Design and Evaluation of a New Machine Learning Toolbox for Optimal Traffic Light Control with SUMO and Tensorflow

Reda Mali
LTI Laboratory, National School of Applied Sciences, Chouaib Doukkali University
El Jadida, Morocco

Mohammed Bousmah
LTI Laboratory, National School of Applied Sciences, Chouaib Doukkali University
El Jadida, Morocco

ABSTRACT
Today, all the major metropolises of the world suffer from serious problems of congestion and saturation of road infrastructures. Within this context, one of the main challenges is the creation of appropriate Machine Learning algorithms for the optimization of the traffic lights systems. The objective is to minimize the total journey time of the vehicles that are present in a certain part of a city. In this article, we propose a new toolbox and a framework that brings Tensorflow features to Simulation of Urban Mobility (SUMO). Our work aims to facilitate the use of the SUMO simulator with Tensorflow, for road traffic management. With this tool, researchers will be able to easily test their different models quickly. Instead of spending several days studying the SUMO API, and setting up data mapping procedures, researchers will be able to get results in minutes with our tool. A Web generator let researchers set simulation scenarios, and they can implement their model with the toolbox, based on neural networks and Deep Q Learning. The toolbox exports many metrics, and can compare multiple policies, and different hyper parameters to optimize models. The experimental results obtained show that such an approach makes it possible to obtain significant gains.

Keywords
Traffic Light Control; Machine Learning; Simulation Tool; Deep Q Learning.

1. INTRODUCTION
Nowadays, the road infrastructure is under pressure. This is mainly due to the increase in the number of private vehicles, and the difficulty of creating new infrastructure or upgrading existing one. Optimizing the use of road infrastructure is therefore becoming a major issue [1].

However, it is difficult to implement a manual solution, as a large number of scenarios must be considered. It is also necessary to consider the case of special vehicles, accidents, weather conditions, etc. Current traffic management systems are based on predefined rules, and usually require human intervention to work optimally [2].

Our goal is to democratize the implementation of artificial intelligence-based solutions to solve this problem. With the advent of smart cities, and the democratization of object recognition and tracking tools, it is possible to implement intelligent solutions based on artificial intelligence, which can be deployed on a large scale.

Researchers have at their disposal several simulation tools, such as SUMO (Simulation of Urban Mobility), which allow to create very detailed simulations [3]. They also have very powerful tools, such as Tensorflow, which allows to create models based on deep learning and reinforcement learning. However, it is difficult to master these tools, and the learning curve is usually long.

Our work consists in proposing a solution that integrates the power of both SUMO and Tensorflow. With this, researchers will be able to focus mainly on models. We propose a set of tools and a framework, which allows to test a model in a few minutes, and to take advantage of the power of SUMO and Tensorflow immediately.

Our paper is organized as follows: Section 2 presents the background of intelligent traffic management. Section 3 details the design of our tool, and its different components. And finally, section 4 presents a use case provided with the tool, and the results that the tool can provide.

2. BACKGROUND
2.1 IA and Smart cities
Today, cities should face several challenges, such as population growth, energy consumption, environment preservation, life quality life improvement for citizens, etc. These challenges require the implementation of new techniques and solutions. In the begin of this century, the role of Information and Communication Technologies (ICT) to improve transportation in smart cities has been shown. In this section , we will discuss the role of Artificial Intelligence (IA) in smart cities. Indeed, with the emergence of Big Data, Internet of Things (IoTs) and 5G, AI can propose innovative and optimal policies to solve smart city problems.

Techniques such as machine learning, deep learning, and reinforcement learning consider the specificities of each city in order to implement effective management solutions and policies. In this section, we will present several aspects of AI and their potential in smart cities [4].

2.2 Machine Learning
Machine Learning provides algorithms that automatically learn from past experiences. This field of research has seen many advances in recent years [5]. These advances are achieved due to new algorithms and the availability of online data. Machine learning regained popularity, thanks to the results obtained in the field of image recognition. Subsequently, scientists have felt the potential of machine learning and it's used in other problems such as natural language processing, speech recognition, traffic prediction, fraud detection, etc. Its strength comes from the fact that it is sometimes more efficient to solve problems using learning, rather than to implement solutions manually.

In smart cities, machine learning has a very important role. Thanks to the learning and data provided by IoTs, machine learning can effectively manage many problems related to transportation [6], environment, and health for example. The following sections will detail some machine learning techniques, and how they can be used in smart cities.

2.3 Deep Learning

When we talk about machine learning, the most used technique in the literature is neural networks. An artificial neural network is a model that takes over the functioning of the human nervous system. It consists of a series of connected artificial neurons, and they detect hidden relationships between input data to make decisions. As with the human brain, a neural network must go through a learning phase to be able to classify or detect the desired patterns [7,8].

An artificial neural network is limited in its ability to process raw data, such as images or signals. For several years, the implementation of a neural network required a certain expertise and knowledge about the field of application, to extract the characteristics to be given as input to the neural network. These characteristics are then manipulated by the neural network to classify or detect a pattern. Deep-Learning allows to get rid of this constraint. By associating several neural networks, we obtain models complex enough to directly process raw data. On the other hand, the learning phase becomes more complex, and this has limited the use of this technique in the past. The evolution of computing resources over the last two decades, and the availability of raw data have made deep-learning possible, especially in the field of image recognition [9].

