

Exploring Reinforcement Learning with Agents

Bekzat Ongdassynov, Dilnaz Sembekova, Yuliya Barko, Assyl Rakhmashev, Yerzhan Yerbatyr

Abstract—In this work, we apply various reinforcement learning methods to the Atari environment Donkey Kong — a unique and challenging setting that has not been thoroughly studied before. We research and compare DQN, A2C and A3C, PPO based on the total rewards earned by the agent and his ability to play in the game. The main purpose of this project is to investigate the unexplored environment and explore what results can we get with standard reinforcement learning algorithms. As a result, the Donkey Kong environment proved to be challenging for the agent due to poorly defined rewards within the game. DQN algorithms failed to learn effectively, while PPO and Actor-Critic methods managed to identify a strategy that maximized short-term rewards but did not lead to winning the game. We also propose future work that could help achieve the desired results.

I. INTRODUCTION

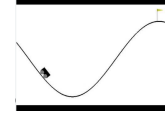
Reinforcement learning (RL) is a machine learning paradigm where agents learn to make sequences of decisions by interacting with an environment to maximize cumulative rewards. It has been successfully applied to complex decision-making tasks, including robotics, game playing, and autonomous systems. This paper investigates the application of RL algorithms in the Atari game environment Donkey Kong, emphasizing the challenges and insights gained from training agents to navigate this unique and demanding setting.

Frameworks like Stable Baselines3 and simulation environments such as OpenAI’s Gym [1] have become pivotal in RL research. Stable Baselines3 provides a collection of state-of-the-art RL algorithms implemented with consistent interfaces, enabling efficient experimentation and evaluation. OpenAI Gym offers standardized environments, including Atari games, to benchmark RL algorithms. The Atari suite offers a challenging testing ground with high-dimensional state spaces, delayed rewards, and complex control dynamics [2].

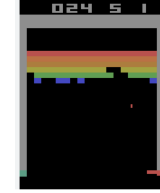
Description	Points
Starting Bonus Value (each Screen)	5000 points
Jumping a barrel or fireball	100 points
Eliminating a Rivet	100 points
Smashing a barrel or Fireball	800 points

TABLE I
SCORE SYSTEM IN DONKEY KONG.

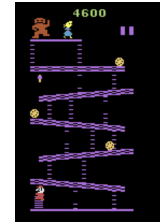
The Donkey Kong environment presents unique challenges for RL agents. The agent controls a character navigating platforms, avoiding obstacles, and climbing ladders to reach higher levels. There are 18 possible actions for the agent in Donkey Kong, including such actions like UP, RIGHT, LEFT, DOWNRIGHT, RIGHTFIRE, etc. Rewards as can be seen in table I are sparse and primarily tied to immediate



(a) Mountain Car



(b) Breakout



(c) Donkey Kong

Fig. 1. Environments used for training

survival or level progression, making it difficult for agents to identify strategies that lead to long-term success. Moreover, poorly defined reward structures can result in suboptimal policies, where agents prioritize short-term gains over overall objectives.

This study evaluates the performance of widely used RL algorithms, including Deep Q-Learning (DQN), Actor-Critic methods (A2C and A3C), and Proximal Policy Optimization (PPO), in the Donkey Kong environment. We focus on their ability to learn effective strategies and maximize rewards while addressing the challenges posed by sparse and delayed rewards. Additionally, DQN was applied to two other environments, Mountain Car and Breakout, following a progression from simple to complex challenges. The experiments began with Mountain Car, the simplest of the three, where the agent learns to build momentum to reach the goal. This provided a foundational understanding of DQN’s capabilities in continuous control tasks. The next step involved Breakout, which features simple game mechanics. Starting with these straightforward games established a solid base before applying RL to the much harder Donkey Kong environment. The results highlight the limitations of traditional methods and explore potential improvements in reward structures and training techniques.

