereby ensuring more stable learning. Alternatively, DeepSeek's new developments in the field of DRL have introduced a new and more efficient method to train Large Language Models (LLMs), which is the Group Relative Policy Optimization (GRPO)[Tea24]. As described by DeepSeek, GRPO avoids the critic model needed in PPO and instead utilizes a group mechanism for policy updates [Tea24]. Effectively, this leads to faster training and more stability, as only an actor model is needed.

As RL continues to play a more important role in large language models (LLMs), Deepseek's adoption of GRPO for its training framework marks an important milestone in the field of AI [SWZ+24]. Deepseek-R1 achieved comparable results to OpenAI's o1 model. Remarkably, this was accomplished with lower computational cost, by leveraging the benefits that RL methods such as GRPO provide.

It introduced improvements over the o1 model, as it does not require any supervised fine-tuning (SFT), which relies on human-annotated data and introduces limitations, such as dependency on high-quality labeled data and overfitting. By directly applying RL without SFT, DeepSeek-R1 demonstrates that reasoning capabilities emerge through reinforcement learning signals alone, allowing for greater adaptability and improved long-term performance in reasoning tasks. Additionally, the DeepSeek-R1 model is computationally more efficient than the OpenAI-o1 model, as it only requires an actor model, as opposed to an actor and critic model, making it computationally more efficient. In conclusion, Deepseek's R1 model seems to perform similarly to OpenAI's o1 model with less resources and less computational power, highlighting the efficiency of a pure RL-based approach.

While GRPO has shown promising results in the context of LLMs, it is important to note that the structure of LLM tasks differs significantly from RL environments. In traditional RL, the agent acts within a well-defined simulation or physical environment. States and actions are clearly separated, with the environment providing a low-dimensional and structured state, and the agent selects from a discrete or continuous action space. Furthermore, each action leads to a state, modeled by a transition function, and is followed by an immediate reward, emphasizing the fact that RL environments provide frequent feedback. Usually, these environments are episodic, resetting after truncation or termination. In contrast, LLM tasks operate in a high-dimensional space where the state is the current sequence of tokens and actions are the predictions of the next token. States

and actions are intertwined as the action becomes part of the next state. Transitions are also dependent on the model rather than the environment. Moreover, rewards are sparse and typically apply to the entire predicted sequence of tokens, as opposed to singular actions [HXZW25]. These distinctions are important to make, as GRPO's strong performance in LLM tasks, more specifically in Deepseek-R1, do not imply a similar good performance in RL environments.
Since GRPO is a recent development in RL, there is little research conducted on this new model and its performance in RL environments. Therefore, this study aims to shed light on GRPO's potential broader applicability and given that PPO has been the industry standard so far, assessing GRPO's effectiveness could have significant implications for future reinforcement learning research and applications.

1.1 Research Aim

This research presents one of the first systematic comparisons of GRPO and PPO in general RL tasks. The goal is to thoroughly investigate their relative performance by answering the following questions:

Main question: How does GRPO compare to PPO in reinforcement learning environments?

Subquestion 1: How do the two methods perform in terms of their converged performance and their convergence speed?
Subquestion 2: Do the methods' performances change depeding on the reinforcement learning environmnent?
Subquestion 3: What effect does the group parameter G have on GRPO's performance?

1.2 Thesis Overview

This research is structured to provide a comprehensive comparison of GRPO with PPO. Section 2 discusses similar papers that this research is based upon. Section 3 presents detailed descriptions of PPO and GRPO, outlining their theoretical foundations. Section 4, describes the GRPO and PPO implementations, the RL environments used and the metrics selected. The results of these experiments are shown in Section 5, which is followed by a critical discussion in Section 6. Lastly, Section 7 summarizes the key findings and implications for future studies.

2 Related Work

On of the foundational algorithms to this thesis is PPO, introduced by OpenAI as an alternative to Trust Region Policy Optimization (TRPO) [SWD+17]. TRPO was designed to ensure safe policy updates by enforcing constraints on the divergence between consecutive policies [SLM+15]. It achieves this by using complex second-order optimization, which involves calculating matrices of second derivatives, called Hessians, making it computationally very complex and expensive. PPO addresses this issue by introducing a clipped surrogate objective function, which uses first-order optimization. As a results, PPO is a much more efficient and scalable algorithm, which has become

a widely used baseline in RL.
Its relevance to this thesis is, firstly, the PPO agent implemented in this work uses the original PPO formulation. Secondly, GRPO builds upon PPO's clipped objective function, but replaces a critic network with group-based comparisons. By directly comparing GRPO to PPO, this thesis aims to address whether GRPO can provide comparable stability while benefitting from reduced architectural complexity.

This thesis builds upon the foundational work presented by DeepSeek in the paper *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models* [SWZ+24]. This was their first paper introducing GRPO in LLMs, popularizing this RL algorithm, which leverages an actor-only approach and replaces a critic network by grouping multiple responses (roll-outs) to the same prompt. With this implementation, it calculates group-based advantages, which are then incorporated into a clipped PPO-style loss. The original GRPO work demonstrated promising results in LLM tasks, especially in mathematical reasoning, achieving better results to the then open-source models. Further research on this topic, such as the DeepSeek-R1 paper [Tea24], showcased GRPO's strong potential, with the DeepSeek-R1 model achieving performance comparable to OpenAI's o1 model.
This thesis takes the GRPO algorithm out of the LLM domain and investigates its performance in more traditional RL environments. By doing so, this thesis will analyze whether GRPO's benefits, such as computational efficiency, stability and overall reduced complexity, are transferable to environments where reward structures, state-action separation and feedback timing are significantly different from LLM tasks. Furthermore, this work will clarify how GRPO's performance varies with different group sizes and how resource constraints impact the algorithm. As a result, the thesis extends the existing research by providing insights into GRPO's broader applicability.

One of the most relevant extensions to the original GRPO framework by DeepSeek is Hybrid Group Relative Policy Optimization, proposed by Soham Sane earlier this year [San25]. While GRPO eliminates the need for a value function by utilizing a value estimation with empirical comparisons between grouped traces, Hybrid GRPO combines the strengths of both PPO and GRPO by reintroducing a value network. The reasoning behind this is that although GRPO with its actor-only architecture can reduce variance through its implementation of multiple roll-outs, it may be subject to higher variance and less stable learning in high-variance environments. Furthermore, Hybrid GRPO incorporates a critic network that estimates value baselines via bootstrapping, leading to greater sample efficiency alongside reducing variance and improving learning stability. The relevance of this work to the current thesis lies in its demonstration of how effective GRPO-based approaches are in RL environments. While GRPO has shown promising results in LLM tasks, Hybrid GRPO illustrates how the core principles of GRPO can translate into standard RL environments as well. The Hybrid GRPO paper presents promising results by combining GRPO with a value function baseline, leading to less variance in policy gradients and faster convergence speed. However, this implementation undermines the original simplicity and computational efficiency of GRPO. Moreover, the paper fails to specify in what way Hybrid GRPO converges faster and does not provide data supporting its findings. It is important to note that it also does not discuss any form of computational trade-offs involved, as one would suppose that the reintroduction of a value network would suggest a higher computational cost. This thesis explicitly addresses and compares the performances of

PPO and GRPO, aiming to offer a more complete evaluation of GRPO's cost-benefit trade-offs.

3 Preliminaries

This section will lay the theoretical foundation of the two RL algorithms compared in this thesis, namely PPO and GRPO. These policy gradient methods are designed to optimize their policies, given an environment.
Both agents interact with MDP environments, which, as briefly discussed before, serves as the basis for defining the learning task. In this setting, the agent observes a state $s \in S$ and selects action $a \in \alpha$ according to a stochastic policy $\pi_\theta(a|s)$, receives reward r_t and transitions to a new state. The goal is to find an optimal policy π^* such that it yields the highest expected cumulative reward. Both PPO and GRPO rely on policy gradients, but they differ in how they calculate the loss and estimate advantages. Therefore, the following subsections will delve deeper into the mathematical formulations and intuitions behind PPO and GRPO, highlighting both their similarities, but more importantly their critical differences. Understanding these foundations is crucial for interpreting the results and comparing both algorithms.

