Sequential Decision making in Autonomous driving using Deep RL

Mini Project Report

As a Course Requirement of

Reinforcement Learning
CS 420

॥ सा विद्या या विमुक्तये ॥

भारतीय प्रौद्योगिकी संस्थान धारवाड
Indian Institute of Technology Dharwad

Course Instructor

Prof. Prabuchandran K J

Group 6

Kushal R (190030023)

Neville Thomas Sebastian (190020028)

Nikunj Pansari (211012001)

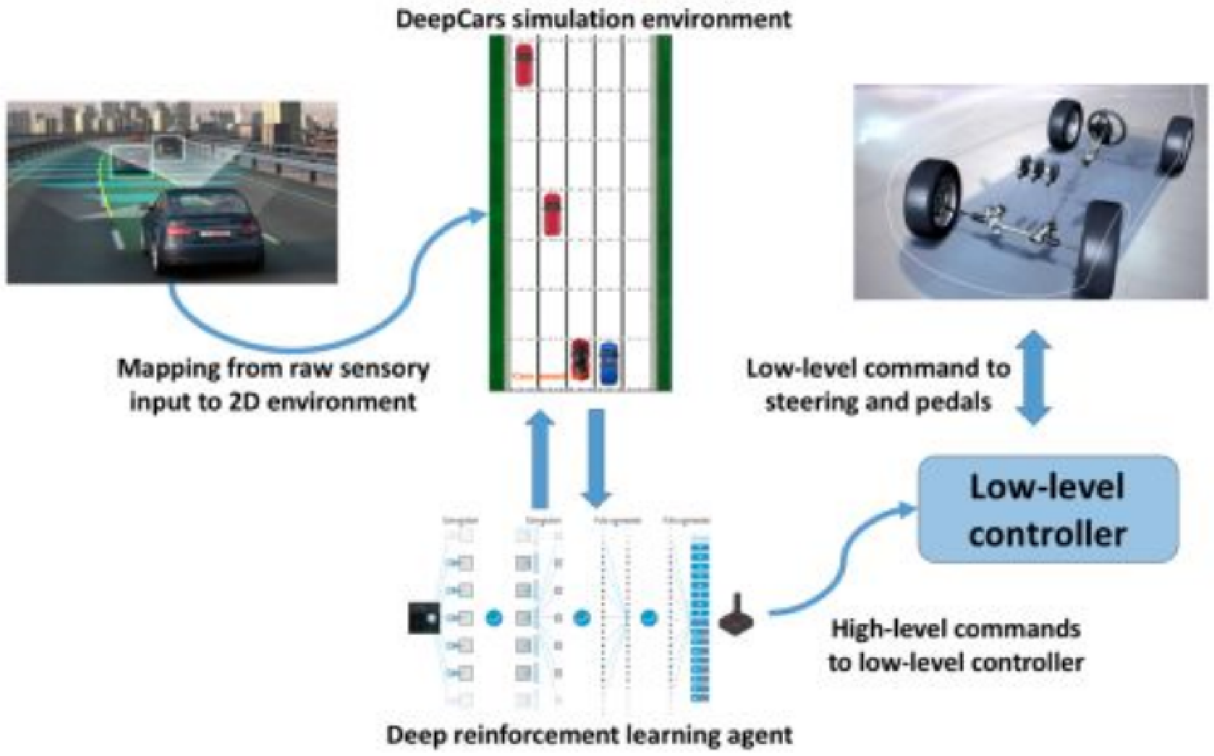
Problem Statement

Employing DeepCars highway simulation environment using Deep Reinforcement learning agents to minimize the collisions with other vehicles on the highway due to the lane change commands.

DeepCars Overview

Illustrating a type of multi-modal architecture consisting of environmental modeling of the ego surroundings (highway scenario) using a deep reinforcement learning agent (DRL) which would be helpful in inferring consistent performance for stochastic highway driving scenarios.

Used DeepCars simulation environment made using Pygame, for simulating the highway scenario.


DeepCars simulation environment
Mapping from raw sensory input to 2D environment
Low-level command to steering and pedals
Low-level controller
High-level commands to low-level controller
Deep reinforcement learning agent

Overall layout of Sequential Decision-making in DeepCars

State & Action Space

- States & actions are discrete.

$$\mathcal{S} = \begin{bmatrix}\text{ego\_lane\_ID} & x_0 & x_1 & \dots & x_n\end{bmatrix}$$

- State representation contains the occupancy grid of the environment.

$$\mathcal{A} = \{left \quad stay \quad right\}$$

- Deepcars the simulation environment receives the high-level control commands as the input vector and provides the state and reward as output.

