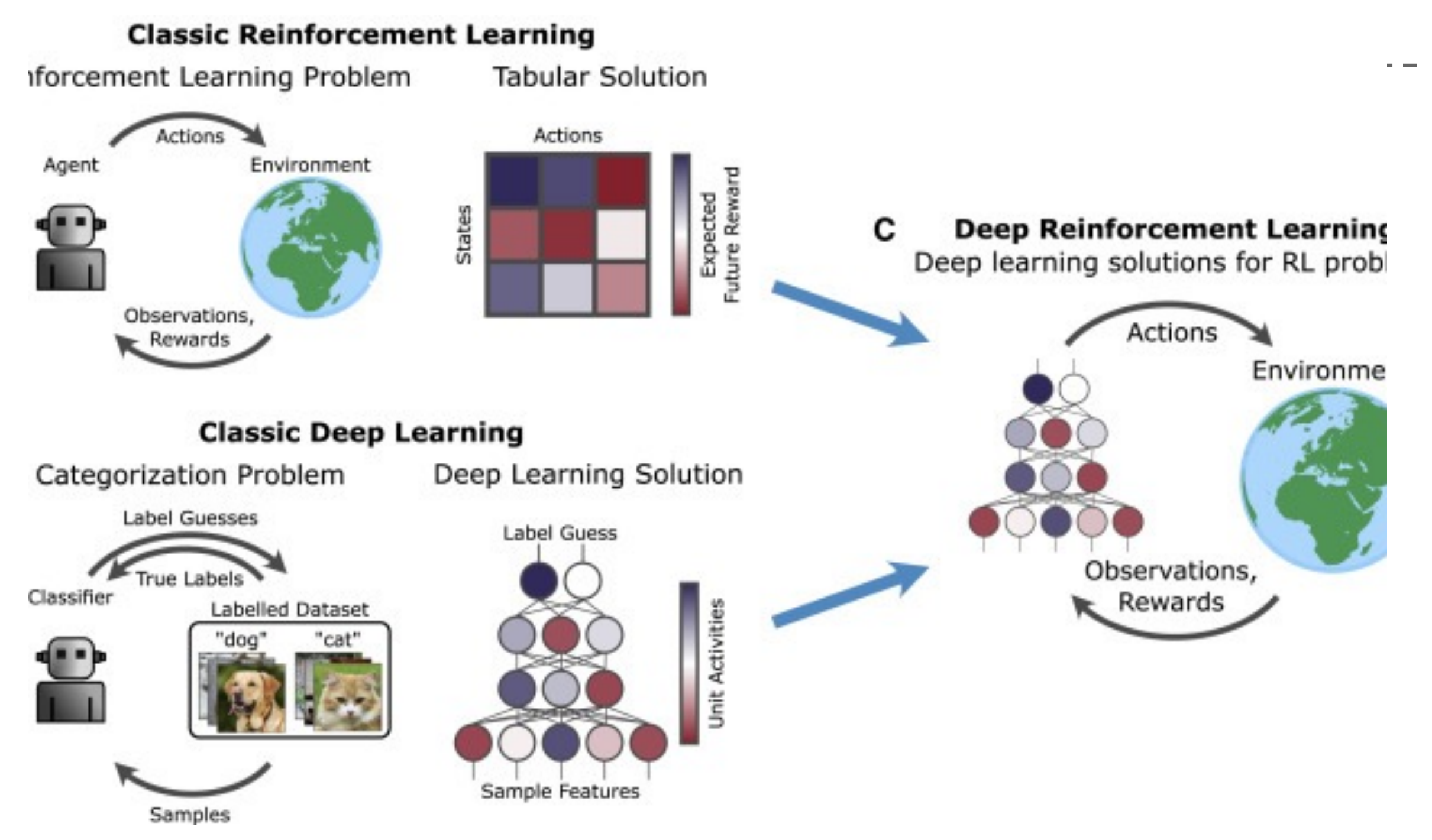
SOLVING FLATLAND ENVIRONMENT WITH PARALLEL AND SEQUENTIAL REINFORMENT LEARNING

Aryan Saini , Divyansh Chopra, Pragati Nagar and Alina Vereshchaka

Introduction

Reinforcement Learning (RL) is a decision-making science. It's all about learning the optimal action in an environment in order to maximize reward. This ideal behavior is acquired by encounters with the environment and observations of how it responds, similar to how dog would receive a treat(reward) for following the command(optimal action) of its owner.