In smart-cities, raw data is generally available, such as images from surveillance cameras, energy consumption, weather, events calendar, geographical data, etc. The use of this data therefore requires complex models to determine the hidden connections between them, in order to optimize the city's limited resources.

Recent work on traffic management uses an image-like representation of the data [10]. This representation allows the use of CNN-based models. Our tool supports this data format as well. Researchers will therefore be able to quickly use CNN models without wasting time setting up data mapping procedures.

2.4 Reinforcement learning

Reinforcement learning is a type of model, different from neural networks. It is particularly effective in problems where a series of decisions must be made. An algorithm based on reinforcement learning learns how to reach a defined objective in a complex environment. In recent years, reinforcement learning has proven its effectiveness. Algorithms, such as AlphaGo, succeeded to surpassing human performance in complex games such as Go. After Go, reinforcement learning algorithms moved on to video games. Now, several algorithms mastered Atari games, without a prior knowledge base. They learned by themselves, playing thousands of times, to progress in games like Space Invader or Pong.

This type of problem cannot be solved effectively with deep learning networks alone. It is not possible to build a knowledge base that indicates for each given situation the right action to take. In this type of problem, it is important to consider past experiences in order to make decisions that bring the algorithm closer to its goal.

In Intelligent Transportation Systems (ITS), there are several problems that are suitable for reinforcement learning, such as automated traffic light management. Several research works [11,12] have demonstrated the effectiveness of reinforcement learning for traffic light management. This is an area that is still relevant today. In this paper, we will present a simulation tool based on reinforcement learning to optimize traffic light management.

2.5 DQN and extensions

Our simulation tool supports Deep Q Learning [13], which is a variant of Q Learning. Q Learning is a Reinforcement Learning technique. It allows an agent to know the right action to take under certain circumstances. To establish the relationship between the actions to take and the environment, a Q Learning model requires a discovery phase. During this step, an agent observes his environment and takes actions. For each action, he gets a reward. The objective is to maximize the received reward. He therefore establishes a relationship between each state of his environment and the possible actions to maximize the reward. When the environment is complex, and with a very large number of states, it becomes impossible to link each state to an action. A neural network is then used to link actions to the state of the environment. This is called Deep Q Learning (DQN).

In recent years, several improvements have been made to DQN algorithms. In order to consider past experiences and remove the correlation between different transitions, a DQN model uses a memory in which it stores its previous experiences. This memory is called Experience Replay. In the learning phase, the agent randomly takes a batch of past experiences, and uses it as input at each iteration.

The DQN is widely used to solve ITS problems. It is used for example to implement traffic light management agents. An agent then observes the state of an intersection, and the DQN model allows him to choose the state of the traffic lights in order to optimize the travel time of the vehicles [14].

2.6 Simulation tools

Simulation tools play an important role in improving solutions for ITS. Since the year 2000, several research laboratories have been working on simulators. The Institute of Transportation Research (IVF) at the German Aerospace Centre (DLR) have been working on one of the most widely used simulators in research: SUMO [15]. SUMO is the acronym for: Simulation of Urban MObility. It is an open-source simulator, which aims to make research results more comparable and facilitate the testing of new models used for traffic management. It offers several functionalities, such as traffic generation, road infrastructure modeling, import of real road network, etc.

It is widely used in several research areas, including vehicle-to-vehicle communication, evaluation of surveillance systems, dynamic routing and navigation, and traffic light management [16]. This paper will deal mainly with the latter issue. Indeed, for the implementation of a DQN model, it is necessary to model the environment in which the agent will evolve, and to calculate the reward obtained for each action performed. These 2 elements are essential for the training phase of a DQN model. Hence our use of SUMO in our approach. It is true that SUMO is not as efficient to evaluate traffic for real life intersections. But its speed, the freedom it offers for modeling intersections, its support for many vehicle types, and especially its TraCI API make it an excellent tool for testing new traffic management algorithms.

3. THE TOOLBOX

Our work aims to facilitate the use of the SUMO simulator with Tensorflow, for road traffic management. With this tool, researchers will be able to easily test their different models quickly. Instead of spending several days studying the SUMO API, and setting up data mapping procedures, researchers will be able to get results in minutes with our tool.

Our tool consists of several software components. Each component offers essential functionalities to perform traffic simulations with DQN models. The figure 1 below presents the global architecture of our tool. It gathers almost all the components.

Our tool is based on the Traci API of SUMO, which allows to control a SUMO simulation via an API. It is therefore possible to retrieve all the data of the simulation at a given time. For the model part, our tool supports the Tensorflow TF-Agents. At first, only DQN agents are supported in our tool.