3.1 Theoretical Framework

3.1.1 PPO

OpenAI's innovation in 2017, namely PPO [SWD+17], improved upon previous policy gradient methods, such as TRPO [SLM+15], by introducing a clipped objective function to prevent unreasonably large updates, as can be seen in 1.

$$L^{CLIP}(\theta) = \hat{E}_t \left[\max \left(\rho_t(\theta)(-\hat{A}_t), \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)(-\hat{A}_t) \right) \right] \tag{1}$$

with

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2}$$

As can be seen in 2, $\rho_t(\theta)$ is the probability ratio between the new and old policies. ϵ is a hyperparameter that defines the clipping range (set to 0.2 in the OpenAI paper) [SWD+17]. This clipping mechanism ensures that policy updates remain within the region of $1-\epsilon$ and $1+\epsilon$, thus maintaining stability.
The intuition behind the clipped surrogate loss function is that it encourages improvements in action selection when $\hat{A}_t > 0$, but due to clipping of the probability ratio it is upward bounded, preventing overfitting. Similarly, when $\hat{A}_t < 0$, the loss function decreases the probability of this action being selected while being bounded from below. This subtle form of regularization is what makes PPO particularly attractive for practical reinforcement learning applications.
To compute how good an action is given a state, PPO uses the advantage function $A(s,a) = Q(s,a) - V(s)$. While $Q(s,a)$ represents the state-action value after taking a in s, subtracting $V(s)$ serves as a baseline, centering the advantage around zero. In effect, $V(s)$ tells us what return is typically expected from a state, and the advantage measures how much better or worse a specific action was compared to this baseline.

The Temporal Difference (TD) error δ_t 3 used in advantage estimation can be seen as an approximation of this advantage function at time step t:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3}$$

This expression mirrors $A(s_t) = Q(s_t, a_t) - V(s_t)$, where $Q(s_t, a_t) \approx r_t + \gamma V(s_{t+1})$. To incorporate information from future time steps while balancing bias and variance, PPO uses the Generalized Advantage Estimate (GAE) for its advantage calculation [SML+15]:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{4}$$

GAE is essentially a reweighted average of n-step TD-based advantages, where the parameter $\lambda \in [0, 1]$ adjusts the balance: lower λ leads to lower variance but higher bias, while higher λ increases variance but reduces bias. Here, γ is the discount factor, which determines how much future rewards are relevant relative to immediate rewards [YUD13].
Finally, the total Loss function can be calculated, so that the Neural Networks can adequately learn. Essentially, the total loss is made up of three parts, which are the policy loss 1, the value loss 6 and an entropy bonus:

$$L^{\text{PPO}} = \hat{E}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 \cdot \mathcal{S}[\pi_\theta](s_t) + c_2 \cdot L_t^{\text{VF}}(\theta) \right] \tag{5}$$

with

$$L_t^{\text{VF}}(\theta) = (V_\theta(s_t) - V_t^{\text{target}})^2 \tag{6}$$

In this formula both c_1 and c_2 are constants and $\mathcal{S}[\pi_\theta]$ is the entropy bonus to promote sufficient exploration. Lastly, $L_t^{\text{VF}}(\theta)$ can be described as the squared error loss of the value function.

3.1.2 GRPO

In GRPO, the total loss function contains two main parts, as it refrains from using both a value network, and therefore does not have a value loss, and an entropy bonus.

$$\begin{aligned} L^{\text{GRPO}}(\theta) = {} & E[s \sim P(S),\ \{a^i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\alpha|s)] \\ & \frac{1}{G}\sum_{i=1}^{G} \left(\max\left(\rho(\theta)(-A^i),\ \text{clip}\left((\theta),\ 1-\epsilon,\ 1+\epsilon\right)(-A^i)\right) + \beta D_{\text{KL}}(\pi_\theta||\pi_{\text{ref}}) \right) \end{aligned} \tag{7}$$

with

$$\rho(\theta) = \frac{\pi_\theta(a^i|s)}{\pi_{\theta_{\text{old}}}(a^i|s)} \tag{8}$$

being the probability ratio.
In 7, the maximization part highlighted in red is the policy loss of GRPO. The policy loss of GRPO is exactly the same as the policy loss of PPO, which also makes use of a clipped surrogate function. The second part of the loss function is the KL divergence part, which promotes stable learning.

$$D_{\text{KL}}(\pi_\theta||\pi_{\text{ref}}) = \frac{\pi_{\text{ref}}(a^i|s)}{\pi_\theta(a^i|s)} - \log\left(\frac{\pi_{\text{ref}}(a^i|s)}{\pi_\theta(a^i|s)}\right) - 1 \tag{9}$$

In 8 $\pi_\theta(a^i|s)$ is the current policy's probability of taking action a^i given s and $\pi_{ref}(a^i|s)$ the reference policy's probability, often a previous or smoothed version of the policy. The KL Divergence term ensures that the new policy does not deviate too much from the reference policy, meaning that the more the new policy wanders off from the reference policy, the more the KL divergence term grows. β is a hyperparameter which weights the KL divergence term.
Unlike PPO, which uses an entropy bonus to promote exploration, GRPO omits this term and instead uses KL divergence for regularization. While both serve to ensure stability in policy updates, this difference means that GRPO may implicitly discourage exploration in a different way. As a result, this makes the comparison between PPO and GRPO a bit more nuanced, as the entropy term in PPO controls exploration versus exploitation, while the KL term in GRPO penalizes large policy updates.
Most notably, the advantage calculation in GRPO differs from PPO, as it does not have a value network. In addition, the advantage for each roll-out of the group A^i is computed by using the sum of rewards of the entire trace R of a roll-out.

$$A^i = \frac{R^i - \text{mean}(\{R^1, R^2, \ldots, R^G\})}{\text{std}(\{R^1, R^2, \ldots, R^G\})} \tag{10}$$

with

$$R^i = \sum_{t=0}^{N^i} r_t^i \tag{11}$$

and N denoting the total number of steps taken for roll-out i.
Intuitively, the advantage for roll-out i describes how good the reward of this roll-out was compared to the rewards of the entire group. As a result, GRPO circumvents the need of a value estimate by introducing a sample-based estimate, more specifically the mean cumulative reward across all roll-outs. This enables a simplification of the standard advantage formula as previously mentioned $A(s) = Q(s, a) - V(s)$. Here, $Q(s, a)$ corresponds to the cumulative reward of a roll-out, while $V(s)$ is the mean of all roll-outs across the group. This ensures stable training and allows GRPO to estimate advantages without learning a value function.

4 Methodology

This section outlines the implementation and evaluation of GRPO and PPO. Both algorithms are benchmarked on two classic control environments and are all written in Python. Special attention is given to the architecture and concepts of GRPO, particularly in how it estimates advantages.

4.1 PPO and GRPO Implementation

The central methodological contribution of this thesis is implementing and adapting GRPO, designed for LLMs under the Zero-Reward Assumptions, where the agent receives sparse, binary rewards at the end of a trajectory [HXZW25], to standard RL environments where rewards are immediate and dense. This change introduces new challenges for the advantage calculation and, overall, for the internal workings of the algorithm, as it now must account for intermediate rewards during an iteration.