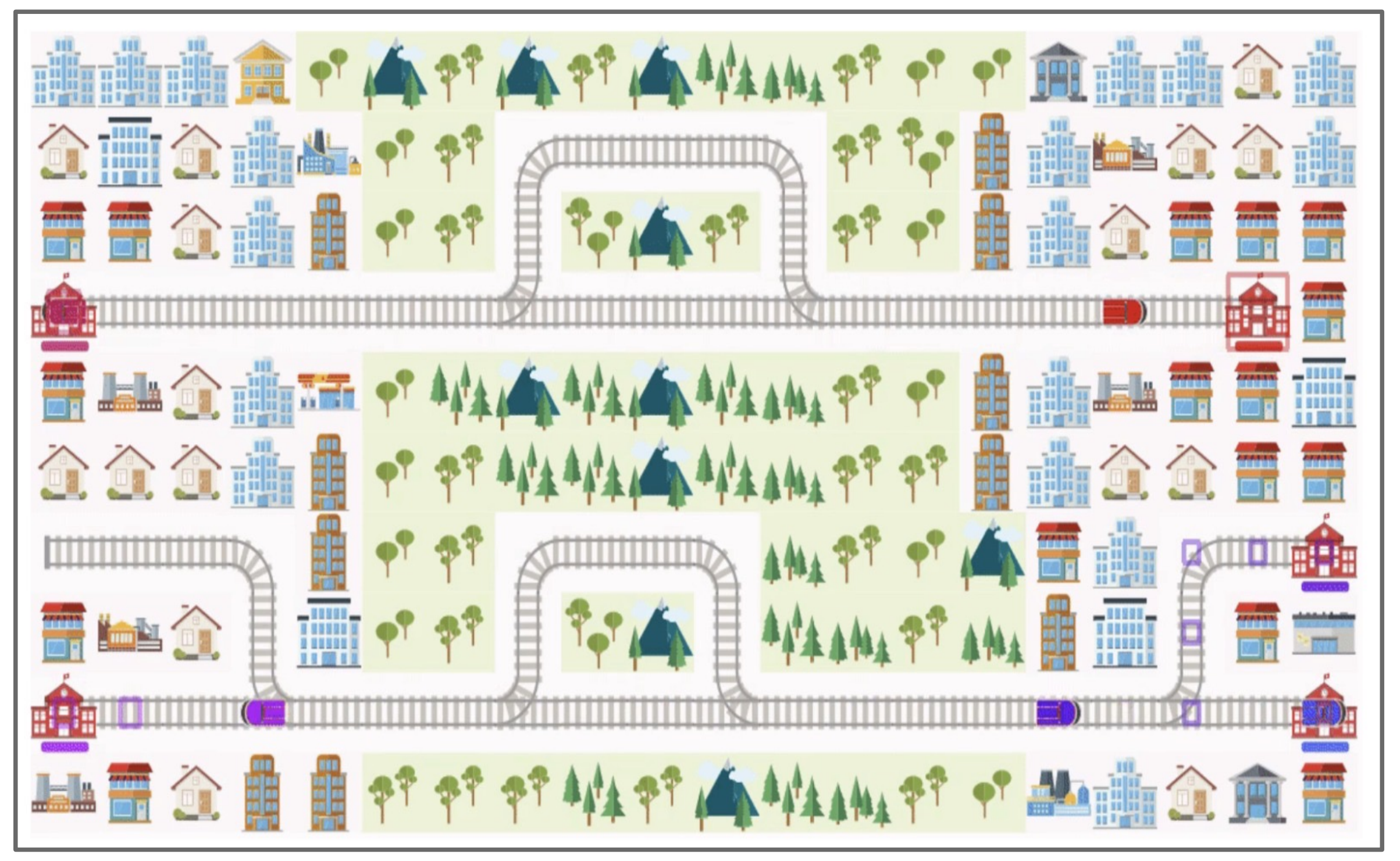


Deep Reinforcement Learning

The combination of Reinforcement Learning and Deep Learning is Deep Reinforcement Learning, a rapidly developing discipline. It's also the most popular sort of Machine Learning since it can handle a wide range of complicated decision-making tasks that were before unsolvable by a machine with human-like intelligence.

Flatland Environment

Flatland is a 2D rectangular grid environment with the smallest unit is a cell. Each cell can hold a single agent (train). Agent can have different orientation. Agents can travel only in the direction they are facing currently. So, the transition for the agent depends both on its position and direction. Transition maps define the railway network.



Double Deep Q-Network

Double DQN employs two neural network that are identical. One learns during the experience replay, and the other is a clone of the first model's final episode. The second model is used in the evaluation of the Q-value.

$$R_t + \gamma Q^c(s_{t+1}.\operatorname{argmax} Q(s_{t+1}.a'))$$

Algorithm 1: Double DQN (Hasset et al. 2015

```
Initialize primary network $Q_\theta$, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau < 1$, $C$
for each iteration do
    for each step do
        Observe state $s_t$ and select $a_t \sim \pi(s_t)$
        Execute action $a_t$ and observe next state $s_{t+1}$ and reward $r_t$;
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$.
    end
    for each update step do
        Sample minibatch of transitions $(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
        Compute target Q value:
            $Q^*(s_t, a_t) = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$
         Perform a gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$ with
         respect to the primary network parameters $\theta$
        Every $C$ steps update target network parameters:
            $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
    end
end
```

Advantage Actor Critic

The A2C algorithm combines two types of Reinforcement Learning algorithms (Policy Based and Value Based). Policy-based agents translate input states to output actions by learning a policy. Value-based algorithms learn to choose actions based on the expected value of the input state or action.