II. METHODS

A. DQN

Deep Q-Learning [3] is a value-based reinforcement learning algorithm that combines the Q-learning approach with deep neural networks, enabling it to handle high-dimensional state spaces. The Q-learning algorithm estimates the action-value function $Q(s, a)$, which represents the expected cumulative reward starting from state s , taking action a , and following an optimal policy after that. However, because it relies on a tabular representation, traditional Q-learning performs poorly in settings with huge state-action spaces. DQN addresses this by using a neural network to approximate $Q(s, a; \theta)$, where θ represents the network's parameters.

To stabilize training and improve performance, DQN introduces several innovations:

- 1) Experience Replay: A replay buffer stores past transitions, (s, a, r, s') , which are sampled randomly during training to break correlations between consecutive updates and improve data efficiency.
- 2) Target Network: A separate target network, $Q(s, a; \theta^-)$, with frozen parameters θ^- , is updated periodically to provide stable targets for the Q-value updates. This mitigates the instability caused by rapidly changing Q-value estimates.

The DQN training objective minimizes the temporal difference (TD) error [3]:

$$L(\theta) = E_{(s, a, r, s') \sim D} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$$

where γ is the discount factor, and D is the replay buffer.

Despite its success, DQN suffers from overestimation bias when $\max_{a'} Q(s', a')$ is used, as the same network selects and evaluates actions. This leads to overly optimistic Q-value estimates.

B. Actor-Critic

The Actor-Critic method [4] is a class of model-free reinforcement learning algorithms that combines value-based and policy-based learning, aiming to take advantage of the strengths of both approaches. The actor part learns the policy $\pi_\theta(a|s)$, where θ represents the parameters of the policy, which usually represented as neural network. The critic part evaluates the actions taken by the actor by estimating the action-value function $Q_w(s, a)$, where w represents the parameters of the value function, which computes the expected return (reward) from a given state or state-action pair. Asynchronous Advantage Actor-Critic [5], A3C, is an improvement of the standard Actor-Critic with a special focus on parallel training, where there are multiple actors that are trained in parallel and interact with independent copies of the environment, each of them has their own gradients and they sync them with the global model from time to time. A2C [5] on the other hand is a synchronized version of A3C, where there is a global coordinator that waits for all parallel actors to finish and only then updates the global model, and on the next iteration, each actor will have the same policy. The loss function looks like:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{actor}} + \lambda \mathcal{L}_{\text{critic}} - \beta H(\pi)$$

Where:

$$\mathcal{L}_{\text{actor}} = -E[\log \pi(a_t|s_t) A_t]$$

$$\mathcal{L}_{\text{critic}} = E[(R_t - V(s_t))^2]$$

$$H(\pi) = - \sum_a \pi(a|s) \log \pi(a|s)$$

A_t is the advantage function that represents the difference between the action-value Q_t and the state-value $V(s_t)$, R_t is the return that is the discounted sum of future rewards, and $V(s_t)$ is the estimated value of state s_t under the current policy, $H(\pi)$ is the entropy that measures the uncertainty or randomness in the policy $\pi(a|s)$, and β controls the trade-off between exploration and exploitation by scaling the entropy term.

C. PPO

Proximal Policy Optimization (PPO) is a state-of-the-art reinforcement learning algorithm that builds on the strengths of policy gradient methods. It combines elements of two established techniques: the multi-worker parallelism of Advantage Actor-Critic (A2C) and the trust region constraint approach of Trust Region Policy Optimization (TRPO) [6]. PPO aims to improve the agent's training by avoiding policy updates that are too large. This is achieved by introducing a clipped surrogate objective function that constrains the policy change within a small range using a clip [7]. This approach ensures that policy updates are both conservative and efficient, striking a balance between exploration and exploitation.

There are two key strategies:

- 1) Clipped Objective Function: The objective function of PPO prevents large updates to the policy, which can lead to catastrophic performance drops. The clipping mechanism ensures that the policy ratio—the ratio between the new policy and the old policy probabilities—stays within a predefined range. The clipped surrogate objective function is defined as:

$$L^{\text{CLIP}}(\theta) = E_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \quad (1)$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$: The probability ratio of the current policy to the old policy for action a_t at state s_t .
- $\hat{A}_t$: The advantage estimate at time t .
- ϵ : A hyperparameter that defines the clipping range.