4.1.1 PPO

The RL environment is initialized before the agent interacts with the environment in discrete time steps, collecting state-action-reward tuples to update its policy. The PPO implementation is based on the CleanRL library and consists of a parallelized environment setup (10 parallel environments), a policy and value network architecture, and an optimization loop[Cle24].
The policy network is a feedforward neural network that takes a state as an input and outputs logits over discrete actions. During each iteration, the agent collects trajectories over several environments in parallel for a fixed number of steps. It records observations, actions, rewards, value estimates, and log probabilities. The value network is also a feedforward neural network that outputs an estimated value given a state. Once all roll-outs have reached the set number of steps, the Generalized Advantage Estimation (GAE) method is used to calculate the advantages for each time step t.
The next part of the code optimizes the policy and value networks. The collected data is flattened and split into minibatches. For each minibatch, the policy and value networks are updated using the PPO clipped objective, which stabilizes training by preventing large policy updates. The value loss is also optionally clipped to ensure stability. Entropy regularization is added to encourage exploration. Gradient clipping is applied to avoid exploding gradients.
Lastly, a greedy actor is implemented to evaluate the learned policy.
This setup ensures both stable training via PPO and a consistent evaluation protocol for comparison with the GRPO variant. A list of PPO's most important hyperparameters and their respective values can be found here 3.

4.1.2 GRPO

The GRPO implementation has some similarities to PPO, whilst having some major key differences. The most noticeable difference is that GRPO only has a policy network. It is important to note that the GRPO implementation was adapted from the existing PPO implementation by CleanRL[Cle24], and the full implementation can be found in the following GitHub repository [1].
GRPO begins by synchronizing the starting states across all roll-outs within a group after the environment initialization. Each roll-out completes a whole trajectory during which the observations, actions, log probabilities, logits, rewards and total number of steps per roll-out are stored. After all roll-outs have finished their trajectories, the cumulative rewards of each roll-out is used to calculate the advantage for the entire trajectory for each roll-out. respectively. Unlike PPO, which computes advantages at each timestep, GRPO treats each roll-out's trajectory as one unit.
During policy optimization, GRPO uses the standard PPO clipping function to constrain the policy update, but adds a KL divergence penalty to ensure that the new policies do not deviate too much from the old policy. This term reduces overfitting to high reward trajectories. Finally, the policy loss is averaged across all roll-outs of a group to compute the total GRPO loss.
GRPO maintains PPO's general structure, including parallel environment roll-out, learning rate annealing and data storing. However, it shifts the focus from fine-grained timestep-level optimization to more general, group-level adaptation, enabling more stable updates when episodic rewards are sparse or highly variable. A list of GRPO's most important hyperparameters and their respective values can be found here 3.

[1] https://github.com/AlexanderCremer/GRPO-vs-PPO-in-RL-env

Algorithm 1 Group Proximal Policy Optimization (GRPO)

```
Initialize G environments and actor network
for 1,2,...,num_iterations do
    Reset environments to identical initial states
    Initialize logging
    for 1,2,...,num_steps do
        Get actions a_t
        Take step
        Store observations
        Update cumulative rewards and done masks
        if all groups done then
            break
        end if
    end for
    Compute advantages
    for num_epochs do
        for group G do
            Compute clipped policy loss
            Compute KL divergence penalty
        end for
        Compute total loss:
        Perform gradient descent
    end for
    Evaluate greedy policy
end for
```

4.1.3 Understanding GRPO

The following example aims to illustrate how GRPO computes its advantages and what the training targets are, given a CartPole-like environment, where the agent gets a reward of +1 for every step.

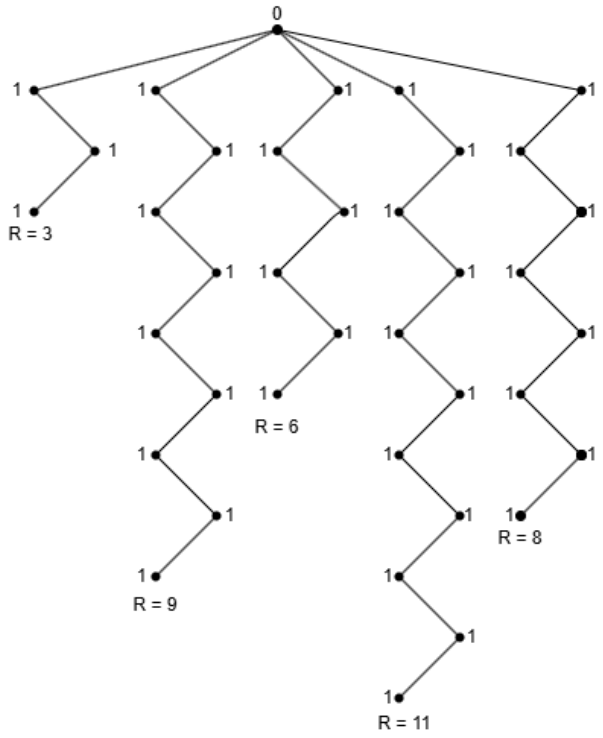


Figure 1: Illustration of GRPO Advantage calculation using full roll-out rewards on a CartPole-like environment. Here, states are represented as nodes and actions as edges with rewards assigned to states after taking an action.

Each branch in Figure 1 represents a roll-out, that is, a sequence of states and actions until the environment terminates. A group size of 5 was chosen with different roll-out lengths to highlight the process of GRPO. For each roll-out their respective cumulative reward R is given below its trace in the Figure.
Importantly, all roll-outs are initialized from the same starting state (represented by the root node with r=0 in Figure 1) to ensure that they are all directly comparable. This is a key requirement in GRPO, as it allows the cumulative rewards to be meaningfully standardized to their advantages respectively. Without a consistent starting state, differences in rewards could stem from environmental variance rather than policy performance, making advantage estimation unreliable.
Unlike PPO, which estimates advantages for each time step, GRPO computes a single advantage value per roll-out based on its total return relative to the other roll-outs in the group. This is done by normalizing the return R_i of each roll-out using the mean and standard deviation of returns across all roll-outs in the group, as can be seen in equation 10.
For the above example from Figure 1, the cumulative rewards for each roll-out are 3, 9, 6, 11, 8 respectively. The mean reward is 7.4 and the standard deviation is 2.73, resulting in the following advantages:

- Roll-out 1 $(R^1 = 3) \rightarrow A^1 = \frac{3-7.4}{2.73} \simeq -1.61$
- Roll-out 2 $(R^2 = 9) \rightarrow A^2 = \frac{9-7.4}{2.73} \simeq 0.59$

- Roll-out 3 ($R^3 = 6$) $\rightarrow A^3 = \frac{6-7.4}{2.73} \simeq -0.51$
- Roll-out 4 ($R^4 = 11$) $\rightarrow A^4 = \frac{11-7.4}{2.73} \simeq 1.32$
- Roll-out 5 ($R^5 = 8$) $\rightarrow A^5 = \frac{8-7.4}{2.73} \simeq 0.22$

These normalized advantages are used to calculate the GRPO policy loss, analogous to PPO. Importantly, this loss is propagated to all time steps within the respective roll-out. This means that all state-actions contribute to the gradient update.
Once an iteration is finished, all parallel environments are reset, with their root nodes (node at the top with r=0 in Figure 1) being set to the same starting state. In this implementation, I make use of so-called exploring starts [SB98], where each episode begins in a random state sampled from a broad distribution. This increases state-space coverage and encourages exploration, which is especially beneficial for evaluating policy performance more robustly and reducing the risk of overfitting to a narrow subset of initial conditions.
The innovation of replacing a value function with a sample-based baseline, reduces the potential bias introduced by a value network, since it only approximates expected values of a state. Simultaneously, standardizing the advantages helps reduce variance which leads to stabilized training, particularly when the group size G is large.

Another important distinction between PPO and GRPO lies in the amount of training data generated. In PPO, advantages and values are calculated for each time step, meaning that each step becomes a separate training target. For example, with 10 parallel environments each generating 200 steps per iteration, PPO yields 2000 training samples per update cycle. These are then divided into four minibatches, which are each iterated over for four epochs.
In contrast, GRPO computes one advantage value per entire roll-out, meaning that the number of training targets is equal to the group size. This results in significantly fewer data points, just 10 per iteration in the same example as above. Consequently, GRPO uses a different batching and optimization strategy to compensate for this reduced data volume. Instead of using multiple minibatches, all data points are kept in a single batch, and the model is updated over four epochs as well. To partially offset this difference, GRPO is trained with a slightly higher learning rate and retains the same number of update epochs per iteration as PPO, as can be seen in the hyperparameter table 3.

