`CartPole-v1` with A3C: Implementation

1 Implementation details

Architecture: A two-layer MLP ($256 \to 128$ ReLU units) feeds both the policy head and the value head. The network is over-provisioned for a 4-D input so I can later reuse it on harder problems.

Parallelism: Three worker processes share an Adam optimiser whose moment buffers live in shared memory. Each worker pushes gradients every 30 steps (the rollout length) and then synchronises weights.

Key hyper-parameters:

Symbol / name	Value	Comment
γ (discount)	0.995	Standard choice
Rollout length	30	Low variance, responsive
Learning rate	1×10^{-4}	Stable with Adam
Entropy bonus β	0.005	Sustains exploration early on
Max grad-norm	5.0	Prevents gradient spikes
Reward scale	0.5	Keeps returns in a sane range

2 Algorithm

2.1 Roles of the actor and critic networks

Actor: The actor is the policy network. Given a state s, it outputs a probability distribution $\pi_\theta(a \mid s)$ over actions (or the most likely action at test time). During an episode it simply chooses an action from that distribution, so its entire job is to decide *what to do next*.

Critic: The critic is the value network. For the same state s it predicts a scalar score $V_\phi(s)$—the expected future return if the current policy keeps acting from that state. During training this value becomes a low-variance baseline for the policy-gradient update: we form the advantage $A_t = R_t - V_\phi(s_t)$, and the critic's estimate reduces noise in the gradient that the actor receives.

In short, *the actor acts, while the critic judges.* The actor learns to choose actions that maximise reward. the critic learns to evaluate how well the actor is doing and supplies a stable learning signal that speeds up the actor's improvement.

2.2 Advantage function

The *advantage* measures how much better (or worse) an action turned out compared with what the critic expected. For timestep t I compute

$$A_t \;=\; R_t \;-\; V_\phi(s_t),$$

where R_t is the discounted return collected after taking action a_t in state s_t, and $V_\phi(s_t)$ is the critic's estimate of that same return. Because A_t is centered around 0, it acts as a baseline in the policy-gradient update:

$$\nabla_\theta J \;\approx\; \mathbb{E}\big[\,\nabla_\theta \log \pi_\theta(a_t|s_t)\, A_t\big].$$

Subtracting the baseline leaves the gradient's expectation unchanged (so the estimator stays unbiased) but slashes its variance, giving the actor a cleaner and more stable learning signal. In practice this translates into faster convergence and smoother training curves.

2.3 Loss functions

Training minimises the composite objective

$$\mathcal{L} = -\log \pi_\theta(a_t|s_t)\, A_t + \tfrac{1}{2}\,(R_t - V_\phi(s_t))^2 - \beta\, \mathcal{H}\big[\pi_\theta(\cdot|s_t)\big],$$

where the last term is an entropy bonus that discourages premature convergence. Gradients from all three terms are clipped to $\|g\|_2 \le 5$ before the shared Adam step.

3 Environment

- **State**: $(x, \dot{x}, \theta, \dot{\theta}) \in \mathbb{R}^4$.
- **Actions**: `0`=push left, `1`=push right.
- **Reward**: $+1$ per step, capped at 500.
- **Termination**: $|\theta| > 12^\circ$ or $|x| > 2.4\,\text{m}$.