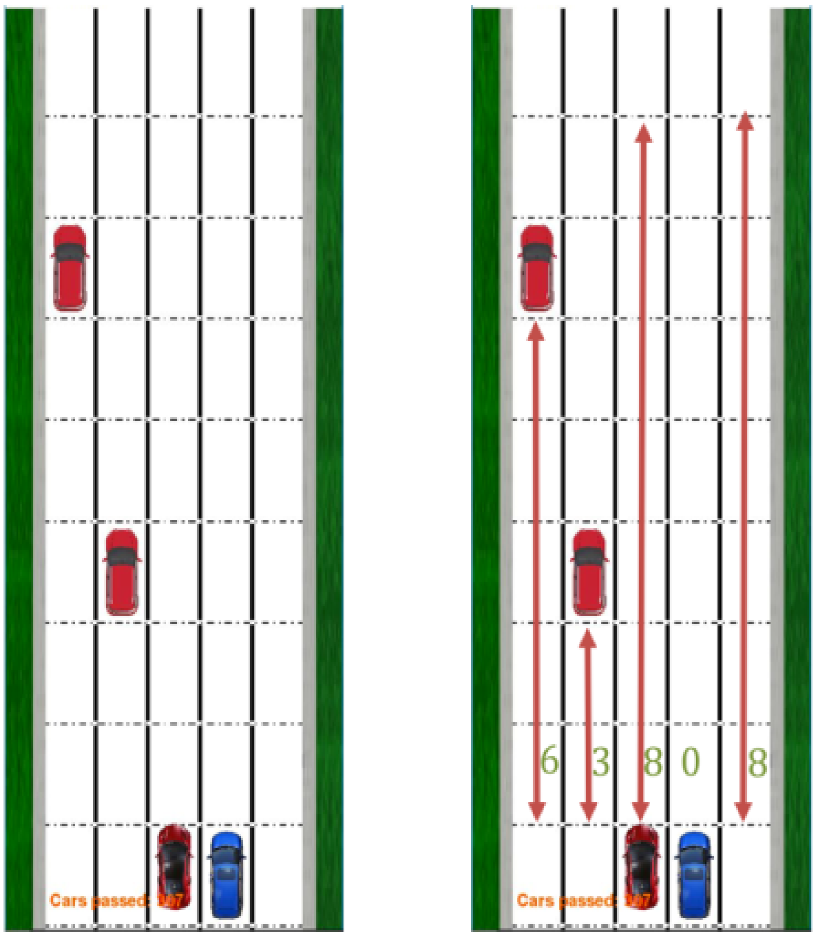


Fig 1: State representation in DeepCars environment

- The prime aim is to train the DeepCars agent such that it can avoid collisions in the highway driving scenario with other vehicles.

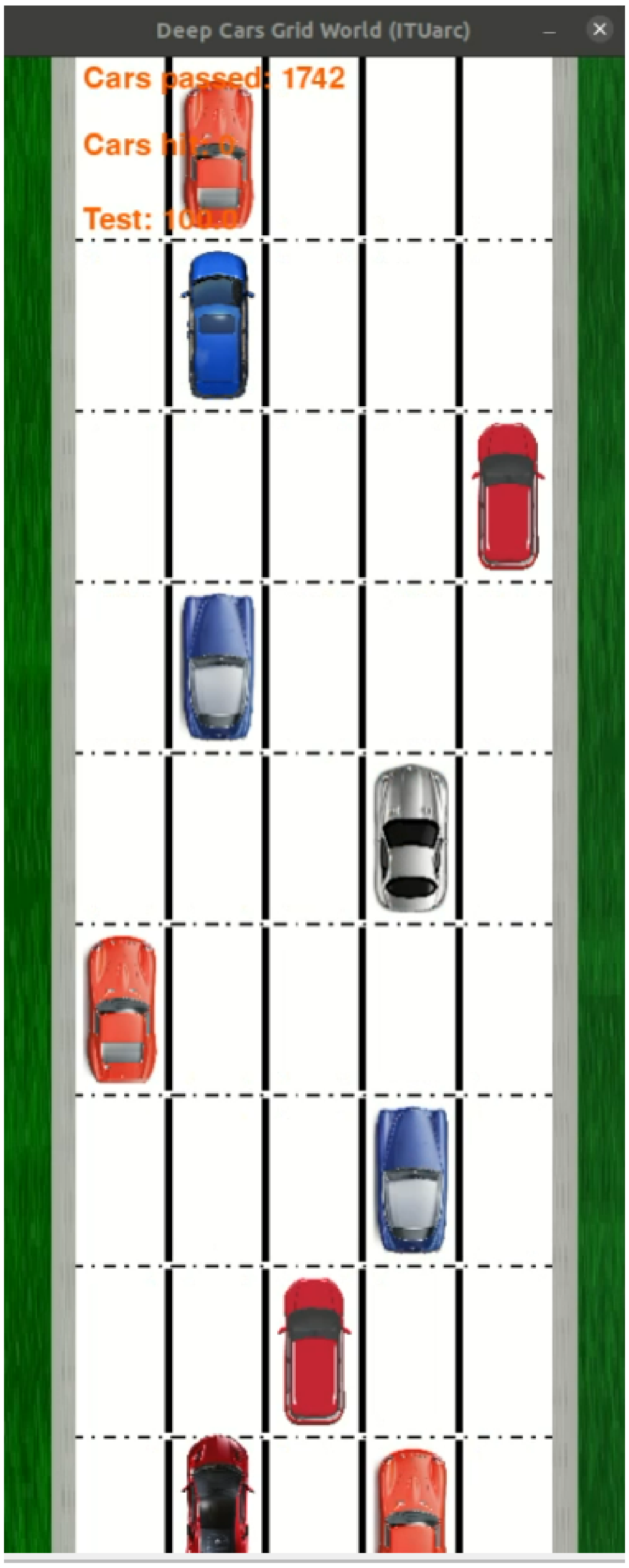


Fig 2: Snapshot of DeepCars environment

- The reward function is defined as follows -

$$\rho(s, a, s') = \begin{cases} +1 & s' \neq s_T \\ -1 & s' = s_T \end{cases}$$

Where ‘sT‘ indicates the terminal state that the agent makes a collision.

Implementation

For large state space, preferred to evaluate using DQN & DDQN deep reinforcement learning algorithms since tabular Q-learning was computationally intensive & time-consuming, and at the same time lead to more no collisions for the test set.

We will illustrate further how DDQN will demonstrate better results in performance analysis.