4.2 RL Environments

The first environment that the two algorithms are benchmarked on, is the CartPole environment, included in the `gymnasium` library by the Farama Foundation [Fou24b]. The task consists of balancing a pole on a cart by applying forces to the left or right. The state space is continuous and consists of four variables: the cart position and velocity, and the pole angle and angular velocity. The action space is discrete with two possible actions (left or right). The episode ends when either the agent fails, meaning the pole falls beyond a certain angle or the cart moves outside a predefined boundary, or when the cart has reached the maximum number of steps in the environment, which is 200. A reward of +1 is given for every time step the pole remains balanced, with the goal being to maximize the cumulative reward and no reward when it fails. Despite its simplicity, CartPole serves a useful test for validating learning stability and convergence behavior in reinforcement learning

algorithms.

The second environment used for benchmarking is the Acrobot environment, which is also part of the `gymnasium` library [Fou24a]. The Acrobot is a two-link, underactuated robotic arm, where only the second joint is actuated. The objective is to swing the end of the lower link (the tip of the "arm") up to a target height above a certain threshold. The environment's state space consists of six continuous variables: the cosine and sine of the two joint angles, and the angular velocities of both joints. The action space is discrete, consisting of three actions that apply a fixed torque in one of two directions or no torque at all.
An episode ends either when the tip of the lower link reaches a target vertical height or after a fixed number of time steps (default is 500). A reward of -1 is assigned at each time step until the goal is reached, encouraging the agent to solve the task in as few steps as possible. The environment is considered solved, when the trained model consistently reaches the goal in ≤ 100 steps.

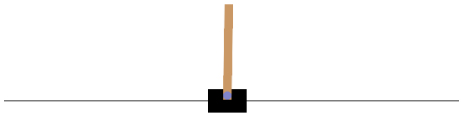


Figure 2: CartPole environment from Gymnasium. The agent must balance a pole on a cart by moving left or right [Fou24b].

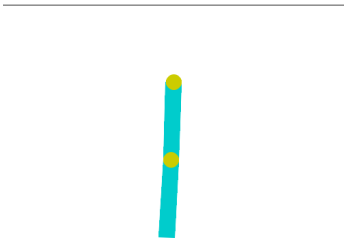


Figure 3: Acrobot environment from Gymnasium. The agent controls a two-link pendulum to swing up and reach a target height (indicated by a line in the figure) [Fou24a].

4.3 Evaluation Metrics

The first evaluation metric implemented to evaluate performance is the mean cumulative reward, plotted against the number of global steps and computation time. Here, global steps refer to the total sum of steps an agent has taken up to a given point. These metrics are commonly used in Reinforcement Learning to assess both the consistency and peak capability of an agent's policy during training.
The implementation of these metrics is directly integrated into the training loop. After each iteration, with an iteration being one complete run of all roll-outs, cumulative episode rewards are collected across all episodes of a group (for GRPO) and parallel environments (for PPO). From this, the mean reward is computed. This value is then logged using Weights and Biases [Bie20], which provides real-time visualization and performance tracking.
Secondly, a greedy actor evaluation is used to effectively assess the quality of the policy that is being trained. It is important to note that the number of learning iterations is fixed at 1,000 for all experiments, regardless of the model or group size. This was done to unify the number of network update rounds across both algorithms and for both environments. However, there are other metrics

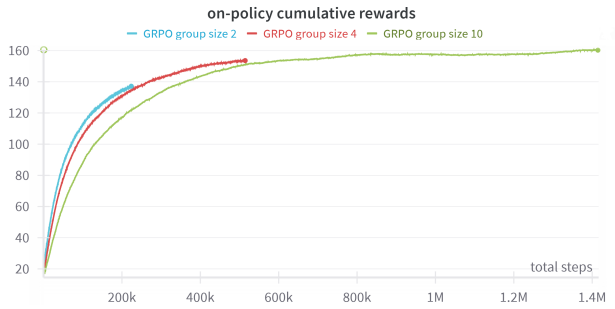
to consider for fair comparison, such as computation time or total environment steps. Therefore, when plotting these metrics on the x-axis, the graphs may differ in length across models and group sizes, as they process different amounts of data per learning iteration. The greedy actor selects the action with the highest probability given a state, as opposed to sampling from the probability distribution. This greedy actor runs in evaluation environments, which do not interfere with the training process, to ensure stable and unbiased measurement. The evaluation is performed at the end of each iteration, therefore after all roll-outs have completed and backpropagation has been applied. The mean cumulative reward over 10 independent evaluation runs is computed and logged, providing a more robust comparison of PPO and GRPO in terms of exploitation performance under deterministic conditions. Although the policy and environment dynamics are deterministic under a greedy actor, the initial state is sampled stochastically, which may introduce variability. Hence, repeated runs are necessary to account for differences resulting from starting states and to ensure a fair comparison.

5 Results

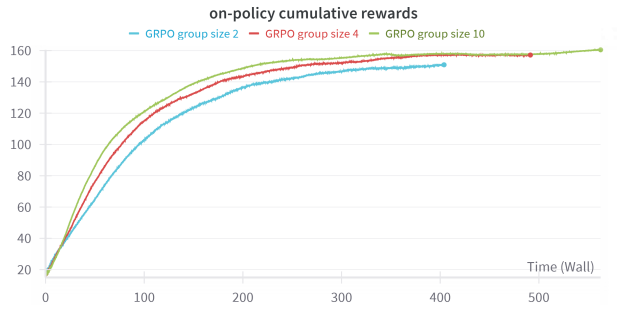
The environments used for evaluation are CartPole and Acrobot, both of which serve as valuable benchmarks for comparing the learning dynamics of GRPO and PPO. It is important to note that each configuration, meaning for all GRPO group sizes and all PPO parallel environments, was run multiple times with random seeds (10 times for CartPole and 5 times for Acrobot) to minimize statistical bias.
Furthermore, PPO was configured with 10 parallel environments, which aligns with the group size of 10 in GRPO to which it will be compared to. This ensures a more balanced and meaningful comparison in terms of environment roll-outs per learning iteration.
For CartPole, the agent is rewarded +1 for every time step the pole remains balanced, making it a useful test for evaluating stability and learning speed. In contrast, Acrobot penalizes the agent at every time step with a reward of -1, incentivizing it to solve the task as quickly as possible by minimizing the number of steps. To guarantee an unbiased and proper depiction of the mean cumulative rewards, the mean graphs were truncated at the point where the earliest seeded run completed.

5.1 Effect of Group Size in GRPO (on-policy)

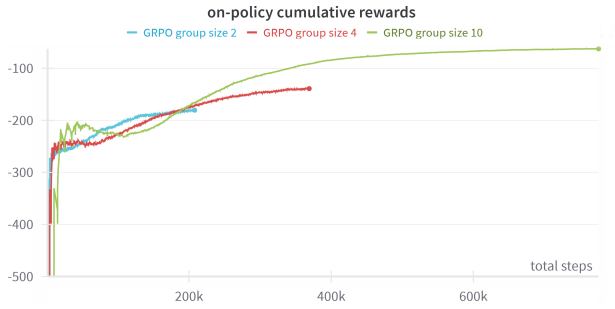
Figure 4 presents a comparison of GRPO's on-policy performance with varying group sizes G = {2, 4, 10} for both the CartPole as well as the Acrobot environment. Each curve represents the smoothed episodic return over the course of training. These group sizes were chosen as they yielded the most promising results, given convergence speed and value. The on-policy training on the CartPole environment was capped at a maximum of 200 steps per roll-out, as this was found to be sufficient to reach stable and competitive performance, details of which will be discussed later.