4 Training curves and what they mean

4.1 Episode return

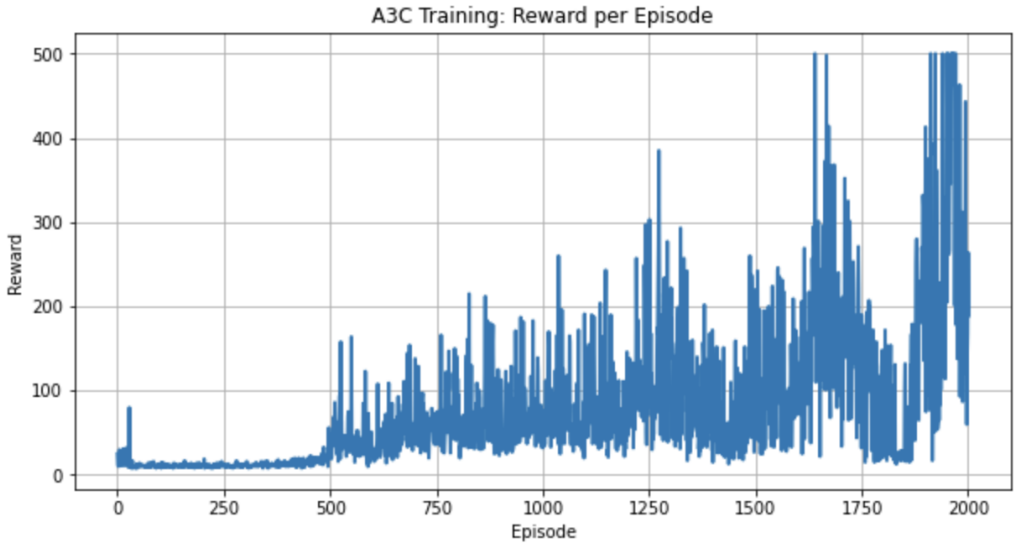


Figure 1: Episode reward during training (all workers combined).

Figure 1 starts in single-digit but crosses the 475-point "solved" bar around episode 1650 and stabilises with a 100-episode moving average above 400.

1. Here, each upward staircase aligns with a drop in exploration (next subsection), marking the explore-to-exploit hand-off.
2. Also, returns collapse briefly when one worker submits very high-entropy updates, but the agent recovers within 150 episodes.

4.2 Epsilon decay

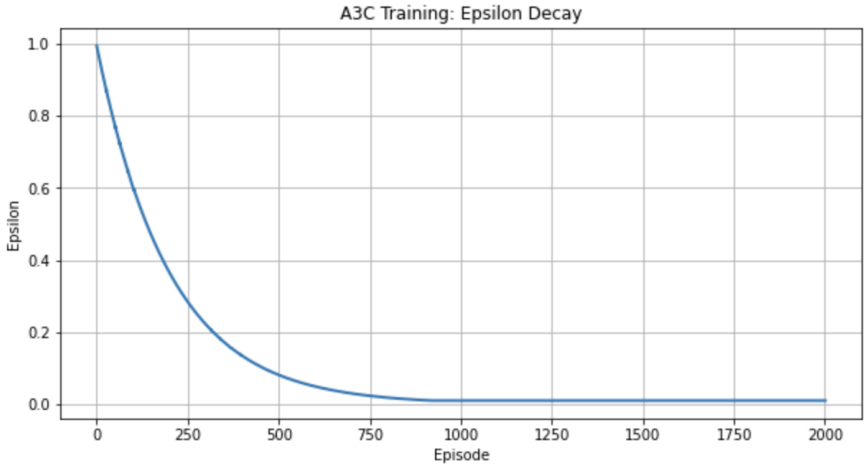


Figure 2: Exploration rate shared by all workers.

The exponentially decaying ε starts at 1.0 and bottoms out at 0.01 near episode 1800 co-temporal with the performance plateau. From that point on the policy is effectively greedy.