Hyperparameters Tuning

Optimal settings obtained are defined below :

❖ Learning rate (ɑ) : 0.1
❖ Discount factor (ɣ) : 0.9
❖ Exploration (ϵ) : 0.2
❖ Exploitation (1 - ϵ) : 0.8

Algorithm 1 Deep Q-Network (DQN) with experience replay and real-time validation

```
Input: Initialize:
    replay memory $D$ to capacity $M$
    action-value function $Q$ with random weights $\theta$
    target action-value function $\hat{Q}$ with wights $\theta^{-} = \theta$
Output: $Q^{*}$
initialize sequence $s_1 = \{x_1\}$
initialize preprocesses sequence $\phi_1 = \phi(s_1)$
for $t$ = 1, ..., $T$ do
    $a_t = \begin{cases} \text{random action} & \text{, with probability } \epsilon \\ arg\max_{a} Q(\phi(s_t), a; \theta) & \text{, otherwise} \end{cases}$
    $r_t, x_{t+1}$: apply($a_t$)
    $s_{t+1} = s_t, a_t, x_{t+1}$
    $\phi_{t+1} = \phi(s_{t+1})$
    $D \leftarrow (\phi_t, a_t, r_t, \phi_{t+1})$
    $(\phi_j, a_j, r_j, \phi_{j+1}) \leftarrow random(D)$
    $y_j = r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-})$
    preform a gradient descent step on $\frac{\partial\left(y_j - Q(\phi_j, a_j, ; \theta)\right)}{\partial \theta}$
    if validation_phase then
        fix network parameters $\theta$
        for $t'$ = 1, ...,$val\_episodes$ do
            take greedy actions: $a = \max Q(\phi(s_{t'}), a; \theta)$
            record episode rewards
        if mean validation reward is increased then
            $\theta_{backup} \leftarrow \theta$
    every $C$ steps; $\hat{Q} = Q$
```

Fig 3: DQN Implementation Algorithm

The below define figure depicts the generalized architecture of neural networks utilized in DQN and DDQN deep RL algorithms.