This mechanism prevents excessive policy updates that could destabilize learning while allowing for sufficient exploration of the environment.

- 2) Advantage Estimation: PPO uses Generalized Advantage Estimation (GAE) to calculate the advantage function, which provides a more stable and lower-variance estimate of the advantage. This facilitates efficient policy updates while maintaining stability [8].

In this project, PPO was implemented using the Stable-Baselines3 library, which provides robust and efficient reinforcement learning frameworks. The policy and value networks were parameterized by deep neural networks and trained concurrently via stochastic gradient descent. Key hyperparameters, including the clipping threshold (ϵ), learning rate, and number of epochs per update, were fine-tuned to enhance performance.

III. RESULTS

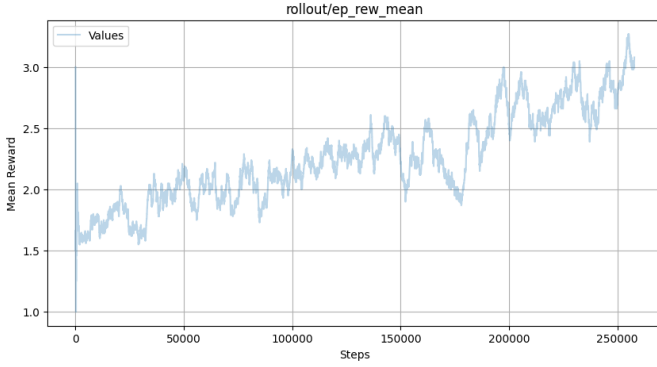
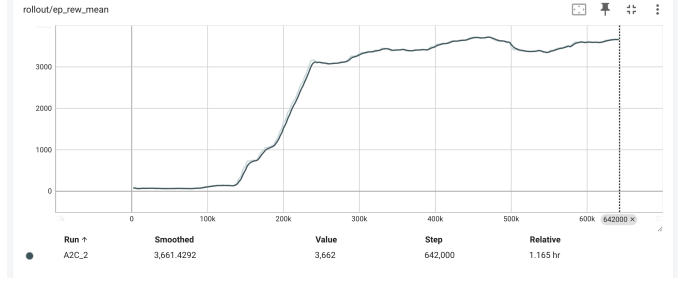


Fig. 2. Progression of rollout/ep_rew_mean for DQN during training on Breakout.

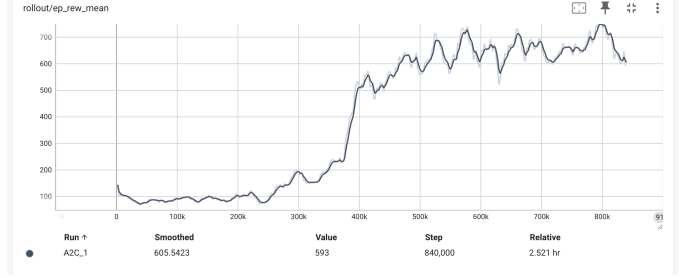
A. DQN

The application of DQN began with the simplest environment, Mountain Car. This environment provided an opportunity to understand how the algorithm operates, focusing on how the agent learns to build momentum to reach the goal by exploiting the physics of the environment. After gaining insights from Mountain Car, DQN was applied to a more complex game, Breakout. The agent initially struggled, frequently missing the ball, but after two hours of training, it successfully learned to aim and break more than 50 blocks without losing, demonstrating significant improvement in playing game. Figure 2 highlights the improvement of the DQN agent in Breakout, showing significant performance gains.