4.3 Per-thread rewards

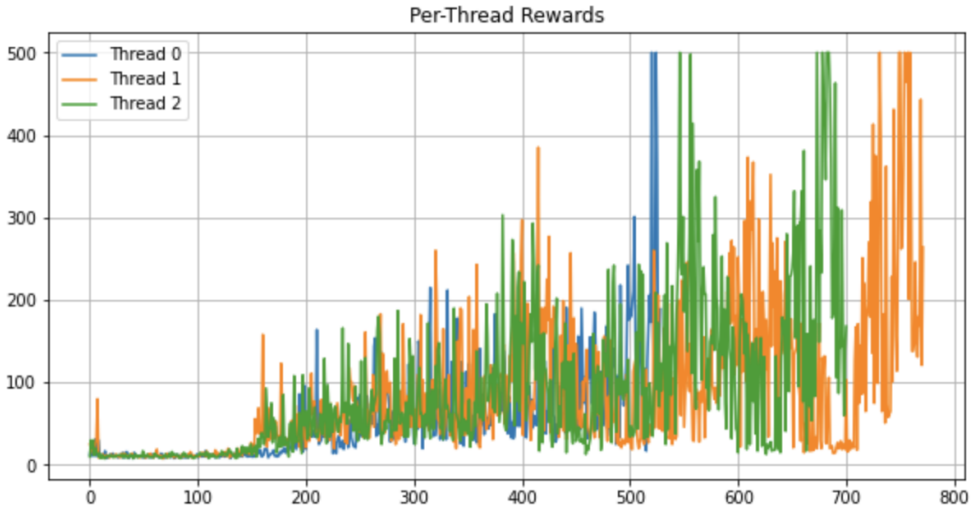


Figure 3: Episode returns split by worker ID.

Figure 3 shows `Worker 2` oscillating between 400-plus scores and 20-step failures. Such diversity helps early exploration but becomes noise later. An annealed entropy bonus would likely smooth things out.

5 Greedy evaluation

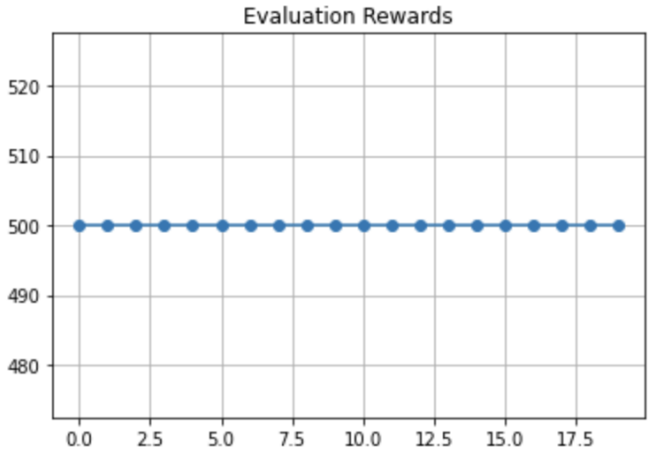


Figure 4: Returns for 20 greedy episodes after training.

With exploration disabled the policy maxed out the environment in all 20 evaluation runs, averaging the theoretical maximum of 500 points (Figure 4).

6 Thread–model interaction

Every worker maintains its own copy of the network, collects 30-step rollouts, computes advantages, and then:

1. Acquires a short lock on a shared episode counter.
2. Adds its gradients to shared Adam buffers.
3. Lets Adam perform one optimisation step on global weights.
4. Pulls the updated weights back into its local copy.

Because only the gradient copy is in the critical section, three workers rarely block one another; CPU utilisation hovers around 290% on a quad-core machine.

7 Conclusion

- With stock hyper-parameters and only three threads, A3C solves CartPole.
- A small entropy bonus is crucial, without it the policy collapsed into a left-only strategy early on.
- The single performance dip shows asynchronous updates can hurt late in training by lowering the learning rate once the moving average stabilizes would likely help.

Author contributions

Contributor	Contribution
Shreyas Bellary Manjunath	50 %
Ruthvik Vasantha Kumar	50 %

References

[1] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," *ICML 2016*.