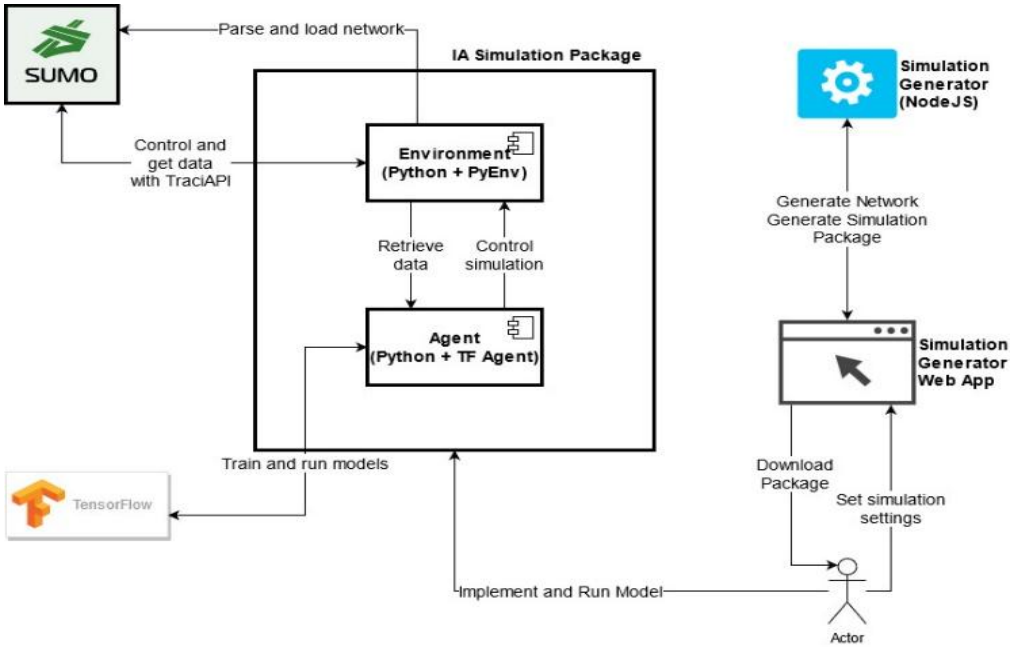

Fig 1: Global Architecture

The following section will detail each component separately.

3.1 PyEnv Environment

To use Tensorflow agents, you need to have an environment, coded in python, that implements a certain logic. The Tensorflow documentation indicates that a valid Tensorflow environment must implement mainly 4 functions: "action_spec", "observation_spec", "_reset", "_step".

"action_spec" must return the specification of the action vector and "observation_spec" must return the specification of the observation vector. In the current version, Tensorflow requires the action vector to be a scalar. Therefore, a projection between supported actions of the traffic manager and integers must be implemented. For the observation, we propose a 3-dimensional tensor, which will contain the following information: position of the vehicles, speed of the vehicles and current status of the controlled intersection traffic lights.

The "_reset" function will reset the simulation's state. It is generally used to start a new epoch. It returns a state which represents the zero time of the simulation. Finally, the "_step" function takes an action as argument and returns the new state of the simulation and the reward obtained following the chosen action.

For our tool, we used a reward formula that reduces the average travel time of vehicles, inspired by this work [17]. Below is the exact formula used:

$$r_t = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{t_i}{T}\right)$$

Where T is the average acceptable travel time.

To control SUMO, we use the Traci API. It allows to get all the information related to the simulation: vehicles, traffic lights, roads, etc. It also allows to control the progress of the simulation, second by second.We set up a Python Environment that implements these functions, and that controls the simulation, using the Traci API. The environment exposes several parameters such as: the duration of an episode, graphical or console mode, SUMO's port, etc.

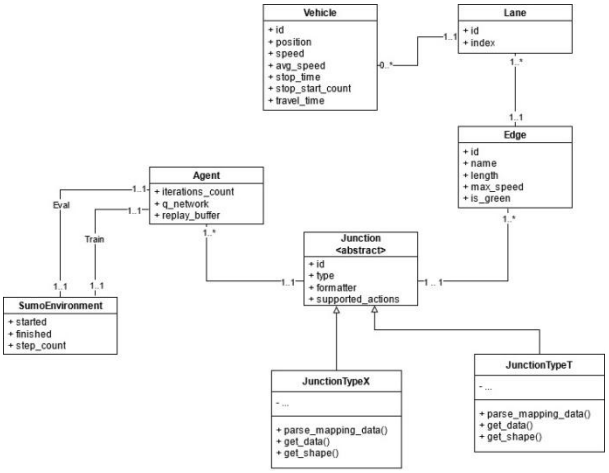

Fig 2: Class Diagram

The tool supports two data representation formats: Image-like mode and Array mode. We have therefore implemented two data mapping processes, one for each representation. From a data model point of view, the tool uses a model based on several entities. They are detailed on the Figure 2.The Edge, Lane, and Vehicle entities represent the same SUMO entities, with fewer attributes. The Junction entities represent an intersection in the road network.

3.2 Managing Networks

Netedit is a powerful tool included in SUMO package, which allows to create networks, in an intuitive way. It also allows to generate vehicles, with departure and destination routes. This information is used by SUMO to create the simulation.