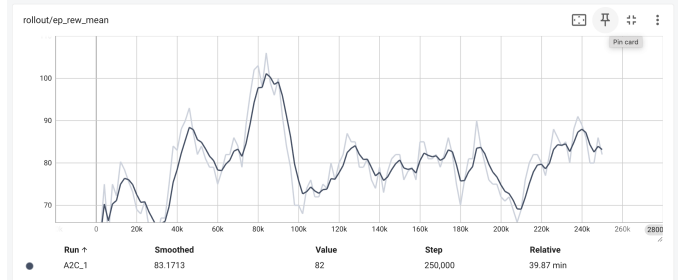
Having achieved promising results in Breakout, the focus shifted to more challenging environment of Donkey Kong. We experimented with different setups, such as varying buffer sizes, training epochs, and other hyperparameters, while training the model in the Donkey Kong environment. The final agent performed below expectations in spite of these efforts. The agent specifically learned to jump over obstacles on a regular basis while staying on the first floor of the environment. In 1306478 timesteps agent received 1.87 episodic mean reward. Such a low reward might indicate that Mario avoids exploring potentially higher-reward areas in favor of safer, low-risk



(a) entropy coefficient=0.01



(b) entropy coefficient=0.1



(c) entropy coefficient=0.5

Fig. 3. TensorBoard logs of A2C for rollout/ep_rew_mean with varying entropy coefficients.

actions. In fact, we observed that this behavior developed when the agent realized that, despite failing to advance through the levels or meet higher overall goals, remaining on the first floor maximized its immediate rewards.

B. Actor-Critic

As shown in Figure 3, the results demonstrate the rollout/ep_rew_mean which corresponds to the non-clipped mean episodic reward observed during training for different entropy coefficients. As a rule, the higher the entropy coefficient, the greater the exploration encouraged by the agent, leading to more random action selection. This can delay convergence to a high mean episodic reward as the agent prioritizes exploring diverse strategies. Initially, the entropy coefficient was set to 0.01. Over time, Mario learned to maximize the reward, even reaching a value of 3662 per episode.

However, upon visual inspection, it became evident that, similar to the agent trained with DQN, Mario tended to stay in one place, jumping over obstacles and receiving rewards for this behavior. The key difference, however, was that Mario

was more precise and earned significantly more reward in less training time.

To enforce exploration, the entropy coefficient was increased to 0.1. With this setting, the agent was able to move right, where he could find the stairs. However, the mean reward stabilized around 600, with little improvement despite continued training, as shown in Figure 4. When the entropy coefficient was further increased to 0.5 (as shown in part (c)), it became apparent early on that the agent exhibited highly erratic behavior. We decided not to continue training with this coefficient, hypothesizing that Mario would not learn anything meaningful under these conditions.

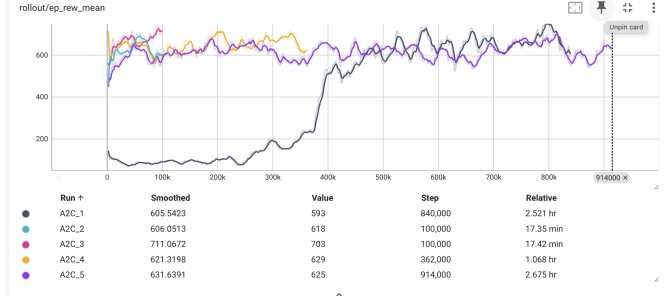


Fig. 4. TensorBoard logs of A2C for rollout/ep_rew_mean with entropy coefficients equal to 0.1. In A2C_i, each i corresponds to the i th training of the same model.

After many training steps, as shown in Figure 4, we were unable to improve Mario’s mean episodic reward. Additionally, upon visual inspection, he was not able to climb the stairs. The results for A2C is presented since A3C’s behavior was the same.

C. PPO

PPO agents were trained under three configurations: a fixed learning rate for 1M steps and linear decay schedules for 2M and 4M steps. Agents trained with the learning rate schedule demonstrated smoother training curves and better overall performance compared to the fixed-rate configuration. The agent trained for 2M steps achieved the highest rewards, with minimal improvements observed at 4M steps, indicating a plateau in learning. The PPO agent exhibited similar cumulative rewards to DQN and Actor-Critic methods. It effectively learned to dodge barrels by jumping over them with high accuracy, demonstrating its ability to align behavior with reward signals. Additionally, PPO encouraged exploratory behavior, with agent frequently moving across platform and occasionally approaching ladders. However, the agent failed to climb ladders or progress to higher levels.