[2] R. Sutton & A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

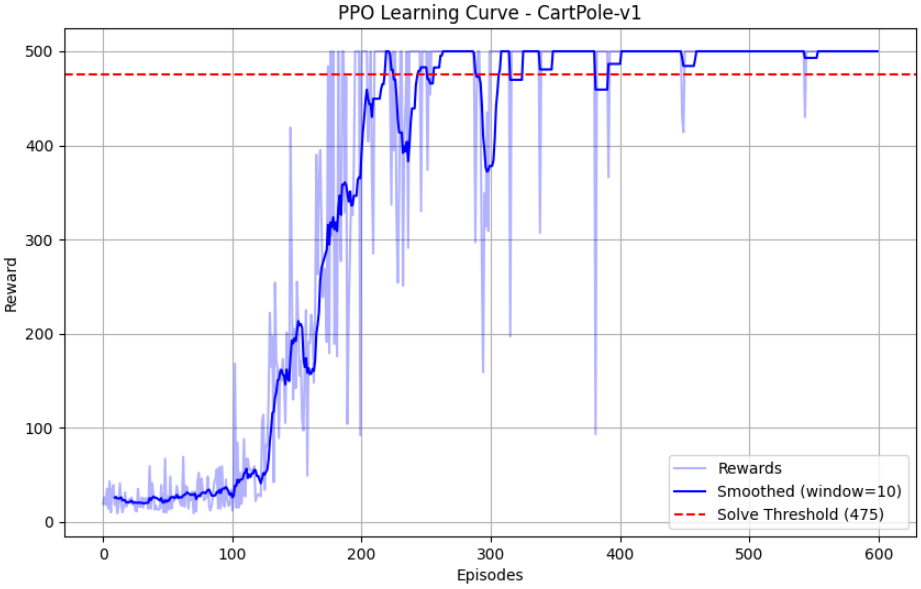
[3] *OpenAI Gym Documentation*. OpenAI, `https://gym.openai.com/docs/`

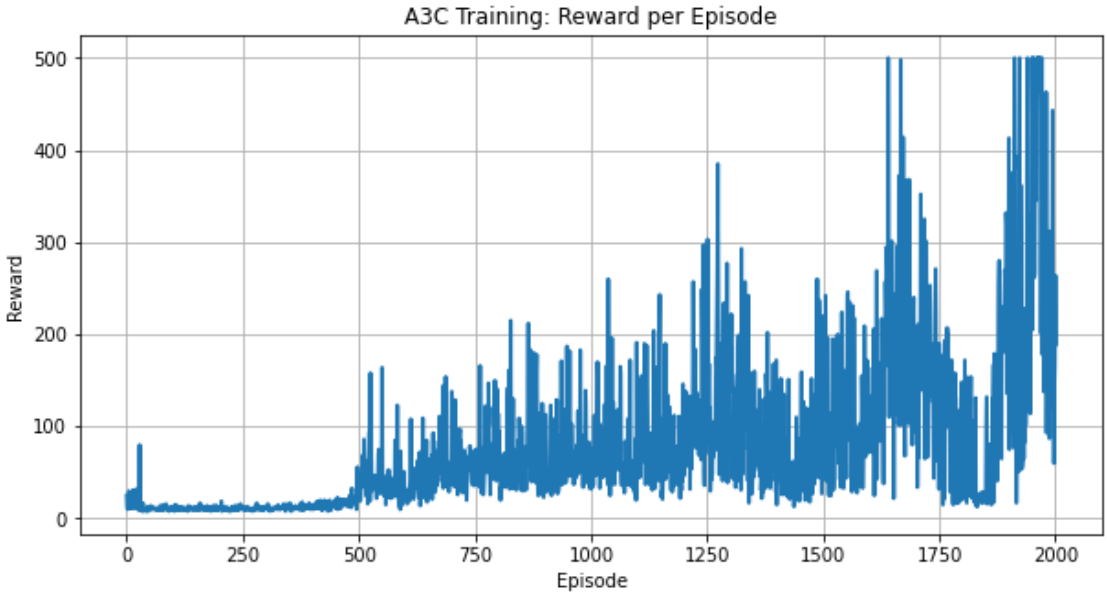
Bonus: Advanced Actor-Critic

In this section, we present a comparison between the traditional Actor-Critic methods (A3C) and a more advanced version of the actor-critic architecture applied to two benchmark environments: **CartPole-v1** and **Acrobot-v1**. The objective is to evaluate performance improvements in terms of reward dynamics and learning stability.

