Evaluating Deep Reinforcement Learning Algorithms under Resource Constraints: A Case Study on SpaceInvaders

1st Yi
Industrial Engineering
University of Michigan
Ann Arbor, USA
taoyi@umich.edu

***Abstract*—Reinforcement Learning (RL) algorithms often rely on large computational budgets to achieve high performance, limiting their practicality in constrained environments. This paper investigates the efficiency of three popular RL algorithms—DQN, PPO, and QR-DQN—under a strict resource constraint of 50,000 training steps. Using the Atari game SpaceInvaders as a testbed, we compare each algorithm's learning progress, stability, and final performance. Our results show that DQN achieves the highest and most stable reward, while QR-DQN exhibits fast early learning but with significant variance. PPO fails to maintain performance due to its on-policy nature. We also discuss the challenges faced in environment setup and propose future work toward building a custom claw machine simulator using the same lightweight pipeline.**

***Index Terms*—Reinforcement Learning, SpaceInvaders, AI**

I. Introduction

Reinforcement Learning (RL) has become a widely adopted method for training agents in complex decision-making tasks. A classic benchmark is the Atari 2600 environment, where agents learn to optimize rewards from visual input. Deep Q-Networks (DQN) [1] and Proximal Policy Optimization (PPO) [2] have shown impressive results across many Atari games.

However, most prior work assumes access to extensive computational resources, often training for millions of timesteps. In contrast, this paper investigates the performance of three RL algorithms—DQN, PPO, and Quantile Regression DQN (QR-DQN) [3]—under a strict constraint of 50,000 timesteps, using the Atari game *SpaceInvadersNoFrameskip-v4* as a case study. We aim to evaluate which algorithm best balances learning efficiency and performance in low-resource environments.

Initial attempts to train on local machines failed due to the absence of a dedicated GPU. Virtual GPU services also proved problematic, as modern container images (e.g., CUDA 12+, Python 3.11+) were incompatible with legacy libraries required by Stable Baselines3 (e.g., ALE-py 0.8, Gym 0.21). These challenges led us to rely on Google Colab, which, despite its resource limitations, offered a stable environment for running lightweight experiments.

II. Methodology

A. Problem Formulation

The task is formulated as a Markov Decision Process (MDP) with 84×84 grayscale observations, a 2-frame stack, and a discrete action space of six actions. Rewards are sparse and game-specific.

B. Environment

We use the `SpaceInvadersNoFrameskip-v4` environment from the Gymnasium Atari suite. Preprocessing includes grayscale conversion and frame-stacking. All training was conducted on Google Colab with 12GB RAM.

C. Algorithms

- **DQN**: Off-policy learning with experience replay and target networks.
- **PPO**: On-policy actor-critic algorithm with generalized advantage estimation (GAE) and clipping.
- **QR-DQN**: A distributional variant of DQN that predicts quantiles of the return distribution.

D. Training Setup

Each model was trained for 50,000 steps using a shared pipeline:

- `DummyVecEnv` with `VecFrameStack` (n_stack=2)
- Learning rates: DQN = 1e-4, PPO = 2.5e-4, QR-DQN = 5e-5
- Replay buffer for DQN/QR-DQN = 20,000 steps
- Evaluation every 5,000 steps over 3 episodes (mean reward)

III. Results

Table I shows peak performance metrics for each algorithm.

QR-DQN showed rapid initial improvement but fluctuated, even dropping to zero at 35,000 steps. DQN outperformed all others in both reward and stability. PPO underperformed due to on-policy inefficiency in low-step training.

TABLE I
BEST MEAN REWARD ACROSS TRAINING

Algorithm	Best Reward	Step	Notes
DQN	535	35k	Fast, stable convergence
QR-DQN	285	20k–45k	Early peak, unstable afterwards
PPO	105	20k	Policy collapse after early gain

IV. LIMITATIONS AND FUTURE WORK

One major limitation was the difficulty in setting up a compatible environment on personal or virtual machines. Much time was lost troubleshooting CUDA and Python incompatibilities, which limited training time and forced reliance on Colab. While Colab's limitations constrained PPO performance, the experience provided valuable lessons in cloud-based RL setups and environment configuration.