Advantage Function :-

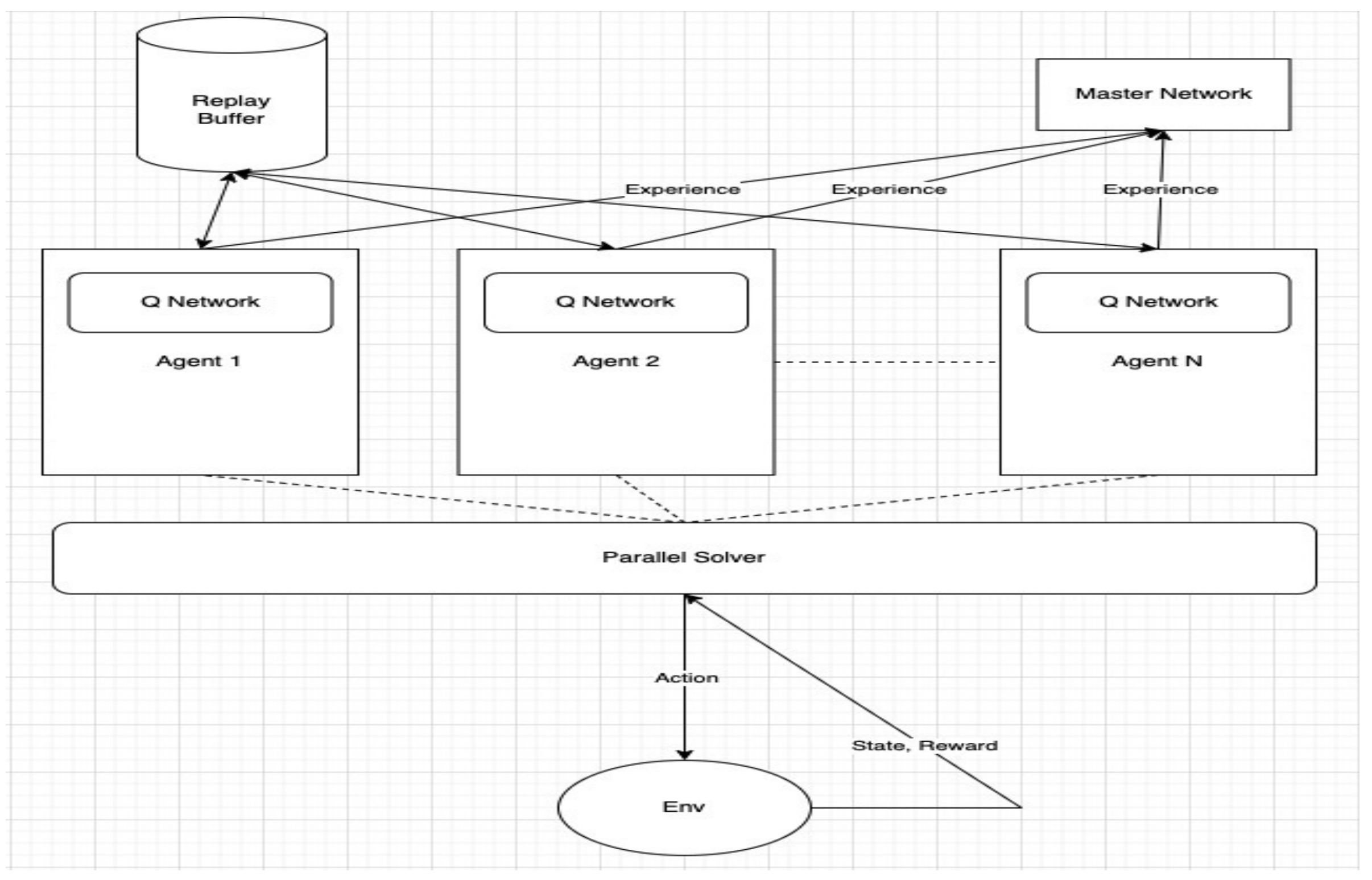
$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\delta_{\pi_\theta}|s,a] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s')|s,a\right] - V_{\pi_\theta}(s)\\ &= Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)\\ &= A_{\pi_\theta}(s,a)\end{aligned}$$