Figures 5 and 6 show the mean rewards and learning rates during training for the three configurations. As shown in Figure 5, the agent trained with a linear decay schedule for 2M steps achieved significantly higher cumulative rewards compared to the fixed-rate configuration. Additionally, the learning rate dynamics (Figure 6) indicate a smoother and more consistent adjustment over time for the agents using the

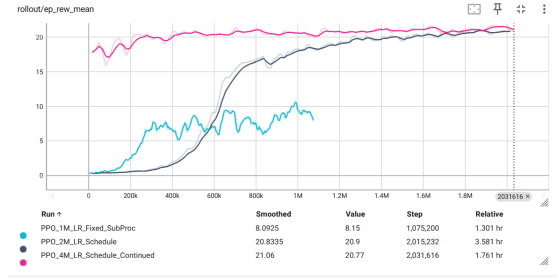


Fig. 5. Mean Rewards During Training.

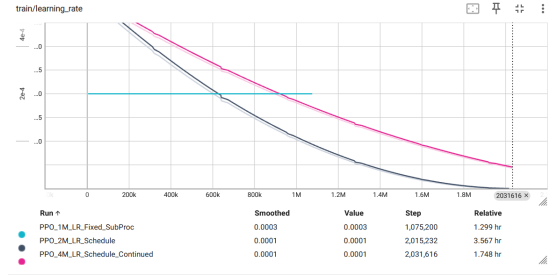


Fig. 6. Learning Rate During Training.

learning rate schedule, whereas the fixed-rate agent showed more fluctuations in its training behavior.

Attempts to improve ladder-climbing through custom rewards, such as incentivizing proximity and jumping near ladders, resulted in the agent exploiting rewards by repeatedly jumping in place without advancing. Similarly, increasing entropy to encourage exploration led to erratic movement without meaningful progress. This highlights PPO’s capability to adapt to complex environments but underscores a common challenge in reinforcement learning: designing reward structures that effectively guide agents toward desired long-term goals. Future work could explore advanced reward engineering and exploration techniques to overcome these obstacles.

IV. DISCUSSION

Interpreting results in reinforcement learning tasks can be challenging, and an environment as complex as Donkey Kong is no exception. All agents managed to learn to dodge barrels by jumping over them. The next step was to ascend using the ladder, and even though all agents failed, some showed more activity around the ladder, trying to find a way to interact with it. For instance, a relatively simpler DQN produced an agent learned in most cases to dodge the occasional barrel, but not so accurate as Actor-Critic methods and PPO even with more time spent on training. There could be several reasons why the agents did not learn climbing.

Atari games have a specific action space from which the agent chooses the action, in the case of the Donkey Kong environment the action space is particularly large consisting of 18 actions. It meant that during training agents wasted time and computation experimenting with different actions.

They could eventually get discouraged from interacting with a ladder.

Climbing proved to be a harder task for reinforcement learning agents to learn due to sparse rewards and the need for precise state-action coordination [9]. Reward system lacked detail and was not intended for agent training. It did not reward for gaining elevation or approaching the princess. The only logical way for the agent to 'realize' its goal of climbing all the way up was for it to climb all the way up at least once without getting hit. Probably no amount of training would help with such a reward system since there was no incremental feedback.

We tried to alter rewards, but it never ended up helping in the way we expected. Agents do not follow our motivations, they just come up with behavior as a response. Since the agent lacked a natural incentive to climb the ladder, we introduced a reward for the action of moving upward. However, rather than learning to climb effectively, the agent discovered that jumping repeatedly maximized rewards and thus focused solely on this behavior.

Agents in RL have to be pushed the right direction, and engineering the reward system is tricky and does not necessarily motivate the agent the way we expect.

Entropy implementation alone did not show improvements since rewards were still sparse (see Figure 3). However, observing the agent with greater inclination towards exploration as opposed to exploitation, we still recommend using entropy with later takeover of the latter.

Another key challenge in training deep RL models is how long it takes for agents to learn meaningful patterns.

V. CONCLUSION

In our study, we applied DQN, A2C, and PPO to a Donkey Kong environment. Our findings highlight the limitations of the reward structure in sparse and complex environments. DQN struggled to learn effective policies, while A2C and PPO showed better adaptability but still failed to achieve a long-term objective of an agent climbing ladders.

