Evaluate learning algorithms and frameworks to help solve the problem statement.  
Assess and select appropriate software tools to successfully work with the data and model(s).  
Discuss model(s) architecture and software pipeline needed to successfully create the proposed solution.  
Assess dataset assumptions, limitations, and constraints in order to develop effective models.

1. **Evaluate Learning Algorithms & Frameworks to help solve the problem statement:**

One very interesting part of working with re-enforcement learning, like many other branches of ML, is the number of potential solutions which you must choose from.

Our project here is no different. We’ve a range of algorithms to pick from ranging from simple ML algorithms like MLP (Markov Decision process) algorithm to Deep Q Learning algorithms. ­­­­ Firstly, it’s important that we note that a MLP algorithm is not a traditional RL model. Traditional RL models consists of 2 main branches; the Model Free and the Model based RL. First is Q-Learning, which is a model free RL algorithm meaning that the agent has access to a model of the environment i.e., a function which predicts state transitions and rewards. Model Free RL models like Deep Q-Learning and C51 are relatively easy to implement, and they work very well within certain use cases, but the limitation is that they are dependent on the agents and although they could work well in one situation, they would usually only work well in that environment.

**We chose two algorithms to work with namely:**

* 1. **Markov Decision Process**
  2. **Deep Q Learning**

We chose MDP because it’s relatively easy to implement, it doesn’t require Tensor flow and doesn’t require too many libraries or complexities and we chose the DQN algorithm because amongst the RL algorithms which we explored, it works best in our scenario (T-Rex game) where we want it to perform well in only one environment and don’t plan to use it in others. As stated, the DQN algorithm is a model free RL algorithm, and we could have used others like the C51 but Tensor flow and Keras have an easy implementation for the DQN and hence the reason behind our decision (ease of use).

1. **Assess and select appropriate software tools to successfully work with the data and models**

|  |  |
| --- | --- |
| **Tools** | **Justification for Choice** |
| Visual Studio & Spyder IDE | Because Visual studio Code is Light, easy to use and Spyder comes with a lot of already in-built Python classes |
| Tensor Flow & Keras | Tensor flow and Keras have RL algorithms, policies and agents’ libraries already implemented and hence we don’t need to design them from scratch |
| Deep Q Learning | The Deep Q Learning model fits our scenario perfectly where we only want our RL model to work well in a single environment i.e., our T-Rex game and it’s also easier to implement than the Model Based RL algorithms like the Alpha Zero for example |
| Markov Decision Process | It’s easy to understand & implement and allowed gave us quick solution for our project when Tensor flow was proving difficult to install |
| Selenium | Allows easy manipulation of web pages, it also allows us to execute JavaScript commands from our python code, so it was highly useful while passing the selected action to the Dino |
| Open AI Gym | Easy to implement AI environment for our Q Learning model |

1. **Discuss models and architecture and software pipeline needed to successfully create the proposed solution**

Unlike Supervised or Unsupervised ML processing, Re-enforcement learning doesn’t involve a specific dataset to work with. This is what makes it special in many ways. Our pipeline was basically leaving our model to run on its own for multiple iterations and on good performances, we save the model to allow easy retrieval later.

Our architecture consists of a Dino object, an agent, and an environment. Our dino object is initialized at the start and it contains all the dino functionalities expected i.e., its look (using sprites), its potentials actions i.e., ducking, jumping, or running and other details of the dinosaur. Our environment is initialized with open AI GYM which creates an environment based on our screen size in a format which can be easily understood by our Q Learning model. Finally, we have our Q Learning algorithm which we pass our Neural network model into along with the number of actions our dino can take i.e., running, ducking, or jumping and the potential states of our game. Our Q Learning model handles the reward system i.e., if our AI dino hits an obstacle it’s rewarded negatively and hence it reduces the probability of it taking that same action again.

For our Dino to learn, we pass it states like the distance from the next obstacle, the height of the obstacle and the speed of the game. The game speed increases as the game goes on.

1. **Assess dataset assumptions, limitations, and constraints in order to develop effective models.**

As stated before, because of the nature of reinforcement learning and the fact that the data is self-generated based on the needs of the algorithm, we do not need to make any assumptions about how we can handle missing/improper values. Also, because our dataset is self-generated by the model, it is impossible to insure coverage of almost all our domain. As well, as previously mentioned, our data is constrained to be positive/non-negative numbers bound by the dimensions of the screen, as well as the possible states of the dino and obstacles.