We have included in our tool a set of ready-to-use networks. To simplify the modeling, we have set up vehicles with a unit size (each vehicle measures one-unit length). This is very useful for the image-like representation. It allows us to assume that a vehicle is in one cell at a given time.

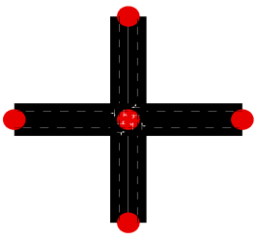

Fig 3: Simple Network type "Cross".

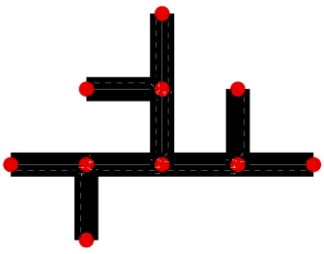

Fig 4: Network with Adjacent, type T

In this version, we propose 16 network models. They are generated according to 4 criteria:

- Intersection type: Cross Intersection, T-Intersection
- Network Complexity: With or without adjacent intersections
- Lane count: one or two lanes per edge.
- Edge length: 20 or 40 units

The traffic generated for each network ensures a maximum duration of one hour for the simulation. This provides the necessary time to test strategies that span over time. The figure 3 and 4 shows examples of networks supported by our tool.

3.3 Supported actions

Generally, the traffic lights of an intersection have standard sequences. For example, for a Cross Intersection, with 4 traffic lights, we can have one of these examples:

- red-greed-red-green, green-red-green-red
- red-red-green-red, green-red-red-red, red-green-red-green

To give more flexibility, our tool supports all possible states. It allows the models to innovate and explore all possibilities. The learning phase will retain the actions that give the most reward to the agent.

Our generator allows to define the possible actions, depending on the type of intersection. In the case of a network with several intersections, only one action set will be used for all the intersections, for simplicity reasons.

During our testing we found that models can sometimes make very quick changes to the network traffic lights. After reflection, we decided to limit this, as it is not practical in real life conditions. To implement this limitation, we allow agents to control the simulation only once every 5 seconds. This parameter is exposed, so it can be modified if needed.

3.4 Tensorflow Agent

Our tool offers a ready-to-use agent, which is based on a DQN Agent from Tensorflow. It is compatible with any Q Network. We provide a default implementation, which uses a simple Q Network. It is possible to give a custom Q Network quite easily.To test a model with the tool, you need to provide the Q Network that represents the model to be tested. The defined Q Network will receive as input the data received from the environment. It is therefore recommended to use the "observation_spec" function of the environment to obtain the specifications of the input layer. The output of the Q network must return the action to perform. The "action_spec" function of the environment allows to obtain the specifications that the network must respect in output.The agent exposes a set of hyper parameters. It therefore gives total freedom when implementing the model. Among the available parameters we find: **num_iterations, initial_collect_steps, collect_steps_per_iteration, replay_buffer_max_length, batch_size, learning_rate, etc**.All these parameters are documented in the README.md file of the generated project.

In addition to the Q Network, it is possible to define an Optimizer and a Loss Function. By default, the tool uses an AdamOptimizer as optimizer and Element Wise Squared Loss as Loss Function. The tool leaves the freedom to modify these options when declaring the agent.

The agent uses a Replay Buffer for the learning phase. All the parameters of the Replay Buffer are included in the agent's hyper parameters. The use of the Replay Buffer was added because of the very good results obtained by the community when it was added. At each iteration, the agent collects a set of data from the environment and adds them to the Replay Buffer. Subsequently, the agent uses a batch of data from the Buffer to update the Q Network weights

```
Init hyperparameter
Load model
Init TensorFlow agent
Init Replay Buffer using a random policy
Evaluate Agent before training in eval env
Reset training env
For i in (0 -> num_interations) do
  For j in (0 -> collect_count_by_iteration) do
    Get current training env data
    Compute next action
    Update training env with next action
    Save data in Replay Buffer
  Get sample data from Replay Buffer
  Train TF Agent
  If (i mod eval_interval) == 0 then
    Reset eval env
    For j in (0 -> eval_episodes_count)
      Reset eval env
      For k in (0 -> eval_episode_steps_count)
        Compute next action
        Update eval env with next action
        Update episode reward
    Update global eval reward
Save policy
Export data
```

Algorithm 1: Training Algorithm

Two environments are used by the agent: a training environment and an evaluation environment. The agent automatically pilots the two environments during the training and evaluation phases. At each epoch, the agent resets the two environments.

The tool offers functions for saving and loading policies. The agent implements these features. It is therefore possible to save a policy by calling a simple function. Loading an existing policy is also easy, the agent can use an existing policy on a given environment by calling a simple function. The algorithm used by the agent for the training phase is detailed in Algorithm 1.