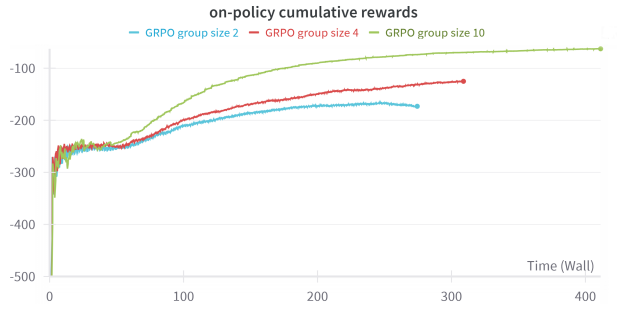
(a) CartPole: On-policy cumulative reward vs. total environment steps



(b) CartPole: On-policy cumulative reward vs. computation time



(c) Acrobot: On-policy cumulative reward vs. total environment steps



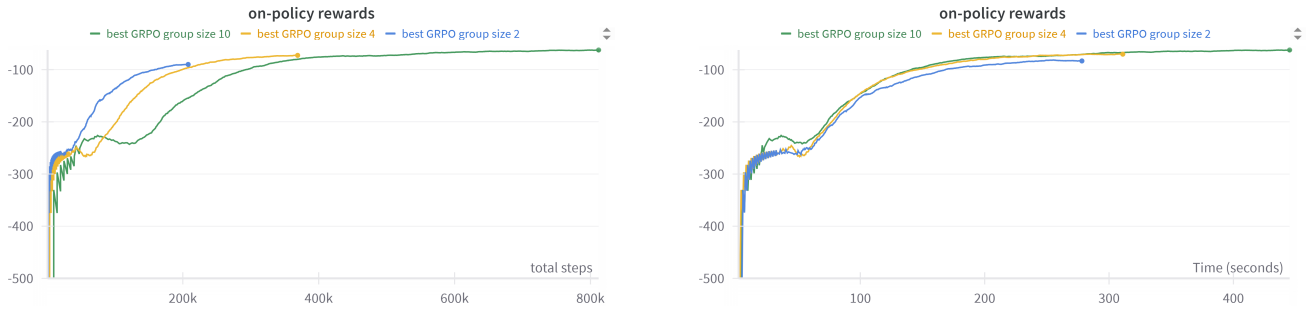
(d) Acrobot: On-policy cumulative reward vs. computation time

Figure 4: Performance comparison of GRPO across two environments. Each plot shows the cumulative on-policy reward averaged across multiple runs (10 for CartPole and 5 for Acrobot). The top row corresponds to the CartPole environment, and the bottom row to Acrobot. Left: performance measured by total environment steps. Right: performance measured by computation time.

As can be seen in the left plots of Figure 4 the group size has a big effect on the total amount of steps taken in the environment. This is expected, as greater group sizes correspond to more environments running in parallel, which accelerate data collection and lead to faster progression in terms of total steps. Essentially, the algorithm experiences more steps per learning iteration, resulting in a less steeper learning curve when measured against total environment steps.
Figure 4a shows an interesting trend: smaller group sizes outperform larger ones when measuring cumulative reward against total steps. Here, G=2 achieves higher sample efficiency than G=4 and G=10. This reversal suggests that the optimal group size depends heavily on whether one is prioritizing sample efficiency (reward per step) or training speed (reward per wall-clock time).
However, when plotting cumulative reward against wall-clock time, a different picture emerges, as can be seen in Figures 4b and 4d. Despite GRPO with group size G=10 taking substantially more steps than G=2, it completes training only slightly later and achieves the highest performance at every point in time. This illustrates a key trade-off: a moderate increase in computation time enables faster learning and more stable convergence. The results also suggest that, although larger group sizes generate higher throughput, the computational cost scales efficiently due to effective parallelization. It is important to note, that this only works as long as the hardware can efficiently

parallelize it, which will be further discussed in Section 6.

As seen in Figure 4c, the expected pattern, where smaller group sizes perform better per step, is not as pronounced in the Acrobot environment. This is largely due to badly seeded runs that fail to learn, particularly for G=2 and G=4, skewing the average results. To isolate this effect, Figure 5 shows only the best runs for each group size. In these plots, the expected efficiency reappears: smaller group sizes outperform larger ones when comparing steps taken. And over time, all group sizes converge to nearly the same reward, showcasing similar results to those of CartPole.



(a) Acrobot: best on-policy cumulative reward over total steps taken

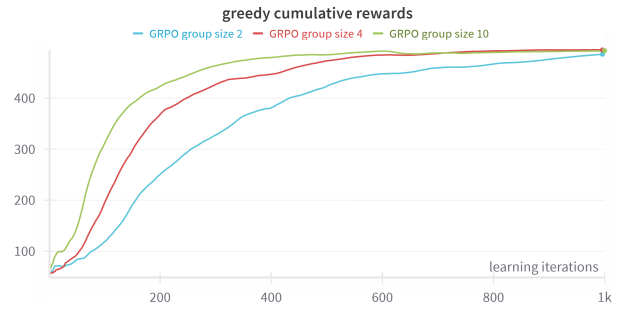
(b) Acrobot: best on-policy cumulative reward over time

Figure 5: The best performing runs of G = {2, 4, 10} in the Acrobot environment.

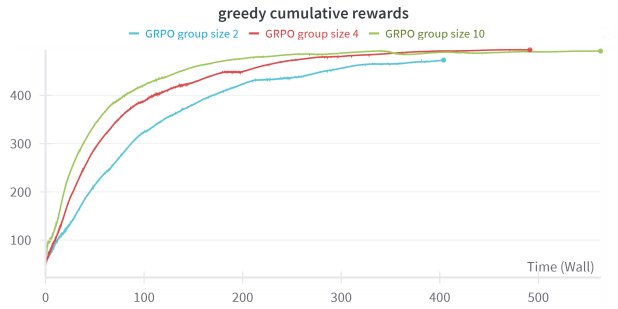
In terms of converged value, for G = 10 GRPO yields marginally better results. This can be attributed to gradient estimates resulting from higher sample diversity and reduced variance in policy updates. However, the difference between G = 10 and G = 4 is not substantial, indicating diminishing returns beyond a certain group size. This happens especially in simple environments, where the task complexity is low. Yet, the consistency with G = 10 across both environments suggests that increasing G may lead to more stable and reliable results across different seeds. This supports the key observation that larger group sizes trade off additional computation time/steps for more stable learning, highlighting an important design choice when implementing GRPO.

5.2 Effect of Group Size in GRPO (greedy)

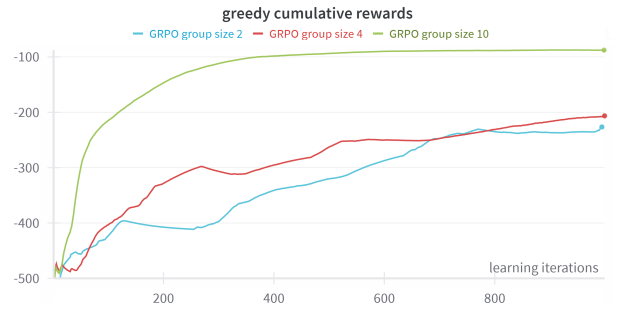
To evaluate the learned policy's improvements throughout training, the performance using a greedy actor is also tracked. After each learning iteration, the current policy is executed deterministically, without exploration noise, across a full episode. This is why for the plots 6a and 6c the mean cumulative reward is plotted against the 1000 learning iterations. The resulting rewards from this evaluation provides a reliable measure of how well the agent performs when exploiting its learned policy up to that point. It is important to note that the maximum achievable mean cumulative reward was set to 500 for CartPole, to effectively demonstrate that despite the on-policy training being capped at 200, it generalizes well and achieves good results under evaluation.