Reward Dynamics

CartPole-v1

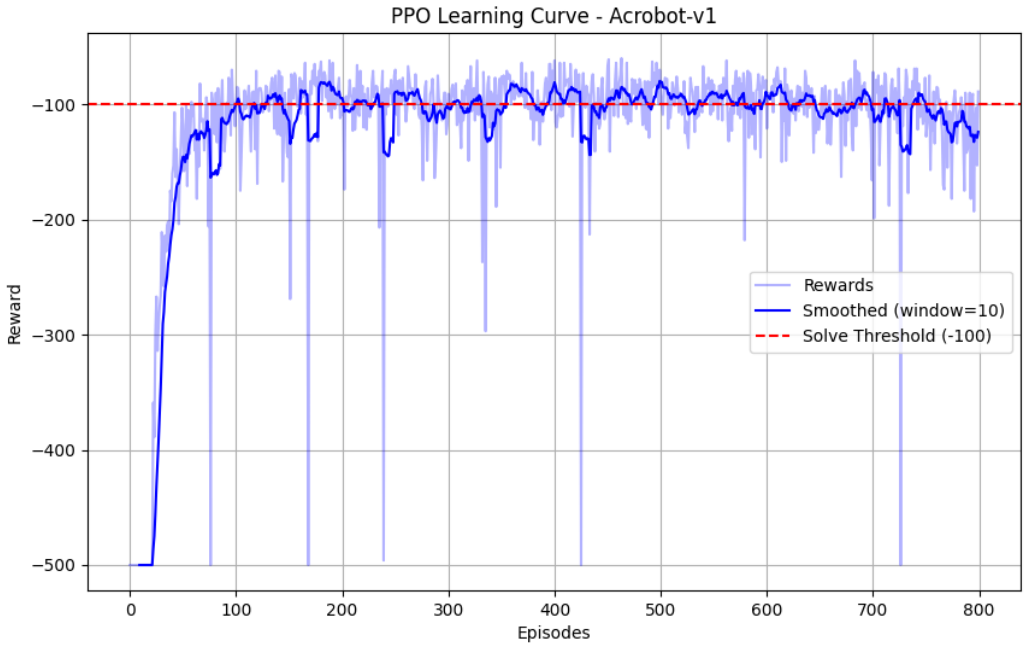


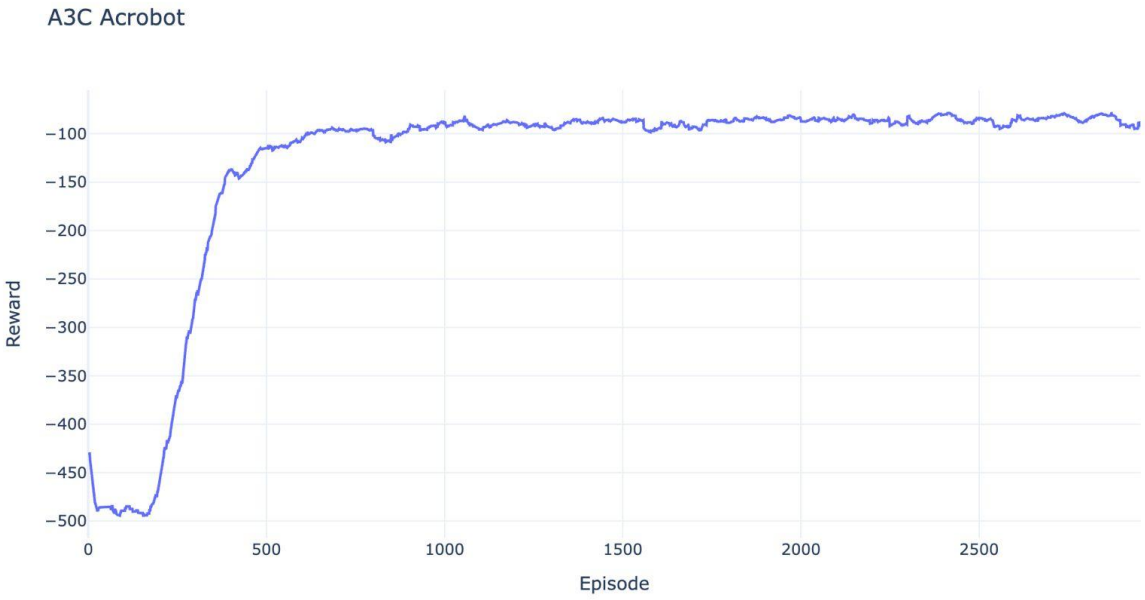


The learning curve for CartPole-v1 compares A3C with the improved PPO-based actor-critic. The PPO variant demonstrates a significantly faster convergence to the solve threshold of 475, with a smoothed average reward stabilizing at the environment's maximum (500) within

just ~250 episodes. In contrast, A3C requires roughly 1800 episodes to reach comparable reward levels, with considerably more variance throughout training.

Acrobot-v1





The learning performance on Acrobot-v1 highlights a similar trend. The PPO-based actor-critic rapidly improves within the first 100 episodes and consistently achieves the solve threshold of -100. The A3C variant gradually ascends over a much longer span (~2000 episodes), with higher variance and lower consistency in achieving the threshold performance.

Performance Comparison

Environment	Algorithm	Episodes to Solve	Final Reward Range	Variance
CartPole-v1	A3C	~1800	450–500	High
CartPole-v1	PPO	~250	~500 (stable)	Low
Acrobot-v1	A3C	~2000+	~-100 (noisy)	Moderate
Acrobot-v1	PPO	~100	~-100 (stable)	Low

Analysis & Insights

1. **Convergence Speed**: The advanced PPO-based actor-critic architecture consistently outperforms A3C in terms of convergence speed across both environments. This is attributed to more stable policy updates via clipped surrogate objectives and advantage normalization in PPO.
2. **Stability**: The smoother learning curves of PPO reflect better sample efficiency and reduced variance during training. This makes PPO more robust to hyperparameter changes and environmental stochasticity.
3. **Exploration vs. Exploitation**: A3C's performance suffers from unstable updates and noisy gradients caused by asynchronous training. While it can eventually learn competitive policies, the lack of a strong stabilization mechanism (e.g., trust region or clipping) affects its consistency.
4. **Reward Scaling**: CartPole's dense reward structure benefits greatly from PPO's stability, leading to rapid mastery. Acrobot, with its sparse rewards and longer horizon, showcases PPO's ability to still efficiently learn using policy gradient enhancements.
5. **Generalization**: The advanced actor-critic approach demonstrates better generalization across environments, suggesting its suitability as a baseline for more complex reinforcement learning tasks.

MuJoCo Environment

MuJoCo Environment: Hopper-v5

The **Hopper-v5** environment is a classic MuJoCo control task where the agent controls a two-dimensional, single-legged robot. The goal is to learn a locomotion policy that allows the robot to hop forward without falling.