The results emphasize the importance of careful reward engineering and exploration techniques. Sparse rewards hindered the agents' ability to associate actions with long-term success, while attempts to modify the reward system led to unintended behaviors, such as agents exploiting short-term rewards.

While PPO outperformed other methods in terms of cumulative reward, none of the algorithms achieved the desired level of competence in the environment. These findings underscore the need for advanced methods, such as hierarchical reinforcement learning or behavior cloning.

VI. FUTURE WORK

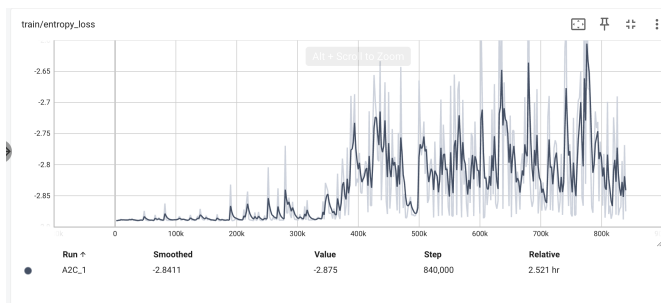
Based on our experiments with entropy adjustments, reward modifications, and the exploration of alternative methods to encourage the agent to climb stairs, we identified behavior cloning as a potential solution. Behavior Cloning (BC) [10] is a supervised learning technique in reinforcement learning (RL) where the agent learns to mimic the behavior of an expert

by observing demonstrations. The agent trains directly on the state-action pairs provided by the expert's policy, bypassing the need for an explicit reward signal. As a result, the policy network is learned through imitation. In our case, we began by collecting a dataset of demonstrations, where we manually played the game and recorded each (state, action) pair for every action taken. The objective is to gather a larger dataset and analyze the outcomes to evaluate the effectiveness of behavior cloning in this context.

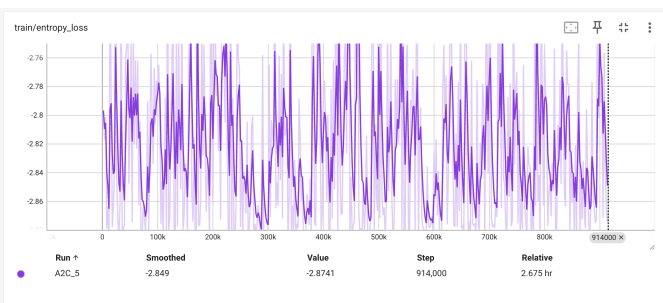
REFERENCES

- [1] S. Ravichandiran, *Hands-on reinforcement learning with Python: master reinforcement and deep reinforcement learning using OpenAI gym and tensorFlow*. Packt Publishing Ltd, 2018.
- [2] V. Mnih, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [3] V. Mnih, K. Kavukcuoglu, D. Silver, and et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
- [4] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 834–846, 1983.
- [5] V. Mnih, "Asynchronous methods for deep reinforcement learning," *arXiv preprint arXiv:1602.01783*, 2016.
- [6] Stable-Baselines3, "Ppo documentation," <https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html>, accessed: December 7, 2024.
- [7] Hugging Face, "Clipped surrogate objective," <https://huggingface.co/learn/deep-rl-course/unit8/clipped-surrogate-objective>, accessed: December 7, 2024.
- [8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint*, vol. abs/1707.06347, 2017. [Online]. Available: <https://arxiv.org/abs/1707.06347>
- [9] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explora: A new approach for hard-exploration problems," *Journal of Artificial Intelligence Research*, vol. 72, pp. 1–38, 2021.
- [10] M. Bain and C. Sammut, "A comparison of reinforcement learning and monte carlo tree search for game playing," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999, pp. 1038–1043.