3.5 Comparing Policies

When developing a model, it is imperative to compare the results with other models. Our tool offers two pre-implemented models to facilitate the evaluation of a model.The first model implemented is based on the "Fixed-Time" policy. It is the most used model in the field today. It is therefore imperative for any new model to give better results than the "Fixed-Time" policy [18]. Our tool proposes to use the "Fixed-Time" policy on an environment identical to the one used to train or evaluate the new model. The tool returns the average of the reward obtained for each epoch, in the form of a CSV file. It is therefore possible to draw comparison curves between the two models.

The second model implemented is a random policy. It allows to see if the tested model gives better results than a random algorithm. This policy can be used with all possible actions. The policy will choose at each time interval, a random action to perform. The result is retrieved in CSV format, identical to the one generated by the tool during the training phase of the model to study.It is of course possible to test policies that are already documented in the literature. However, they must be implemented. One of the goals of our tool is to facilitate the comparison between models in the future. When researchers use our tool to test and prepare a new road traffic management model, they can publish the source code of their model or generate the policy. Other researchers can then retrieve their work and compare it with new models.

3.6 Web Generator

To simplify the use of our tool, we have set up a web interface. The web interface allows you to configure your simulation, and to download a project ready to use. It is possible to set up the project, according to the following options: (1) Network Complexity: With or without adjacent intersections, (2) Intersection type: Cross Intersection, T-Intersection, (3) Lane count: one or two lanes per edge, (4) Edge length: 20 or 40 unit, (5) Data Representation: Image-Like or Array, (6) Supported Actions

We have decided to limit the number of options available in the interface to not complicate its use. Other options can be configured directly on the project, especially the hyper parameters.The generated project contains a README.md file detailing the different options that are not configurable in the UI. All parameters are provided with default options that allow to start safely the implementation of a model.On the technical side, the web interface is based on NodeJS for the Backend part, and on VueJS for the FrontEnd part. The choice of NodeJS is justified by its ability to easily manage a large number of requests, and for its asynchronous processing of IO operations [19]. Indeed, the web tool uses algorithms and pre-formatted files to generate the project template, hence the importance of efficiency in terms of asynchronous IO processing of requests.

For authentication, we started with an authentication with Google and Github to simplify. Users will then authenticate directly via an external provider, without having to enter a new password. The Figure 5 below details the technical architecture of the web part.

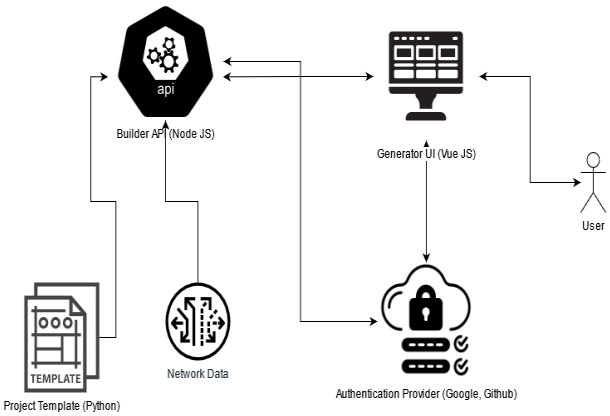

Fig 5: Web Generator Architecture.

3.7 Project Structure

The web interface generates a ready-to-use Python project. We have designed the project structure to be as simple as possible to use. The Figure 6 below shows the structure of the generated project.

The Figure 6 below shows the structure of the generated project.

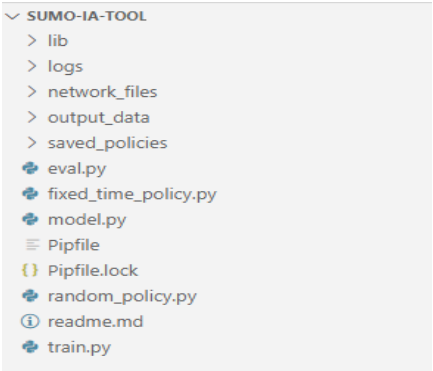

Fig 6: Project Structure.

Below is the role of the most important elements:

a. *Output_data:* This folder contains the CSV files generated during the training of a model. It also contains the data generated by the pre-implemented policies (random and fixed-time policies). The file name is prefixed with a string to indicate whether it is a file generated for a model or for a pre-implemented policy. The file name ends with the creation date. For each run, the tool generates two files. A first output CSV file contains the following data for each episode: Vehicles Count, Cumulative Travel Time, AVG Travel Time, Max Travel Time. The second file contains the Cumulative Reward for each epoch.

b. *Network_files:* This folder contains the different files related to the SUMO network. There are mainly 4 types of XML files (network, routes, settings, config). In addition, there are 2 JSON files: a mapping file, and a file of actions configured for the model. The mapping file is used to interpret the SUMO network data correctly, and the actions file is used to define the possible states of the agent's intersection lights.

c. *Logs:* This folder contains the different log files generated during the execution of the training or evaluation scripts. The names of the files follow the same logic as the files in the output_data folder.

d. *Saved_policies :* This folder is used to store policies. It contains subfolders. Each sub-folder represents a policy. The folder name is prefixed by the creation date. In addition to the files related to the policy, the sub-folder contains a JSON file with all the hyper parameters used, to find them easily later.