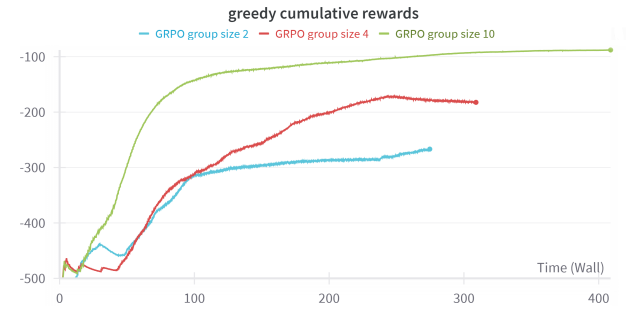
(a) CartPole: greedy cumulative reward over learning iterations



(b) CartPole: greedy cumulative reward over time



(c) Acrobot: greedy cumulative reward over learning iterations



(d) Acrobot: greedy cumulative reward over time

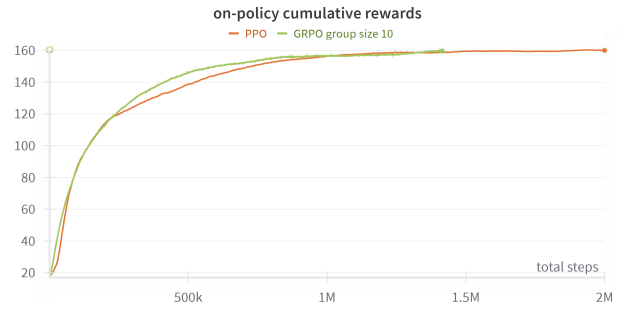
Figure 6: Evaluation of GRPO with group sizes $G = \{2, 4, 10\}$ under greedy (exploitation-only) conditions. Plots show the mean cumulative reward achieved by the greedy policy over the course of training, measured in terms of learning iterations and wall-clock time for both CartPole and Acrobot.

For CartPole, the results are particularly striking. In Figure 6a the group size G = 10 yields the most promising results, as it converges the quickest and seems to be the most stable across runs. This can be explained with the fact that every learning iteration is trained on data from 10 entire roll-outs as opposed to 4 or 2. The increased diversity and sheer volume of experiences per update lead to more accurate gradient estimates and faster learning. However, all three group sizes converge to identical values after all 1000 learning iterations, due to the simplicity of the environment.
When plotting cumulative rewards over time, as in Figure 6b, G = 10 and G = 4 reach comparable performance in terms of converged cumulative reward, while G = 2 lags slightly behind. Nevertheless, GRPO with G = 10 achieves the fastest learning, despite taking more environment steps per learning iteration. This indicates that the benefit of increased training data per learning update outweighs the additional computational cost, making the larger group sizes that were chosen more favourable. Again, this trade-off is only beneficial as long as the hardware can efficiently parallelize the roll-outs. Overall, the relative performance of the group sizes in the evaluation setting is similar to that of the on-policy performance, with the differences being slightly more highlighted when using a purely exploitative actor.
For Acrobot, the trends are consistent with the CartPole results but more extreme. In both Figures

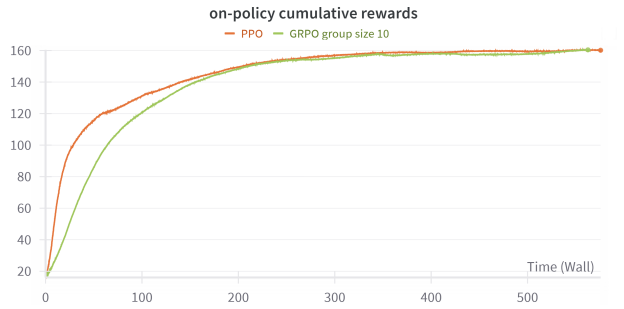
6c and 6d, GRPO with G = 10 GRPO yields good results with the greedy actor also managing to solve the environment (consistently reaching a cumulative reward of ≥ -100). In contrast, group sizes G = {2, 4} show significantly lower performance, which can again be explained by the presence of poorly performing seeds. These lead to longer training and significantly worsens the average performance curve, as was observed in the on-policy performance. In addition, and as was noticeable in the CartPole environment, the greedy actor seems to emphasize already existing differences in the on-policy performance, which is why the group sizes G = {2, 4} perform considerably worse. In terms of converged value, G = 10 shows a slightly worse converged value with the greedy actor than with the on-policy actor, with values ~ -88 and ~ -64 respectively. Although the greedy actor strategy leads to stable and good performance under pure exploitation, it may also miss out on beneficial stochastic exploration and reveals a critical limitation. A purely deterministic policy can suffer greatly from isolated but crucial mistakes in its learned policy, which it may never correct.

5.3 PPO vs GRPO

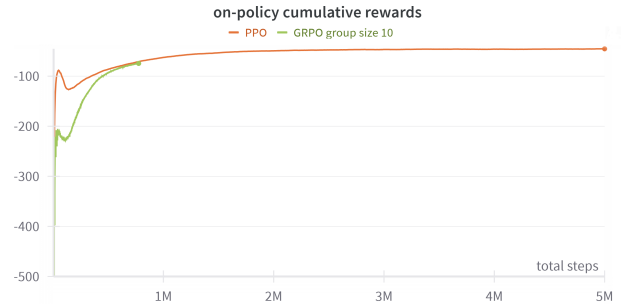
To assess the performance of GRPO against PPO, the configuration for GRPO with G = 10 is chosen, as it yielded the best overall results across all group sizes. For a fair comparison, it is important to note that the PPO baseline was run with 10 parallel environments.



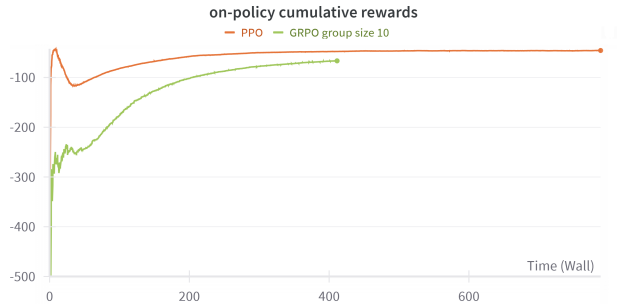
(a) CartPole: cumulative on-policy reward over total steps



(b) CartPole: cumulative on-policy reward over time



(c) Acrobot: cumulative on-policy reward over total steps



(d) Acrobot: cumulative on-policy reward over time

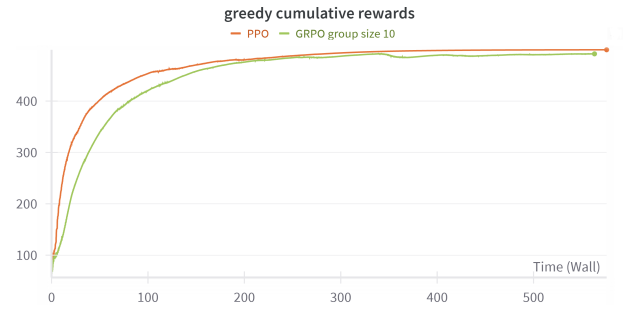
Figure 7: Comparison of PPO and the best performing GRPO configuration on CartPole and Acrobot. The plots show cumulative on-policy rewards across different metrics: total environment steps and computation time.