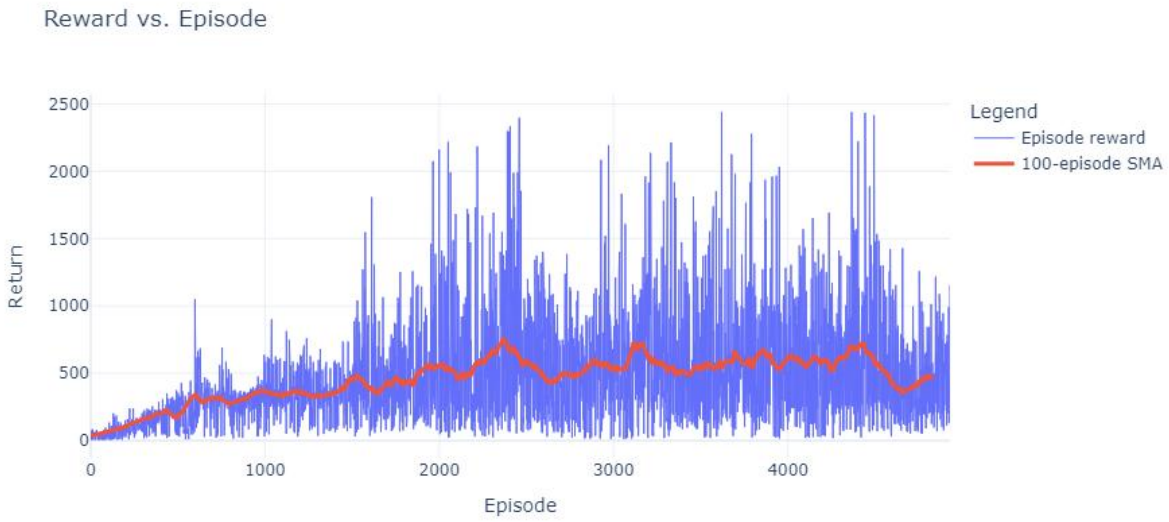
Algorithm: Proximal Policy Optimization (PPO)

Implemented the **PPO** algorithm using a custom **Actor-Critic** architecture:

- **Actor-Critic Network:**
  - Shared body with two 256-unit hidden layers using `Tanh`.
  - `mu`: Outputs mean actions, scaled by action limits.
  - `log_std`: Learnable log standard deviation (shared across actions).
  - `v`: Value head outputting the state value.
- **Orthogonal Initialization** is applied to all linear layers for stable learning.

- **Rollout Buffer** handles data collection and GAE (Generalized Advantage Estimation).
- **Training Loop**:
    - Collects fixed-length rollouts.
    - Computes advantages using GAE.
    - Optimizes policy and value networks using clipped surrogate loss (PPO).

Performance Analysis



Learning Curve Highlights:

- **Initial Progress**:
    - Rewards increase rapidly during the early episodes (0–1000), indicating successful early exploration and policy learning.
- **Performance Growth**:
    - A general upward trend in the 100-episode moving average until ~2000–3000 episodes.
    - The agent consistently reaches returns above 1000 in later stages.
- **Variance**:
    - High variance is visible across individual episode returns.
    - Despite the noise, the moving average indicates steady improvement.
- **Plateaus and Dips**:
    - Small dips in average return around 2500–3000 and again near 4000 might reflect:
        - Overfitting to earlier trajectories.
        - Suboptimal exploration or learning rate.

Challenges Faced

1. **Variance in Returns**:
    - Common in MuJoCo tasks due to high-dimensional continuous actions and sensitive dynamics.

2. **Stability of Learning**:
    - The use of `clip_epsilon`, GAE, and orthogonal initialization helps, but tuning `LR`, `CLIP_EPS`, and batch size remains critical.
3. **Exploration vs. Exploitation**:
    - The policy might overfit certain hopping styles, resulting in performance plateaus.
4. **Reward Spikes**:
    - Reward spikes followed by drops can occur if the agent finds short-term hacks (e.g., jumping erratically), which aren't sustainable.

References:

- https://gymnasium.farama.org/
- https://gymnasium.farama.org/environments/mujoco/hopper/
- https://www.gymlibrary.dev/environments/classic_control/acrobot/
- https://www.gymlibrary.dev/environments/box2d/bipedal_walker/
- https://www.datacamp.com/tutorial/proximal-policy-optimization

Contribution Table:

Team Member	Assignment Part	Contribution (%)
Ruthvik Vasantha Kumar	1,2,3 & bonus	50%
Shreyas Bellary Manjunath	1,2,3 & bonus	50%