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
   Initialize $S$ (first state of episode)
   $I \leftarrow 1$
   Loop while $S$ is not terminal (for each time step):
      $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
      Take action $A$, observe $S', R$
      $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
      $\boxed{\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})}$
      $\boxed{\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})}$
      $I \leftarrow \gamma I$
      $S \leftarrow S'$
```

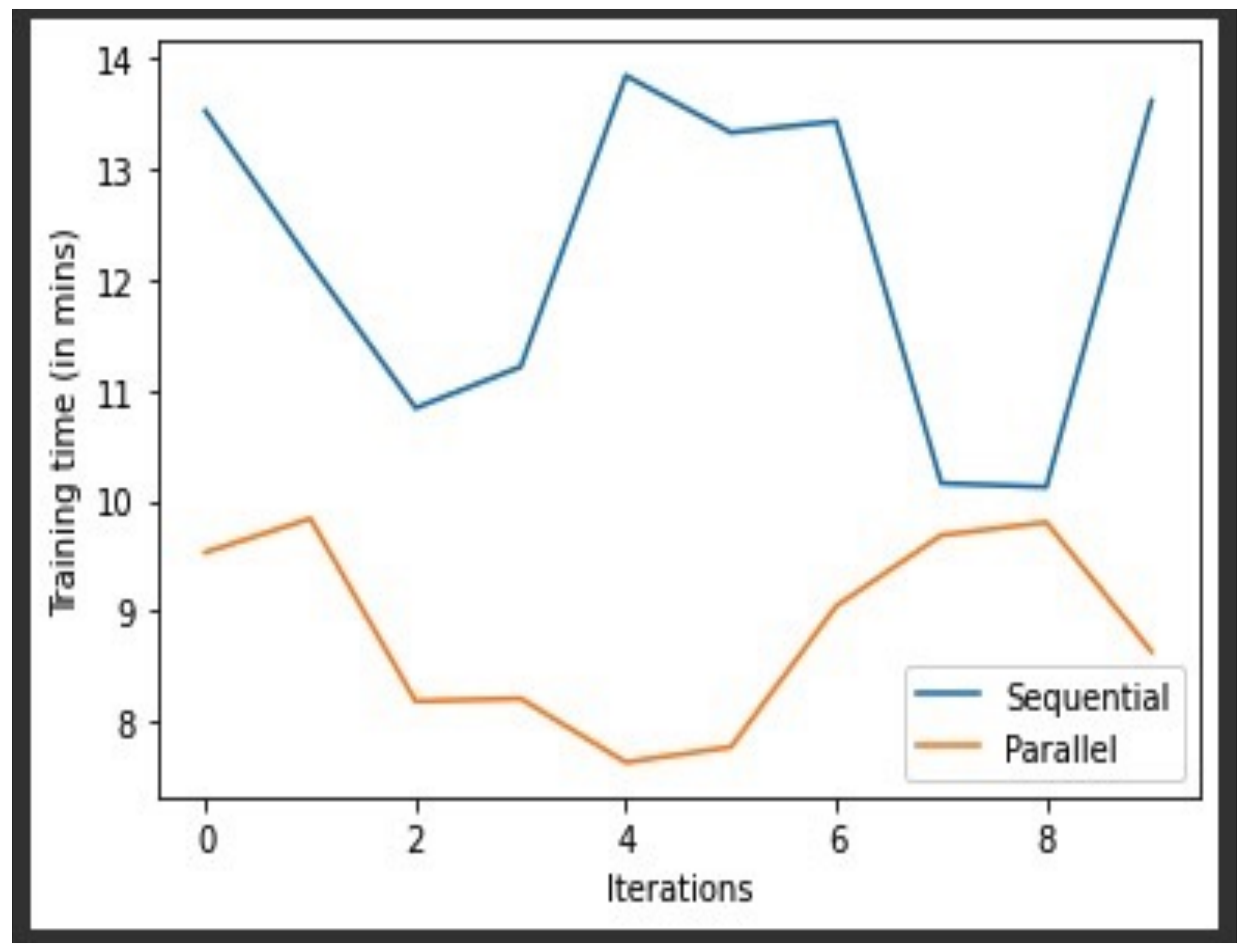
Our Approach

In our algorithm we aim to train N agents parallelly using different RL Algorithms. In case of DDQN, each agent will have its own evaluation network and will be connected to a global target network while computing loss and learning from it. For this we make use of the Parallel Solver as mentioned in the diagram. The solver is responsible for orchestrating the entire episode and for the data flow between different components. We