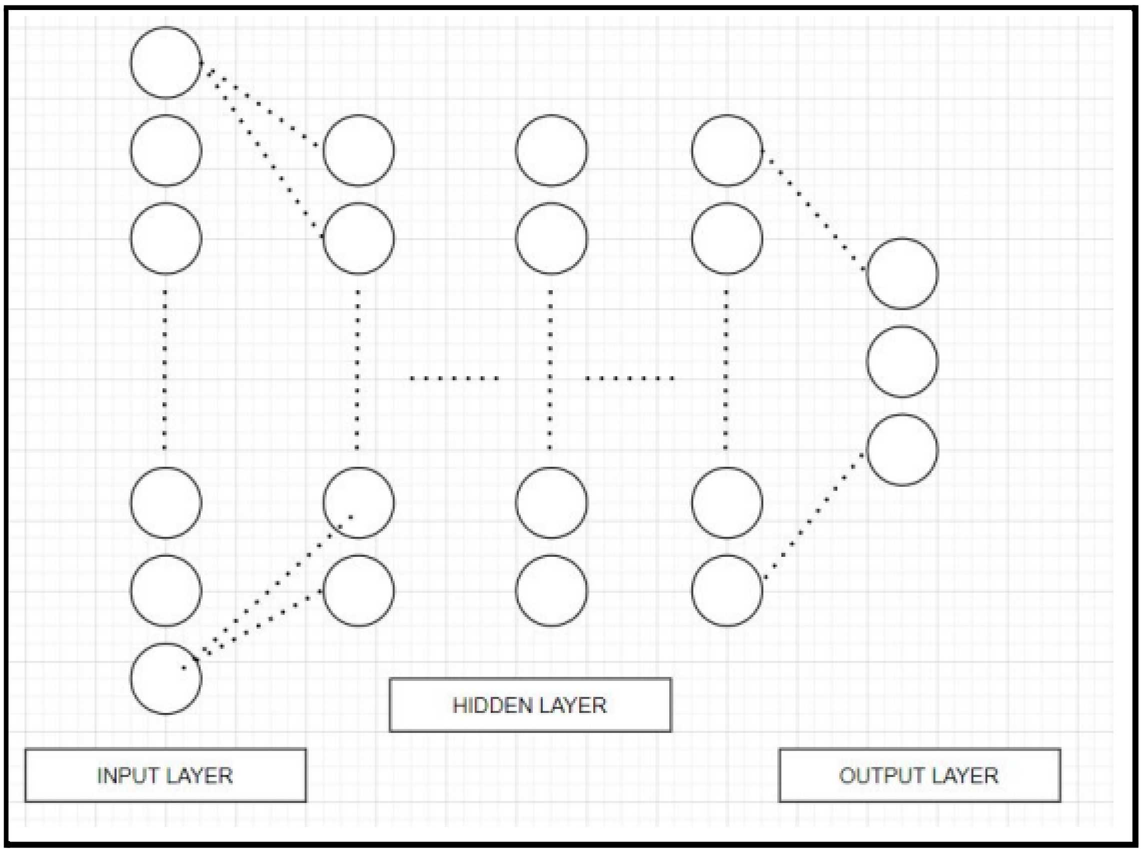


Fig 4: Generalised Neural Network Architecture

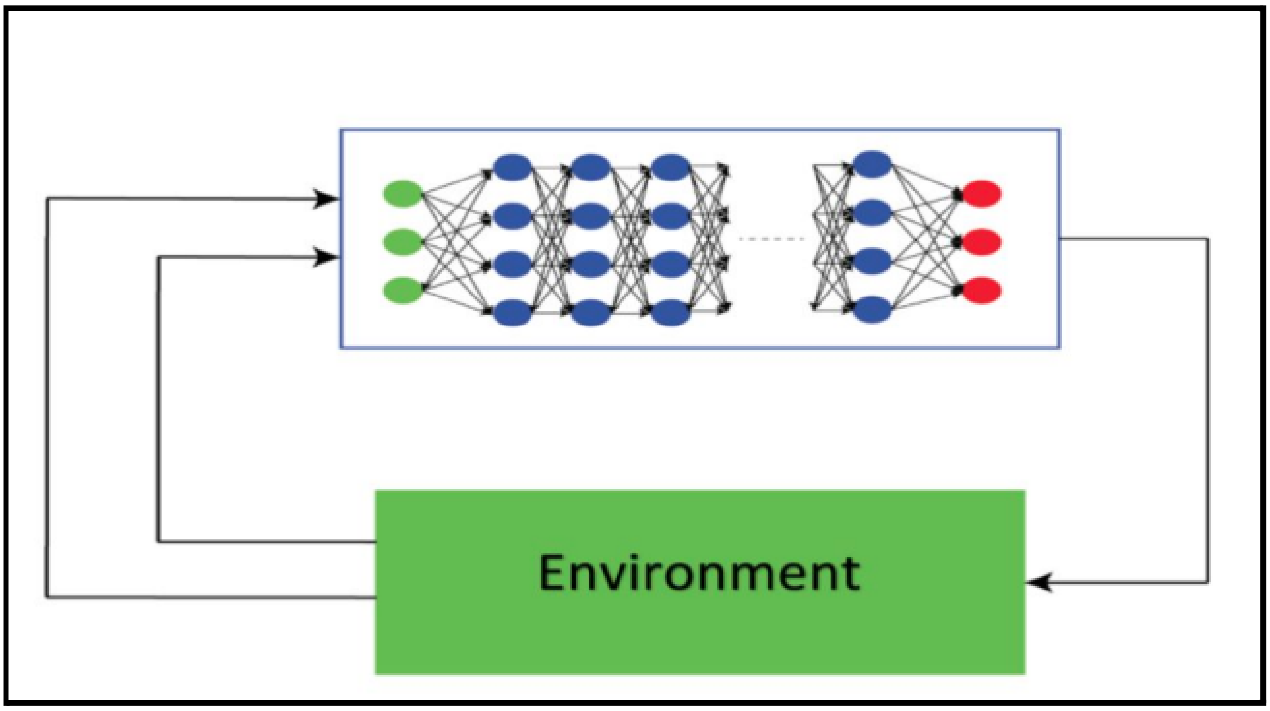


Fig 5: DDQN generalized architecture

Performance Metric

$$\text{Accuracy} = \frac{\#\text{ of passed cars}}{\#\text{ of all cars (passed + collided)}} \times 100$$

Important parameters:

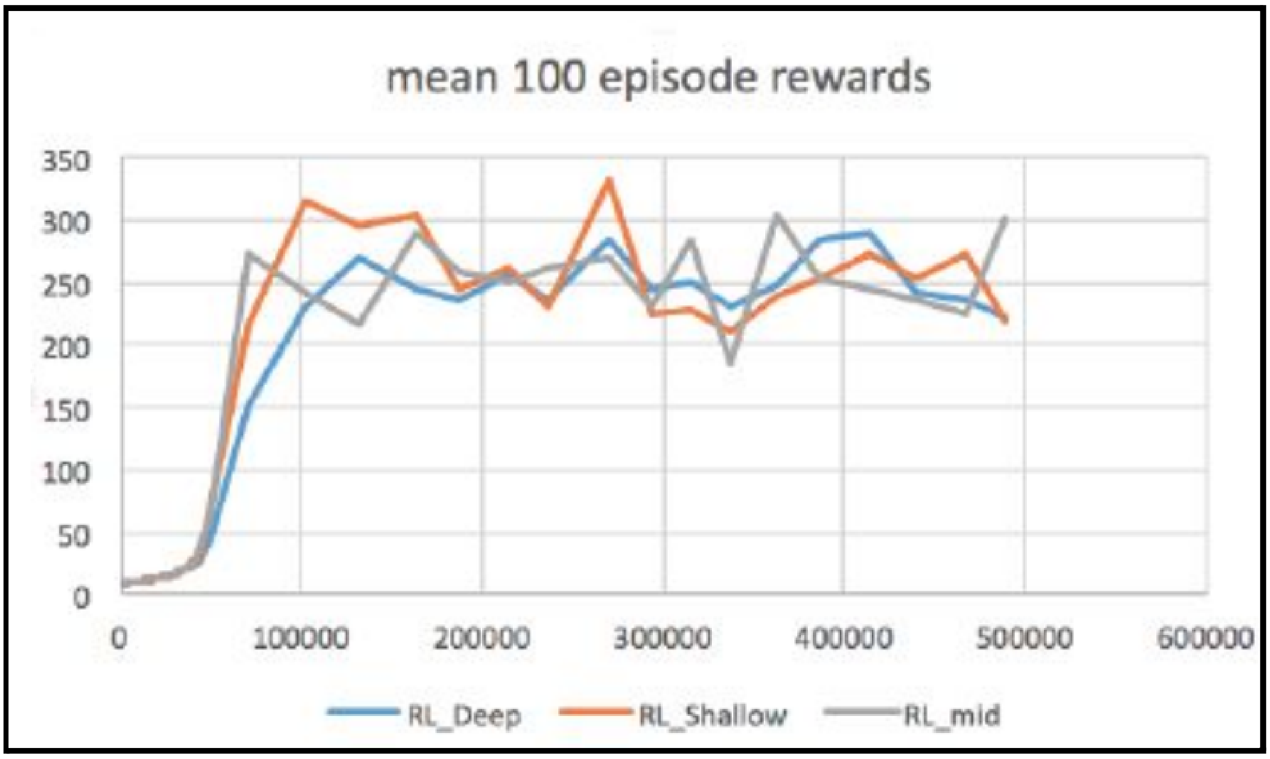
- Accuracy
- 100 eps reward time
- reward

Experimental Setup

- Trained for 500000 steps and considered for evaluation for 100000 steps using 3 different types of neural network architectures : shallow {16} , medium {16,32} & deep {16,32,16} neural network architectures . [Tested on Intel Core i9 2.60GHz and GPU: GeForce GTX 1080 Ti]

- To get a smooth plot for the reward values, use means cumulative reward every 100 episodes as the performance metric.