In the CartPole environment, GRPO performs on par with PPO, with a phase in the middle where GRPO exhibits faster learning, when performance is measured against total steps, as can be seen in Figure 7a. This indicates, that in terms of sample efficiency, i.e. the amount of data needed to achieve a certain performance, GRPO is at least on par with PPO. The comparison showcases how sample efficient the algorithms are. Despite their structural differences, with PPO consistently taking 2000 steps per learning iteration, whereas GRPO may take fewer, the key insight here is that GRPO matches PPO's performance with fewer environment interactions, highlighting its potential in scenarios where sample efficiency is critical.
In contrast, when plotting performance against computation time, as shown in Figure 7b, PPO achieves slightly faster learning and is marginally better in terms of computational efficiency. Despite, PPO's lower sample efficiency, it benefits from effective parallelization and high data throughput per learning iteration, allowing it to collect large batches of data quickly.
Across both Figures 7a and 7b, GRPO and PPO converge to nearly identical cumulative rewards. Nevertheless, these Figures highlight a trade-off between sample efficiency (favouring GRPO) and compute efficiency (favouring PPO).
In the Acrobot environment one can deduce from Figure 7c, that PPO takes significantly more steps than GRPO due to its fixed number of steps per iteration, while GRPO dynamically reduces its

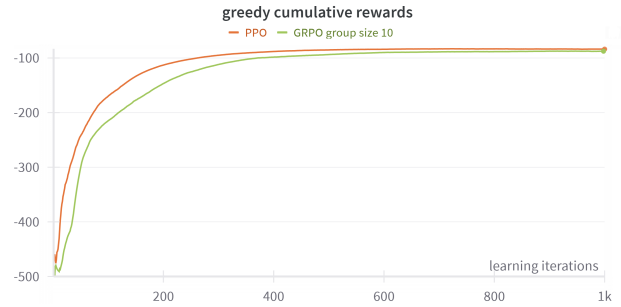
steps as performance improves. Due to the very nature of the environment and GRPO's design, this can result in a gradual slowing of its learning speed over time due to decrease in data throughput. Nevertheless, GRPO catches up to PPO's performance after an initial phase where PPO has the advantage.
When plotting their performances over time, as in Figure 7d, PPO consistently outperforms GRPO at every point in time. It achieves higher rewards more quickly, indicating faster learning from a computational standpoint. This can be attributed to PPO's fixed parallel roll-out structure, collecting 5000 steps per iteration across its parallel environments, ensuring a higher and steadier data throughput at the cost of slightly more computation. This effective implementation causes an imbalance in the data-to-compute trade-off, where the increased data throughput more than compensates for the additional computational time, resulting in a higher computational efficiency. However, this advantage only holds as long as the hardware can efficiently parallelize the environments. Otherwise, additional parallel environments may decrease its computational efficiency.
Overall, PPO appears computationally more efficient, as it effectively leverages its use of parallel roll-outs to accelerate learning at a moderate increase in computational time. Nonetheless, GRPO remains competitive when it comes to sample efficiency, achieving respectable performance.

The greedy evaluation plots in Figure 8 compare the performance of PPO and GRPO in terms of cumulative reward over both learning iterations and computation time.

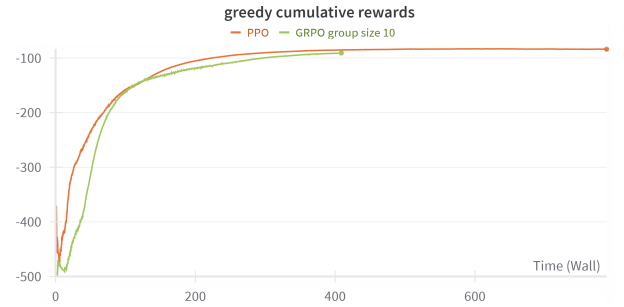
(a) CartPole: greedy cumulative reward over learning iterations



(b) CartPole: greedy cumulative reward over time



(c) Acrobot: greedy cumulative reward over learning iterations



(d) Acrobot: greedy cumulative reward over time

Figure 8: Comparison of PPO and the best-performing GRPO configuration using greedy evaluation on CartPole and Acrobot. The plots show cumulative rewards over learning iterations and computation time.

Across all plots in Figure 8, PPO outperforms GRPO. Nevertheless, PPO showcases only marginally better results with GRPO eventually converging to similar values in all plots. Therefore, despite PPO having a significantly higher data throughput per learning iteration, especially in the Acrobot environment, it performs only slightly better and only initially, as can be seen in Figures 8a and 8c. This potentially indicates that GRPO has a higher sample efficiency, extracting more information per environment interaction. This is particularly evident in the Acrobot setting, where PPO collects more data per iteration due to its fixed 500 steps per environment. Yet, GRPO manages to close the performance gap over time and "solves" the environment only shortly after PPO, despite collecting fewer and shorter roll-outs per iteration. GRPO's use of a more aggressive learning rate likely plays a role here, helping to amplify the updates made from less data and a smaller number of minibatches.
Moreover, when evaluating performance over computation time in Figures 8b and 8d, the gap between PPO and GRPO is even smaller, making GRPO even more competitive. This suggests that although PPO's design favours high throughput and rapid updates, GRPO's lightweight and adaptive sampling strategy can offer advantages in environments where sample budgets are constrained.
In summary, while PPO exhibits slightly faster learning, GRPO proves to be a robust alternative,

delivering identical final results at a potentially lower sample cost, highlighting its potential in sample-efficiency-critical applications.

Finally, it is also important to analyze the stability alongside the converged value of the two different models. This can be shown by calculating the mean cumulative reward of the last 5% across all seeded runs, as well as the average standard deviation across the final 5% of training, as can be seen in Table 1.

Algorithm	**CartPole**		**Acrobot**	
	Mean	**Std**	**Mean**	**Std**
GRPO	159.07	18.43	-61.91	11.60
PPO	160.67	16.91	-44.82	9.39

Table 1: Comparison of PPO and GRPO on CartPole and Acrobot using on-policy evaluation. Values reflect the mean and standard deviation of the average reward over the final 5% of training episodes.

For CartPole, both algorithms converge to similar mean cumulative rewards and only slightly differ in standard deviation, with PPO being marginally better. The minor difference in standard deviation indicates that, despite PPO being slightly more stable, GRPO remains competitive. This suggests that for this environment, both algorithms yield consistent and stable policies.
In contrast, in the Acrobot environment, PPO yields a significantly higher mean cumulative reward, demonstrating its superior final performance. However, the difference in standard deviation across both methods is relatively small, highlighting that while GRPO lags in final performance, it still offers competitive stability across runs. The reason for this pronounced difference lies in the nature of this environment, with GRPO taking fewer steps as its policy improves, reducing data throughput over time, while PPO's data throughput stays consistent and high throughout its training process. These results indicate that in environments where GRPO and PPO have comparable data throughput during their learning process, such as CartPole, GRPO can match PPO's performance. Moreover, the Acrobot results highlight GRPO's strength in stability, showcasing that even when its converged performance is significantly worse, the variability of its results remains relatively low.

6 Discussion

6.1 Diminishing Returns of larger Group Sizes

The results from the experiments conducted in the CartPole and Acrobot environments, provide valuable insights into the strengths and weaknesses of GRPO in RL. One notable trend that occurs is the diminishing returns with increased group sizes, as can be seen in Table 2.

	Greedy			On-policy		
	Group Size 2	Group Size 4	Group Size 10	Group Size 2	Group Size 4	Group Size 10
CartPole	473	494	492	137	154	160
Acrobot	-267	-182	-88	-180	-139	-63

Table 2: Greedy and on-policy performance measures for different group sizes in CartPole and Acrobot environments (from Figures 6b, 6d, 4a, and 4c). Values represent converged cumulative performance (final smoothed-out values).

In CartPole we see a clear performance gain as the group size increases from 2 to 4. However, the subsequent change from group size 4 to 10 in final cumulative performance is minimal. This plateau indicates that after a certain group size, the additional gain from implementing further parallel roll-outs is little to none, due to redundancy of experiences accumulated. Additionally, it is important to note that the environment's upper bound on achievable reward also leads to diminishing returns, as, for example in the greedy evaluation, both G=4 and G=10 are already reaching close to the maximum limit of 500.
Despite the diminishing returns with increased group sizes not being as prevalent for Acrobot, particularly for the greedy actor, it is still present. While the jump in performance for the greedy evaluation from group size 2 to 4 is substantial (difference of 85), and even more so from 4 to 10 (difference of 94), the rate of improvement per added roll-out is significantly lower (from ~ 45 to ~ 17). This suggests that although larger group sizes ensure a more consistent and reliable performance, the benefit from adding more roll-outs decreases. This diminishing return is to be expected for two reasons. Firstly, the environment poses upper limits on achievable performance, capping the improvements even with better policies. Secondly, the statistical benefit of additional samples decreases over time, as known from the 'law of large numbers' in statistics [Sed15]. In addition, while larger group sizes may improve performance, one must take the increase of computation time into consideration, making it critical to identify an optimal trade-off point for real-world applications.