then compare our results of parallel learning with sequential learning to see which approach is more efficient in terms of 1) Score per episode and 2) Percentage of Completion for all agents

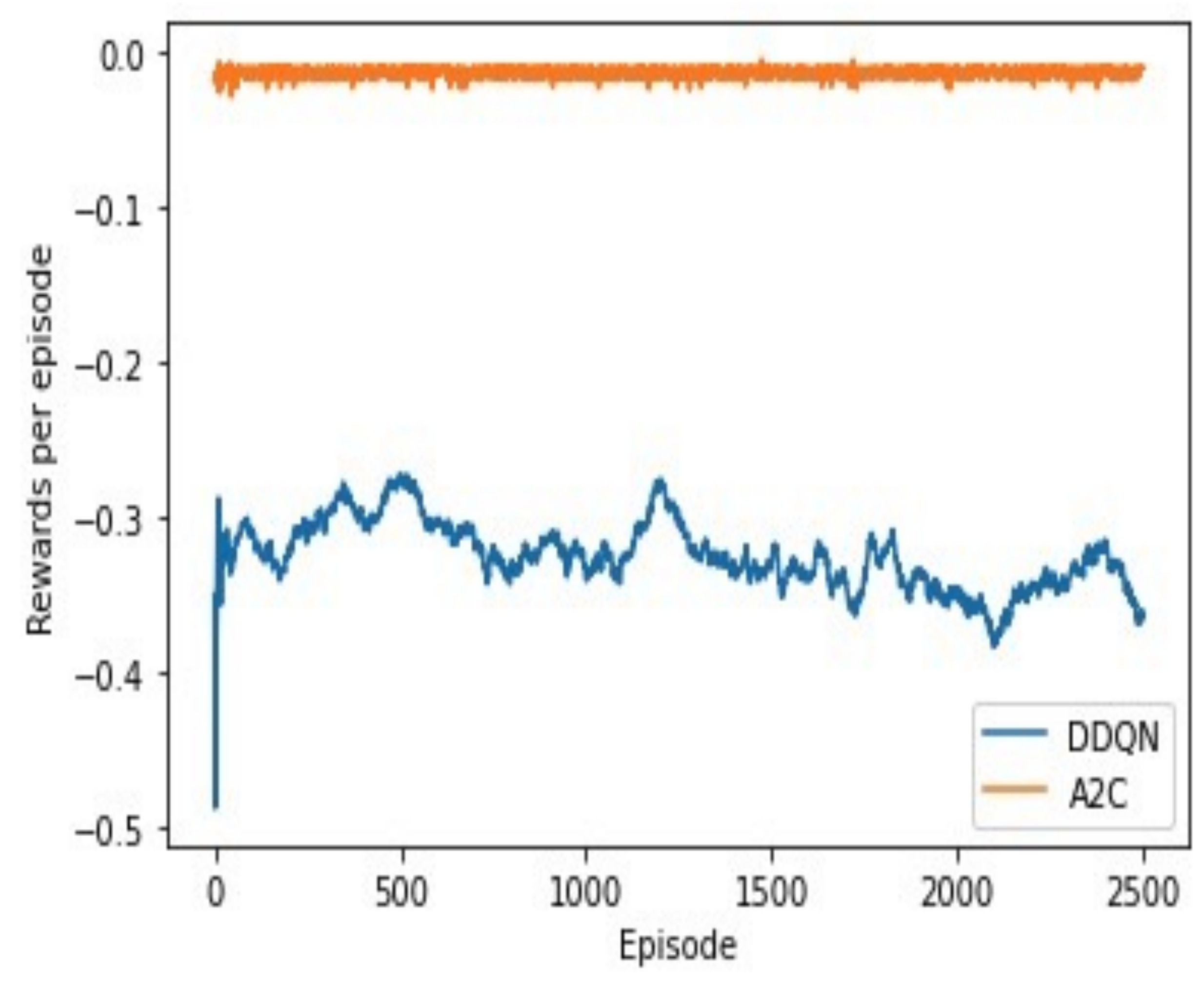


Results

The below graph compares training time for sequential vs Parallel learning, and we observed that parallel performed better.



The below graph depicts the performance of two algorithms: DDQN and A2C. A2C performs way better than DDQN. The training time taken for both the algorithms is nearly equal.



Conclusion

From the graphs and results, we can infer the following things:
1. A2C is a better fit for training and evaluating our environment. DDQN on the other hand gave us highly negative results
2. Parallel Solver, as mentioned in our methodology, reduces the computation time by a huge factor.

References

1.https://flatland.aicrowd.com/gettingstarted/The_Rail_Environment. html
2. https://ars.els-cdn.com/content/image/1-s2.0-S0896627320304682-gr1.jpg
3. CSE-446 Lecture Slides