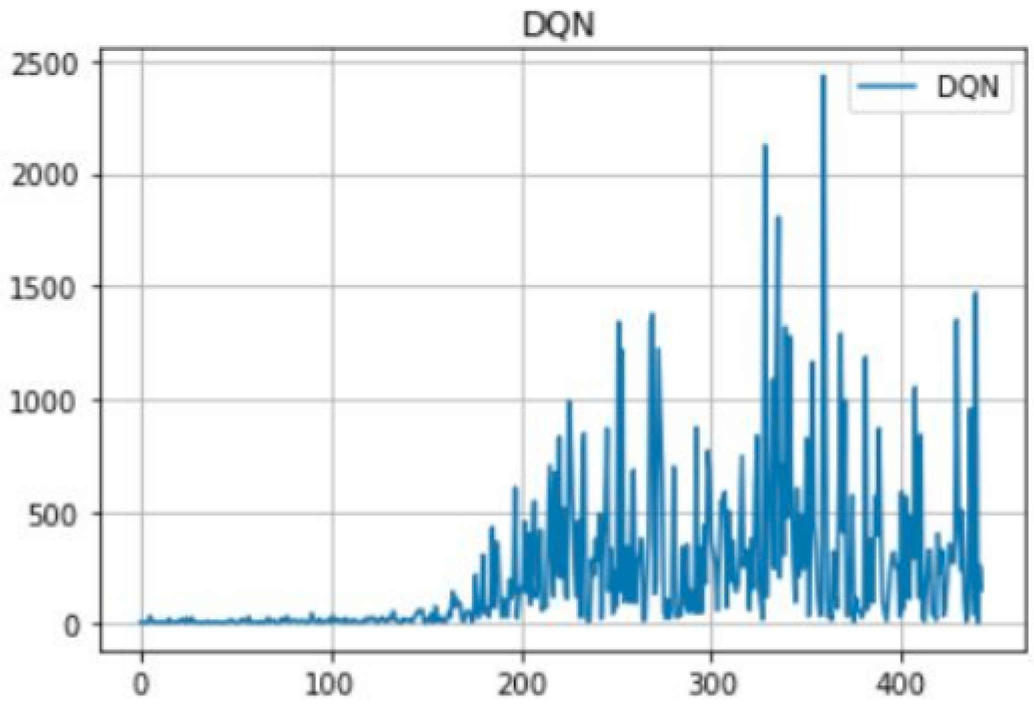
DQN Performance



- From the inferred plot, decided to go with RL_Shallow as the Deep neural network model consisting of 16 neurons, with input dimension 50 and output dimension 30.