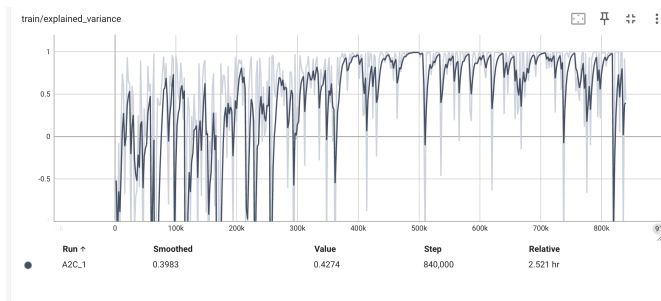
VII. APPENDIX



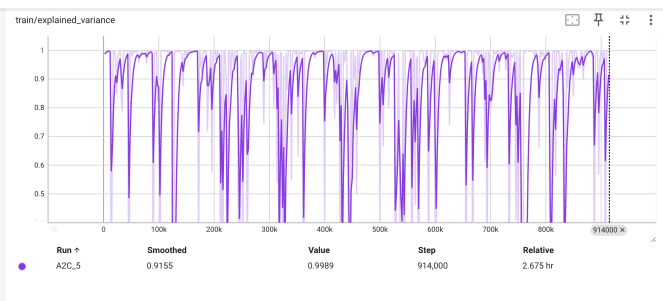
(a)



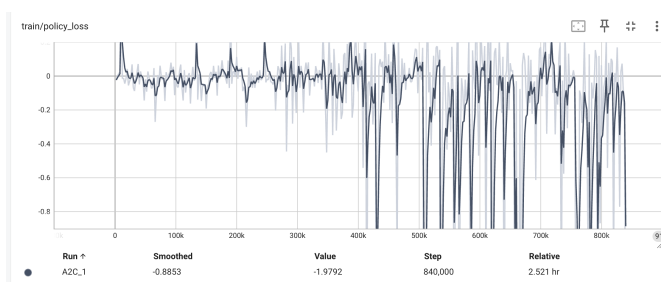
(b)



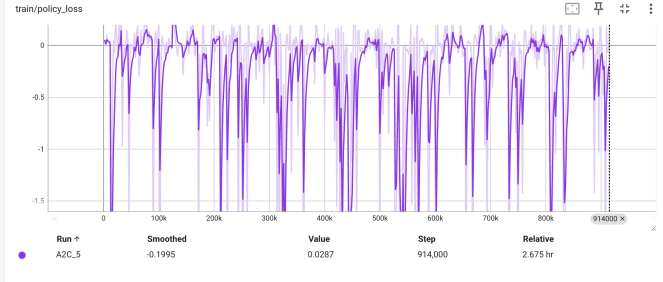
(c)



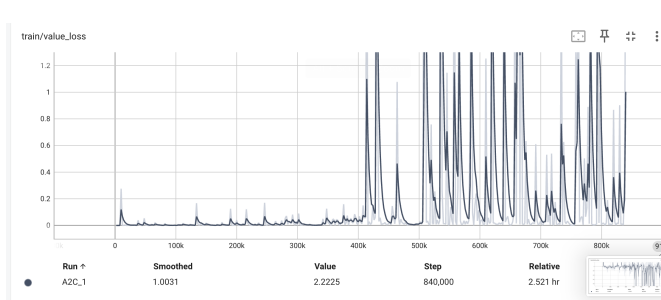
(d)



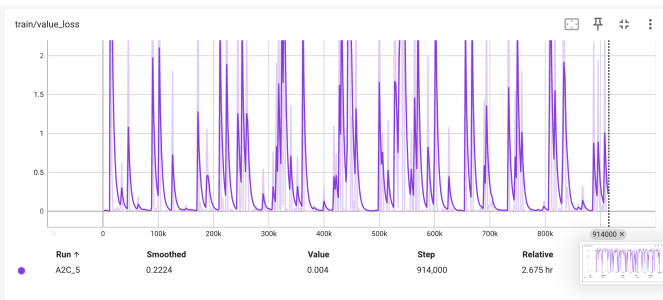
(e)



(f)



(g)



(h)

Fig. 7. The figure presents the results of training A2C, comparing the first training run (on the left) and the final training run (on the right). The color corresponds to the ones seen in Figure 4. (a,b) shows entropy loss, (c,d) represents explained variance, (e,f) shows policy loss, and (g,h) displays value loss.