e. *Lib:* This folder contains the different python scripts that we have set up to implement the logic of the tool. For example, there are the files of the PyEnv environment and those of the DQN agent.

f. *Model.py :* This is the file where the model will be defined. The researchers will use this file to define the Q Network they will use. The different hyper parameters will be defined at this level too.

g. *Train.py :* This is the file that must be launched to start the learning phase of the model.

h. *Eval.py :* This is the file you have to launch to load an existing policy.

i. *Fixed_time_policy.py:* This is the file that launches the model based on the fixed_time policy.

j. *Random_policy.py:* This is the file that runs the template based on the random policy.

k. *Pipfile:* The pip environment file. It contains all the necessary dependencies for the project.

l. *README.md:* The project documentation file. It indicates how to install the different dependencies of the project, and how to use the different scripts provided in the template. It also details the different hyper parameters available.

4. EXPERIMENTAL SETUP

In this section, we will use the tool to test a model. As mentioned in the previous section, the tool comes with a simple model to get started. We will use this model to illustrate the use of the tool, and the features it offers.

4.1 A Simple Network with two lanes

The first step is to get a new copy of the project. We will use the web interface to configure the tool with the data below: (1) Network Complexity: Without adjacent intersections, (2) Intersection type: Cross Intersection, (3) Lane count: two lanes per edge, (4) Edge length: 40 units, (5) Data Representation: Array, (6) Supported Actions: GRGR – RGRG.

The Figure 7 below shows the configuration used.

Fig 7: Generator UI Form.

4.2 The generated project

Our thanks to the experts who have contributed towards development of the template.The project generated by the web interface is ready to use. It contains the data of the selected network, and it uses the intersection lights with the actions defined in the web interface. bWe find the structure of the project, as presented previously. The README.md file gives all the details to start the project and presents all the project hyper parameters.

To initialize the python environment, you have to install python and pipenv. Once this is done, all the dependencies will be installed thanks to the pipenv file. The installation of the dependencies is done via the command below, as specified in the README file:**pipenv install.** To use the scripts provided with the tool, you have to start a shell with pipenv:**pipenv shell.** We can now start using the tool. To make sure that everything is OK, we are going to launch the random_policy.py script.

4.3 Use a simple Q Network

To setup a model, you have to modify the file model.py. It allows to modify the different hyper parameters, and to propose a model on Q Network. It is possible to propose a model with several floors, and composed of fully-connected networks, CNN, etc.The project is delivered with a model based on a simple Q Network, composed of a single layer. We will use it as a model in this article to illustrate the capabilities of the tool. The Figure 8 illustrates the architecture of the model used.The hyper parameters included in the template can be used directly for the first tests. The hyper parameters used are detailed in algorithm 2 above.

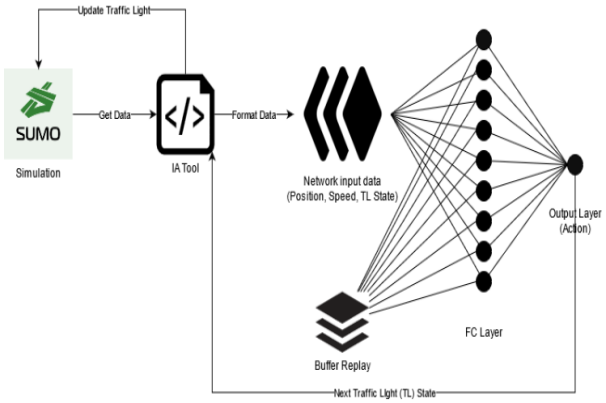

Fig 8: Simple Network Architecture

```
# Environment hyperparameters
episode_duration = 86400. Simulation duration in
seconds.
restart_on_reset = True. Restart simulation after
each episode.
step_length = 5. Traffic Lights change cooldown.
acceptable_travel_time = 40. Acceptable travel time
for each vehicle

# Training and Evaluation hyperparameters
num_iterations = 10000. Training iteration count
collect_steps_per_iteration = 1. Steps count in
each iteration
eval_interval = 1000. Evaluation interval
num_eval_episodes = 2. Evaluation episodes count
eval_episode_steps_count = 600. Steps count in each
eval's iteration
initial_collect_steps = 30. Steps count to
initialize agent

# Q Network hyperparameters
learning_rate = 1e-3. Learning rate.
fc_layer_params = 150. Fully Connected Layer size.

# Buffer Replay hyperparameters
replay_buffer_max_length = 3600. Replay Buffer
size.
batch_size = 240. Replay Buffer batch size.
num_parallel_calls = 3. The number of elements to
process in paralle (Replay Buffer, TF Agents)
num_steps = 2. Specify that sub-episodes are
desired (Replay Buffer, TF Agents)
```

Algorithm 2:Hyperparameters.

4.4 Start training

The train.py script allows to start the training of the model configured in the model.py file. The script initializes two environments, a training environment, and a testing (or evaluation) environment.

By default, the script is configured to run in console mode. It is possible to configure this behavior. The parameter to modify is 'enable_gui'. If it is enabled, SUMO will run in GUI mode, so two windows will appear, one for each environment.