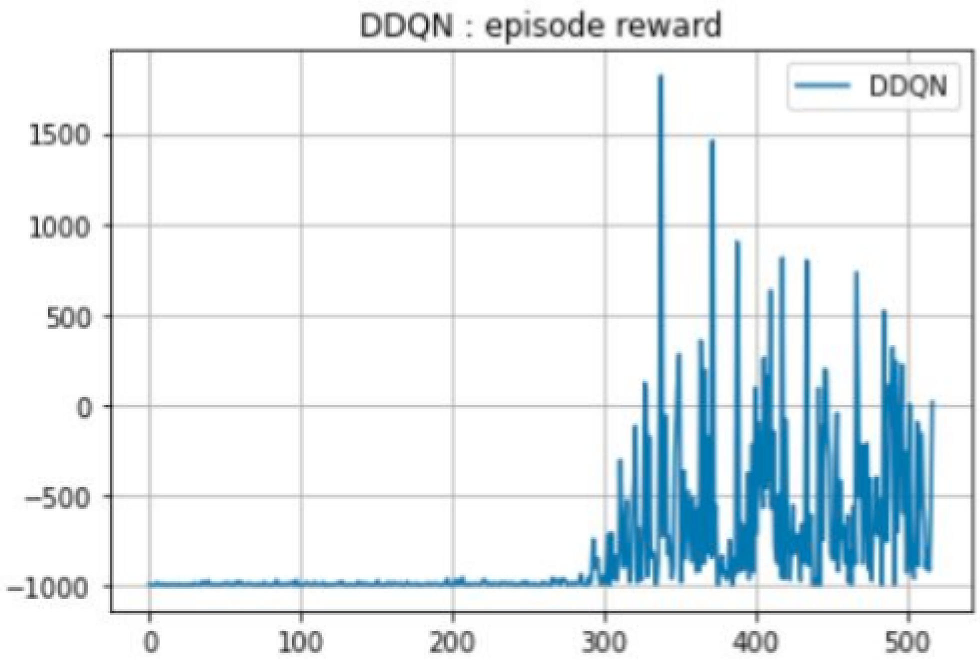
Results

DQN:episode reward



Accuracy=99.48%

DDQN

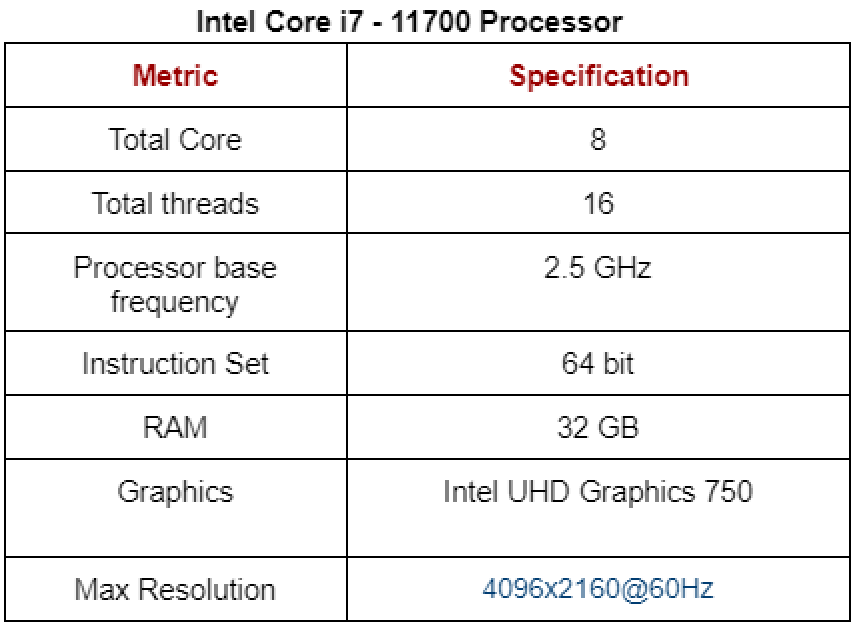


Accuracy=99.4%

- We can infer from the plots that DDQN performed much better than DQN in the long run.

DeepCars Performance Analysis

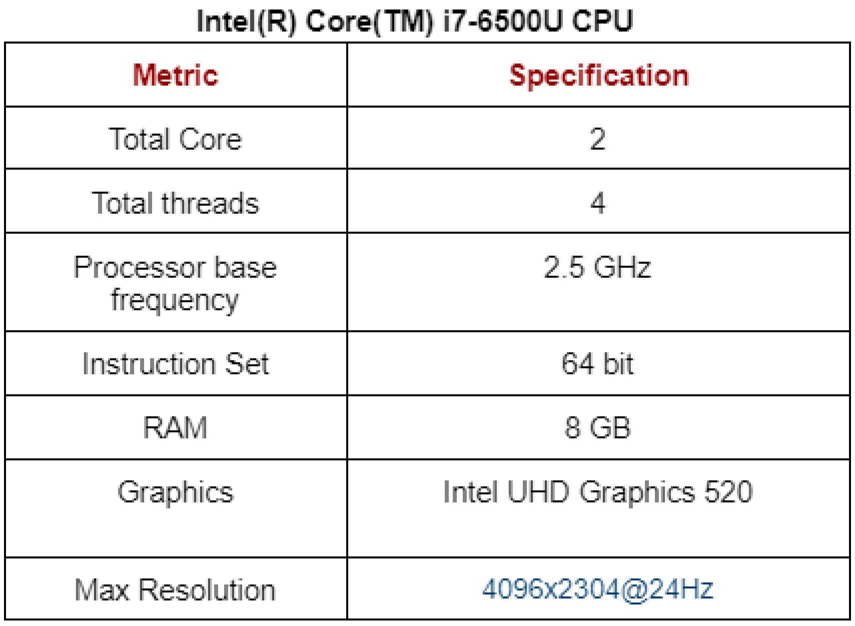
- To test the performance of **DQN & DDQN** deep RL algorithms on different kinds of processors, we carried out a detailed analysis of Performance on 2 different Intel processors namely **Intel Core i7 - 11700 and Intel Core i7 - 6500U.**



Intel Core i7 - 11700 Processor

Metric	Specification
Total Core	8
Total threads	16
Processor base frequency	2.5 GHz
Instruction Set	64 bit
RAM	32 GB
Graphics	Intel UHD Graphics 750
Max Resolution	4096x2160@60Hz

Fig 5: Specification of Intel Core i7 - 11700



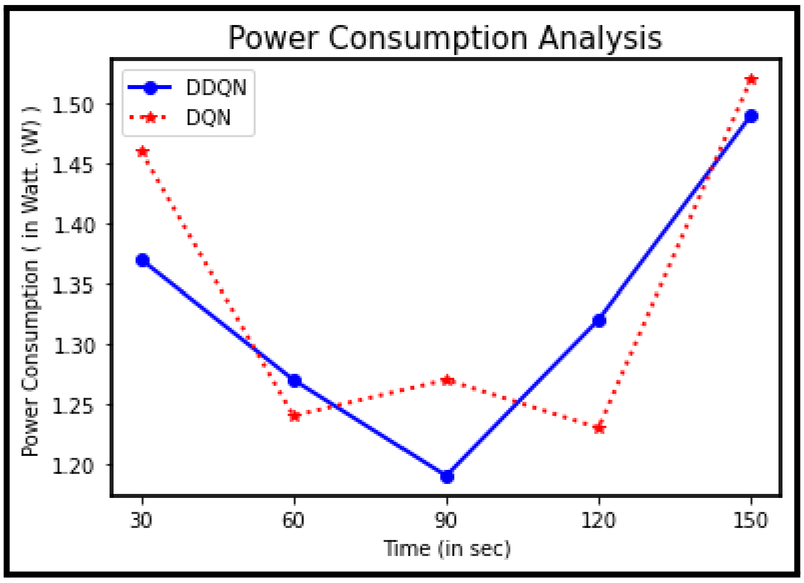
Intel(R) Core(TM) i7-6500U CPU

Metric	Specification
Total Core	2
Total threads	4
Processor base frequency	2.5 GHz
Instruction Set	64 bit
RAM	8 GB
Graphics	Intel UHD Graphics 520
Max Resolution	4096x2304@24Hz