In the future, I plan to develop a custom simulation of a claw machine game using the same RL pipeline. The goal is to design a lightweight environment where an agent learns to control a robotic claw to pick up virtual objects—extending RL from Atari benchmarks to interactive, real-world-inspired tasks.

Fig. 1. DQN STEP

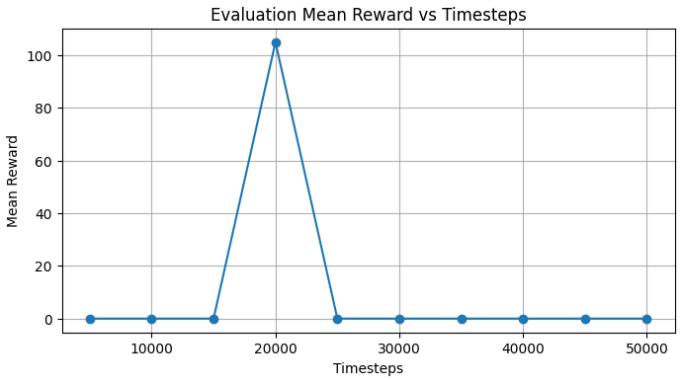

Fig. 2. PPO STEP

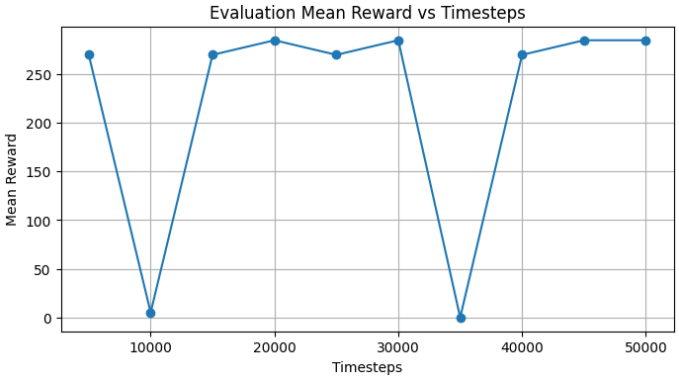

Fig. 3. QRDQN STEP

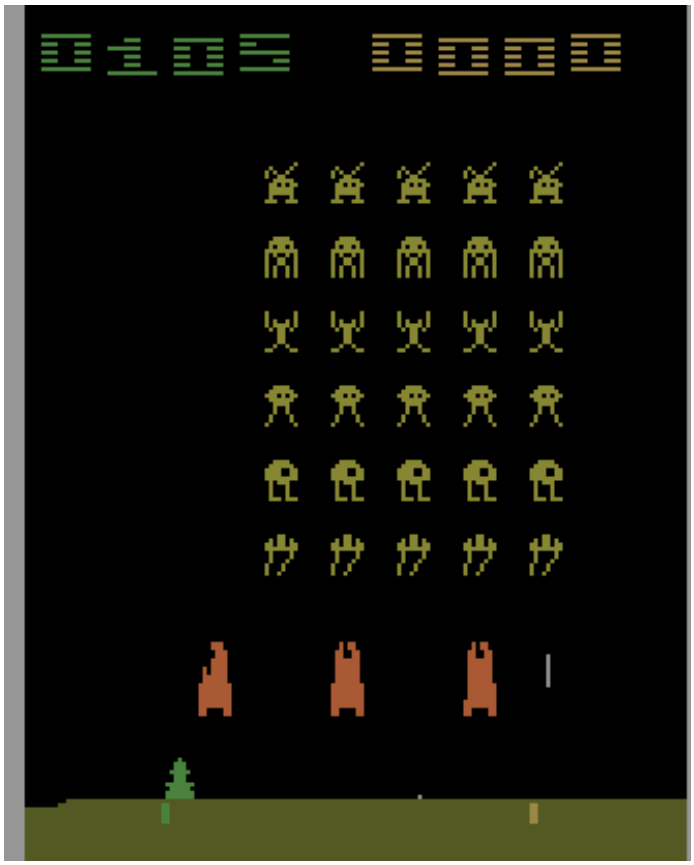

Fig. 4. a cut of the best DQN model

REFERENCES

[1] V. Mnih, et al., "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.

[2] J. Schulman, et al., "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347, 2017.

[3] W. Dabney, et al., "Distributional Reinforcement Learning with Quantile Regression," arXiv preprint arXiv:1710.10044, 2018.