The selected network is displayed, and the training starts. We can directly visualize the actions of the agent on the simulation, as well as the generated traffic. The Figure 9 below shows the training and testing environments, during the training phase.

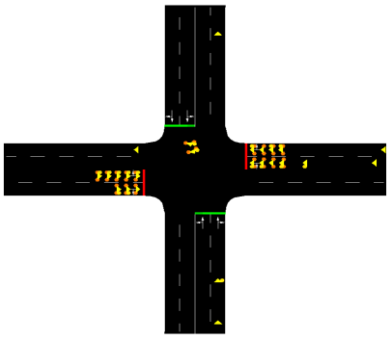

Fig 9:Running simulation

4.5 Exported data

Once the training is finished, the tool generates a set of output files. There is a log file in the logs folder, which allows you to inspect the simulation data afterwards. The output_data folder contains the simulation results as CSV files. This makes it much easier to compare and evaluate the data.

It is possible to test several values of a hyper parameter, and graphically visualize the impact on the simulation results. It is possible to use any library or tool that allows to draw curves to visualize the results. For our model, we obtain the results showed in figures 10-12.

Fig 10: Reward for simple model

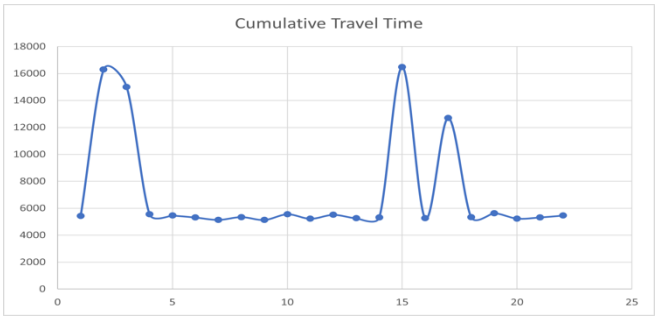

Fig 11: Cumulative Travel Time for simple model.

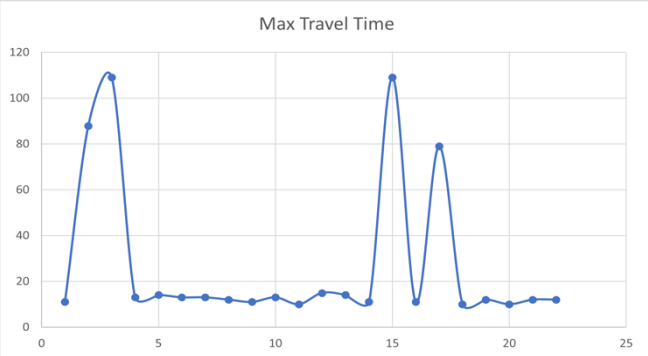

Fig 12: Max Travel Time for simple model.

4.6 Compare Policies / Load Policies

To compare the results of several policies or models, we use the different data files generated during the training of a model. It is possible to use a pre-trained model. To do so, you have to use the eval.py script, with the policy to be loaded as a parameter. The execution generates new data files, which can be compared with another model.

During their work, the researchers compare their new model with the fixed-time method, which is the most used today. To facilitate the work, our tool offers a script that allows to use the fixed-time policy on the network configured for the project. So we obtain a new data file, which we can compare with those obtained with the model under study. To run the fixed-time policy, you have to use the script fixed_time_policy.py.

It is sometimes interesting to compare the results with a random policy to check the efficiency of the studied model. Here again the tool proposes a random-policy which will control the network configured for the project, and will use the configured actions, permuting them in a random way. The same hyperparameters of the model will be applied for this policy. To start the random-policy, you need to run the random_policy.py script. After execution, we get data files related to this simulation, and we can compare them with other data.

For the model we study in this section, we obtain the following results, comparing it with a random policy and a fixed-time policy.

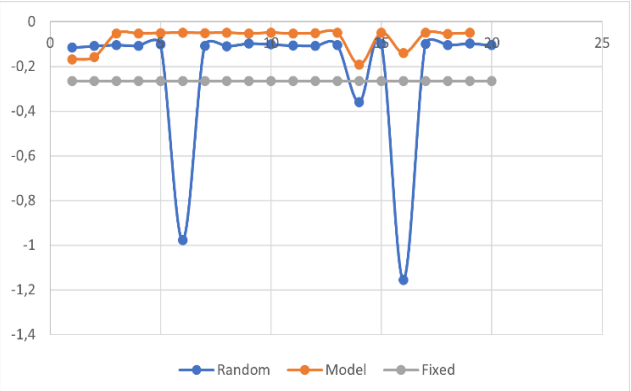

Fig 13: Compare reward between simple model, fixed time mode, and random model.

5. CONCLUSION

In this paper, we have presented a set of tools and a framework that allow to easily use the capabilities of Tensorflow with the SUMO simulator.

Our work allows researchers to focus on the realization of their models, ignoring the complexity of SUMO and Tensorflow.