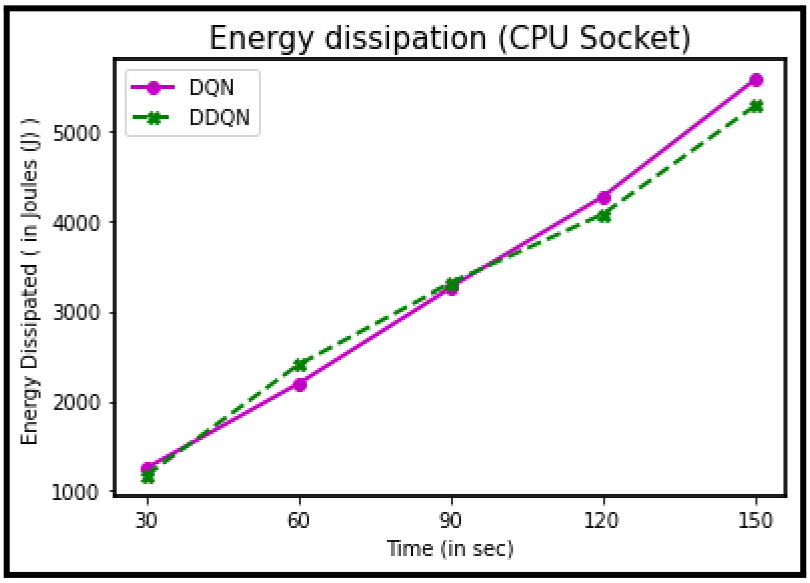
Fig 6: Specification of Intel Core i7 - 6500

- We took performance metrics as mainly power, energy, and temperature.

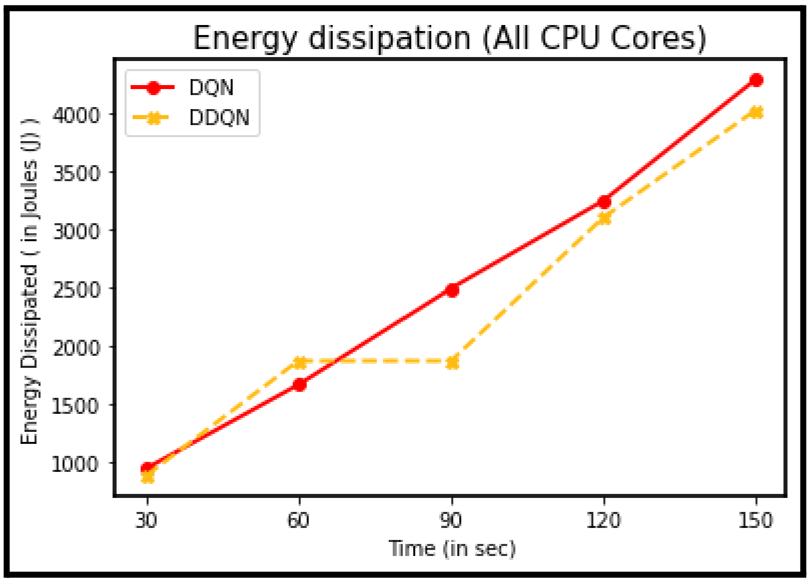
Performance Analysis of i7 - 11700



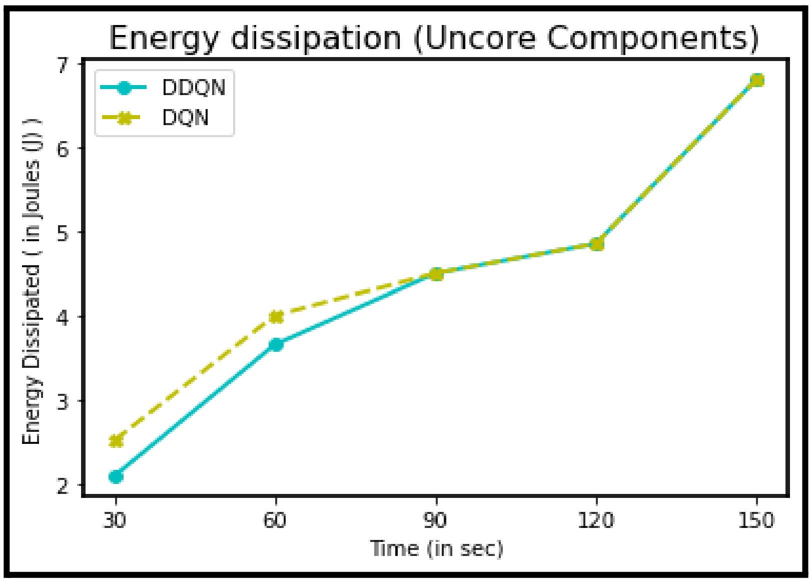
- In the case of energy consumption, the CPU socket refers to the per package domain energy consumption, considering all the sockets in a processor.