6.2 Choosing the Group Size G

Building upon the analysis from above, certain recommendations emerge in regards of the group size for GRPO.
The optimal group size for GRPO is highly dependent on the complexity of the environment, the software and the hardware. For simpler environments, such as CartPole and Acrobot, larger group sizes are not necessary, as they yield little to no improvements in exchange for significantly longer computation times. However, for more complex environments with higher observation and action spaces, larger group sizes would be needed. This is because larger group sizes are able to collect more data from diverse trajectories, leading to a broader coverage of states and actions. In these settings, GRPO would benefit from more roll-outs to counteract the risk of overfitting to narrows regions, which is especially important when dealing with complex dynamics or sparse rewards.
When choosing the appropriate group size one must consider the software in combination with the hardware given as well. In environments such as Gymnasium by the Farama Foundation, the parallel running roll-outs run on the CPU. Therefore, the amount of roll-outs that can be efficiently run in parallel depends on how many CPU cores are integrated in the system. Essentially, the

efficiency of GRPO with parallel running roll-outs does not only depend on the algorithm, but also on the software and hardware configurations.
Importantly, in environments where taking steps are very costly, such as in the real world or heavy physics based domains, the choice of the group size plays an important role as well. In such scenarios, maximizing sample efficiency is the priority. Therefore, selecting a moderate group size that does not take too many steps, yet yields good results is advisable.
Ultimately, the best group size is the one that balances both computation time and efficiency the best while taking hardware limitations into account.

6.3 Reflections on Experimental Design

One of the potential weaknesses of my implementation was setting the learning iterations to the fixed metric that is standardized between PPO and GRPO. While this decision ensures consistency and a somewhat fair comparison across models, learning iterations are ultimately an arbitrary metric that do not account for the differences in how each algorithm processes data in training. For instance, PPO consistently collects a set amount of environment steps per iteration and divides this into four minibatches, which are used to perform gradient updates. In contrast, GRPO calculates a single advantage per entire roll-out, applying it to all steps in that roll-out and dividing this into only one minibatch. Therefore, selecting a different termination criteria could have made the comparison more intuitive. For example, the training could have used a performance-based stopping criteria, such as when the agent reaches a certain cumulative reward. Fixing the number of iterations may have also downplayed the time efficiency of GRPO and how it exchanges a critic network for more simulation.

7 Conclusions and Further Research

This thesis provides a systematic comparison of GRPO and PPO in reinforcement learning environments. GRPO displays competitive results compared to PPO, despite being a more recent and less studied algorithm with a simpler architecture.
Given the results from Section 5 one can conclude that GRPO yields promising results as its performances are nearly akin to those of PPO. GRPO's implementation trades off a value network for more simulation in the environment, causing it to eliminate bias from the value network at the cost of retaining some variance. However, GRPO can reduce this variance by increasing the group size G, at the cost of more compute time due to additional simulation.
Overall, PPO exhibits faster learning across all experiments, likely due to its more consistent data collection per learning iteration. However, GRPO performs on par with PPO in terms of converged performance when the appropriate group size is chosen, demonstrating its effectiveness despite its slower learning. In addition, there are slight environment-dependent differences. In Cartpole, GRPO demonstrates almost identical performance to PPO. In Acrobot, however, GRPO lags slightly behind PPO, possibly because the hyperparameters used are better suited for the CartPole environment. Additionally, the goal of Acrobot, unlike CartPole, is to complete the task in as few steps as possible. As a result, PPO benefits from having more training data per iteration as opposed to GRPO, giving it an advantage in these kinds of environments. Lastly, the group size G has a clear influence on GRPO's performance, with larger values of G leading to better performances

and more stable training. However, this comes at the cost of greater computation times, creating a trade-off between learning quality and computation resources. The results indicate diminishing returns beyond a certain point, especially in simpler environments.

It is also important to note that the PPO implementation from CleanRL has all its hyperparameters configured to the ideal values, making it operate at its highest potential. In contrast, the hyperparameters used for GRPO were the result from manually tweaking them based on their observed performance. As such, GRPO's true potential is underrepresented in this thesis. Therefore, applying hyperparameter optimization (HPO) to GRPO could result in significant performance increase, with potential of outperforming PPO. This highlights an interesting direction for future research, where an optimized GRPO implementation could yield deeper understanding and stronger results.
Given the relatively simple environments of this study, GRPO's lower computational cost, due to only having an actor network, was not evident. This is the result of the neural networks and environments being lightweight, limiting the benefits GRPO offers. Future research should extend the comparison to more complex environments, such as procedurally generated ones, where both the generality as well as the computational gains of GRPO would be tested.
Another promising direction for future work is to adapt GRPO to utilize step-based learning instead of resetting the environment after each learning update. In the current implementation, all roll-outs restart from the same initial state sampled from a broad distribution after every learning iteration, which can limit exploration and reduce diversity of state transitions observed in other environments especially. Allowing GRPO to continue roll-outs from the current environment states would enable smoother learning, better capture long-term dependencies, while improving sample efficiency by not oversampling the starting states. This change would be particularly beneficial in more complex and sparse environments, where reaching meaningful states repeatedly is rare or costly.

In conclusion, GRPO deems itself a worthy alternative to the current widely used algorithm PPO, as it combines competitive performance with improved stability. It is surprising how a sample-based solution remains so competitive, even in general RL tasks. With this paper, I introduce the first adaptation of GRPO in general RL settings, demonstrating its promising results, despite its simplified architecture. With future refinement and optimization, GRPO has potential to become even more competitive with the current state of the art.

References

[ADBB17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[Bie20] Lukas Biewald. Experiment tracking with weights and biases. https://wandb.ai, 2020. Software available from https://wandb.ai.

[Cle24] CleanRL. Ppo implementation in cleanrl, 2024.

[Fou24a] Farama Foundation. Gymnasium: Acrobot environment - a classic control task for reinforcement learning; acrobot, 2024.

[Fou24b] Farama Foundation. Gymnasium: Cartpole environment - a classic control task for reinforcement learning; cartpole, 2024.

[HXZW25] Shenghua He, Tian Xia, Xuan Zhou, and Hui Wei. Response-level rewards are all you need for online reinforcement learning in llms: A mathematical perspective. *arXiv preprint arXiv:2506.02553*, 2025.

[iam25] iamrajmihir. A beginner's guide to deep reinforcement learning, 2025. Accessed: 2025-05-17.

[LC17] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[San25] Soham Sane. Hybrid group relative policy optimization: A multi-sample approach to enhancing policy optimization. *arXiv preprint arXiv:2502.01652*, 2025.

[SB98] Richard S Sutton and Andrew G Barto. 5. monte carlo methods. 1998.

[Sed15] Kelly Sedor. The law of large numbers and its applications. *Lakehead University: Thunder Bay, ON, Canada*, 2015.

[SLM+15] John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization (trpo). *CoRR abs/1502.05477*, 2015.

[SML+15] 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.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[SWZ+24] 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, 2024.

[Tea24] DeepSeek Team. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *DeepSeek Research*, 2024.

[YUD13] Naoto Yoshida, Eiji Uchibe, and Kenji Doya. Reinforcement learning with state-dependent discount factor. In *2013 IEEE third joint international conference on development and learning and epigenetic robotics (ICDL)*, pages 1–6. IEEE, 2013.

[ZQW20] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.

Appendix

	PPO	GRPO
learning rate	2.5e-4	2e-4
gamma	0.99	N/A
gae lambda	0.95	N/A
number of minibatches	4	1
number of update epochs	4	4
clipping coefficient	0.2	0.2
entropy coefficient	0.01	N/A
value function coefficient	0.5	N/A
kl coefficient	N/A	0.2

Table 3: A list of the most important hyperparameters used in the experiments for both GRPO and PPO. Their Hyperparameters are the same for both environments.