The tool allows to configure a simulation scenario via a web interface, which generates a traffic management python project, based on Tensorflow and SUMO. The project is ready to use. Researchers can directly fill in a model that will be used as Q-Network by our tool. The tool can run several simulations, and retrieve different data to analyze, such as reward, average travel time, maximum travel time, etc. It is possible to modify the hyper parameters of the simulation, and to compare them or to compare the models between them to obtain a conclusion.

In the future, we would like to add support for other types of intersections, and to increase the number of options available in the current simulation criteria. We plan to add support for traffic anomalies. Finally, as we worked with coordinated agent, we will add support for coordinated agents in this toolbox.

6. REFERENCES

[1] Lijun Wei, Heshan Du, Quratul-ain Mahesar, Kareem Al Ammari, Derek R. Magee, Barry Clarke, Vania Dimitrova, David Gunn, David Entwisle, Helen Reeves, Anthony G. Cohn,“A decision support system for urban infrastructure inter-asset management employing domain ontologies and qualitative uncertainty-based reasoning”, Expert Systems with Applications, Volume 158, 2020, 113461,ISSN0957-4174.

[2] Talha Oktay, Erdenay Yoğurtçuoğlu, Ramazan Nejdet Sarıkaya, Ali Recep Karaca, Mehmet Fırat Kömürcü, Ahmet Sayar,“Multimodel anomaly detection on spatio-temporal logistic datastream with open anomaly detection architecture “, Expert Systems with Applications,2021,115755,ISSN0957-4174,

[3] Jang Seung-Ju, “Design of Traffic Flow Simulation System to Minimize Intersection Waiting Time” International Journal of Advanced Computer Science and Applications(IJACSA), 9(5), 2018

[4] Qi Chen, Wei Wang, Kaizhu Huang, Suparna De, Frans Coenen, “Multi-modal generative adversarial networks for traffic event detection in smart cities”, Expert Systems with Applications, Volume 177, 2021,114939, ISSN0957-4174,

[5] Jafar Alzubi, Nayyar Anand, Kumar Akshi, “Machine Learning from Theory to Algorithms: An Overview,” 2018 J. Phys.: Conf. Ser. 1142 012012.

[6] Johanna Ylipulli, Aale Luusua, “Smart cities with a Nordic twist? Public sector digitalization in Finnish data-rich cities”, Telematics and Informatics,Volume 55, 2020,101457,ISSN0736-5853,

[7] Schmidhuber J., "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85–117(2015). arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003. PMID 25462637. S2CID 11715509.

[8] Bengio Yoshua, LeCun, Yann, Hinton Geoffrey (2015). "Deep Learning". Nature. 521 (7553): 436–444. Bibcode:2015Natur.521..436L. doi:10.1038/nature14539. PMID 26017442. S2CID 3074096.

[9] Nosratabadi, S., Mosavi, A., Keivani, R., Ardabili, S., & Aram, F. “State of the art survey of deep learning and machine learning models for smart cities and urban

sustainability". In International Conference on Global Research and Education (pp. 228-238), 2019, September, Springer, Cham.

[10] Jie Xie, Kai Hu, Guofa Li, Ya Guo, "CNN-based driving maneuver classification using multi-sliding window fusion," Expert Systems with Applications, Volume 169,2021,114442, ISSN 0957-4174,

[11] Liang, X.; Du, X.; Wang, G.; Han, Z. "A Deep Reinforcement Learning Network for Traffic Light Cycle Control.", IEEE Trans. Veh. Technol.2019,68, 1243–1253.

[12] Shabestary, S.M.A.; Abdulhai, B. "Deep Learning vs. Discrete Reinforcement Learning for Adaptive TrafficSignal Control". In Proceedings of the 2018 21st International Conference on Intelligent TransportationSystems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 286–293.

[13] Jianqing Fan, Zhaoran Wang, Yuchen Xie, Zhuoran Yang, "A Theoretical Analysis of Deep Q-Learning", 2019 https://arxiv.org/abs/1901.00137

[14] Bouktif, Salah, Abderraouf Cheniki, and Ali Ouni. 2021. "Traffic Signal Control Using Hybrid Action Space Deep Reinforcement Learning" Sensors 21, no. 7: 2302.

[15] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2018, pp. 2575-2582 [H]

[16] Lucas Rivoirard, Martine Wahl, Patrick Sondi, Marion Berbineau, Dominique Gruyer. "Using Real-World Car Traffic Dataset in Vehicular Ad Hoc Network Performance Evaluation". International journal of advanced computer science and applications (IJACSA), The Science and Information Organization,2016, 7 (12), p390-398.

[17] Van der Pol, E., & Oliehoek, F. A. "Coordinated deep reinforcement learners for traffic light control. Proceedings of Learning, Inference and Control of Multi-Agent Systems" (at NIPS 2016).

[18] Mannion, P., Duggan, J., & Howley, E. (2016). "An experimental review of reinforcement learning algorithms for adaptive traffic signal control". Autonomic road transport support systems, 47-66

[19] Tilkov, S., & Vinoski, S. (2010). "Node. js: Using JavaScript to build high-performance network programs". IEEE Internet Computing, 14(6), 8083.