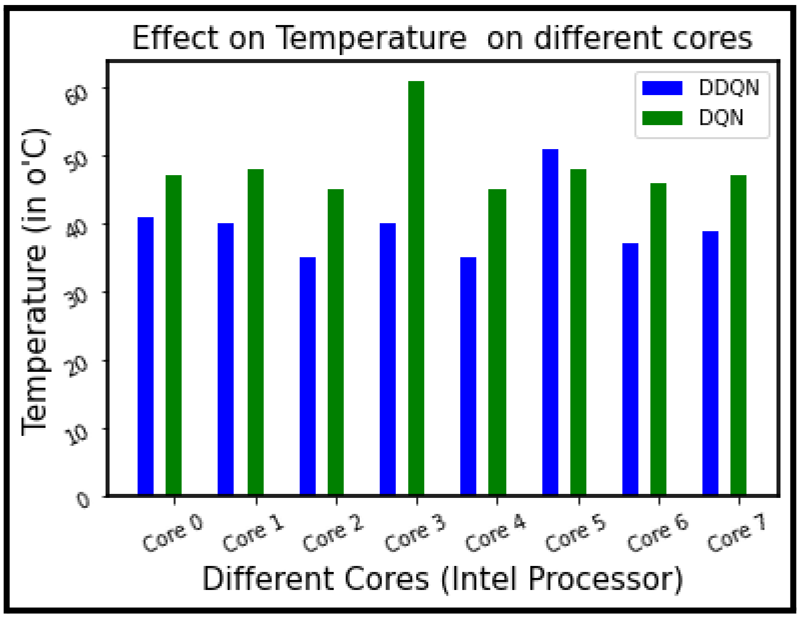


- In the case of energy consumption, all CPU core refers to the per-core domain energy consumption, considering all the cores in a processor.

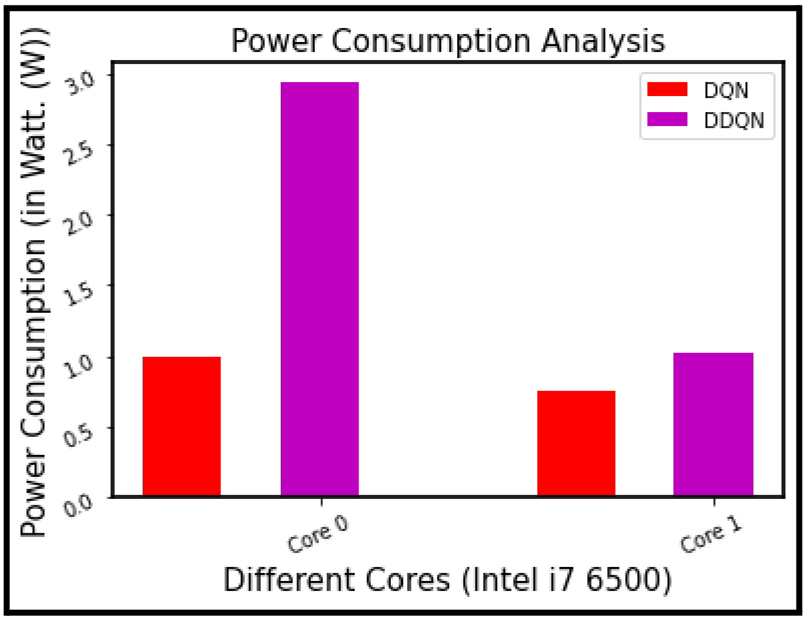


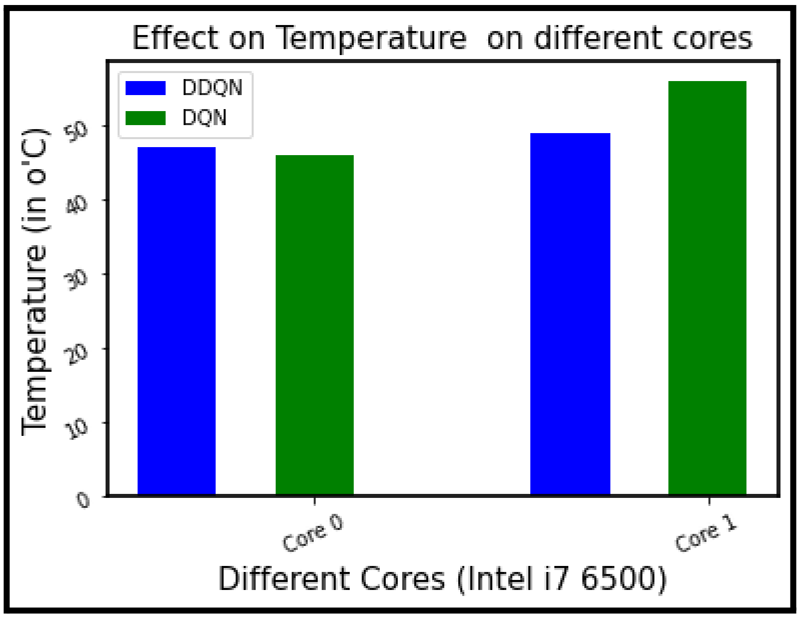
- In the case of energy consumption, Uncore components refer to the per-un core domain energy consumption, considering integrated graphics on Client CPUs in a processor.





Performance Analysis of i7 - 6500U





Inference of Performance Analysis

- DDQN was better than DQN in terms of performance metrics like energy consumption, and power dissipation in most of the cases, but in terms of temperature, DQN performed better than DDQN since DDQN consists of two neural networks thus it leads to an increase in temperature of the core in the processors.

Conclusion

- Focused on autonomous driving in the highway scenario.

- Focused on the hierarchical architecture of decision-making systems where the sequential high-level decision are made using a deep reinforcement learning algorithm.

- We have shown that it's possible to leverage the provided multi-layer architecture to generate high-level commands using DRL with an acceptable reliability score.

- Proper analysis of the performance of DQN & DDQN Deep RL algorithms on two different Intel processors inferred that DDQN in most cases performed better than DQN for both the processors.

References

[1]https://github.com/MajidMoghadam2006/deepcars-reinforcement-learning.

[2]Majid Moghadam and Gabriel Hugh Elkaim. A hierarchical architecture for sequential decision-making in autonomous driving using deep reinforcement learning, 2019.

[3]https://github.com/sosy-lab/cpu-energy-meter