Modeling Report

Project 12
Reinforcement Learning-Based Traffic Signal Control

Contributions

Maheep Brar
Nicholas Dullam
Elena Gomez
Alejandro Mayo
Zack Reynolds
Siddharth Singh
Qingyuan Yan

1. Select Modeling Technique

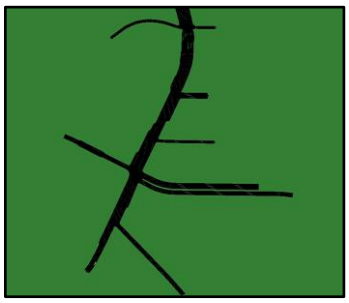
1.1 Modeling technique

- Given our task of using reinforcement learning for optimizing traffic signal control, we can break down our potential approaches into two different categories:
    - Value-based: Where we look to maximize the rewards we obtain (for example, Q-learning), from which initial cases and decisions are sensibly random but learn from the results of previous actions.
    - Policy-based: Where probabilities are given and iterated for each action, given environmental and hyperparameters.
- Various optimizations have been lightly introduced – for example, the use of neural networks for Q-table selection in the value-based approach. While these optimizations have been made aware, we remain shallow on their implementation, as they are not our main focus. These experiments will be noted further in section 3 of the report.
- The reinforcement models we will implement and introduce in this report are a Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC).

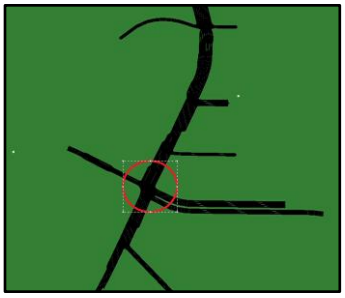
1.2 Modeling assumptions

- One of the main modeling assumptions for our tasks involves our environment – as noted in prior reports, rather than basing our models strictly upon the dataset provided, we opted to use the traffic simulation tool SUMO – using our original dataset as a form of baseline aggregate. In turn, our assumption stands that the SUMO environment is representative of real-world traffic scenarios, given our configurations.
    - While representative of aggregate traffic information, as we noted, it comes with the assumption that no accidents or excessive reaction times will be simulated, however, emergency stops and other abrupt behaviors can be anticipated and regarded in our state.
    - All inputs to the model flows are drawn with the assumption of a relatively normal distribution – including intersection entry speeds, and turning ratios, with the expected value and standard deviation representative of our aggregate standards noted in our data preparation report, post-smoothing. This assumption is drawn as it stands infeasible and counterintuitive to directly represent vehicles from the dataset; as we believe, the introduction of noise will further improve the model's generalizations.
    - As prospected after our previous report, a reduction of scope has been drawn from optimizing Lankershim Boulevard to a single 4-way intersection along itself – for this task, we've chosen intersection 2. All aggregate statistics remain on the basis of the entire boulevard, while turning ratios, as are to be noted, will be relevant to the given intersection. We draw the assumption, of the reduction of scope, on the performance of gains of intersection 2 being generalized to the remainder of the boulevard given the use of the same modeling techniques on each given intersection without awareness of a shared intersection state.

INITIAL SCOPE



REDUCED SCOPE



- Intersection entry speed, as noted to be normally distributed, is also assumed to be similar to the behaviors of the entire boulevard; while turning ratios are explicit to our given intersection, entry speeds are generalized from the boulevards aggregate, smoothed, velocities. For the sake of the model, zero velocities have been removed, given our use of a desired maximum speed on the vehicle types of the simulation, leading to an average velocity, post-smoothing, of roughly 11.3 m/s, alongside a factored standard deviation of 0.69; the speed factor also has a lower cap of 0.75 and an upper cap of 2, as to include the peaks noted in our data preparation report, while removing obvious idle states. Turning ratios/volume for the vehicles can also be found in the data preparation report for intersection 2 with no further adjustments.
- An assumption is finally drawn that the volume of traffic entering intersection 2 (as noted in our prior reduction of scope) at each time is predetermined and will not change unpredictably as the algorithm is being trained.

- We will also try to keep a random percentage of swaps (typically called signified by *gamma*) for specific cases and to avoid some out-of-scope scenarios.
- It is presumed that both the set of possible states for the system and the set of possible actions for the agent are discrete and finite. Discrete action spaces limit our choice of RL algorithms, as many only work with continuous action spaces.

2. Generate Test Design

- When it comes to generating our test design, in our case, given access to prior works in the field, we intend to draw comparisons directly to competing models over shared evaluation statistics – primarily on the basis of vehicle delay/wait times. What must be noted before drawing these comparisons is the importance of consistent state – that is, ensuring similarity of builds in SUMO, consistent intersection design patterns, alongside vehicle speeds and ambiguous environmental state parameters. To accomplish this, we intend to create or share a standard configuration among competing builds when possible; however, due to hardware limitations, equitable build parameters (including sample size, depth of neural networks, and other resource-intensive hyper-parameters) may be minimized to ensure proper builds – these changes, while necessary, may reduce the validity of any comparisons drawn. If infeasible, efforts may be made to draw our configuration similar to the originally reported work, where we'll compare the results from there.

- As was alluded to prior, when it comes to evaluating the performance between competing models, we have a few intended evaluative statistics:
    - Mean wait time: The mean wait time, an approach taken by most pre-existing models, takes the average wait time of vehicles passing through the given intersection as a basis of effective flow. We aim to meet or achieve marginal improvements over existing models, given our limitations, of up to roughly 10% for our given state when assessing mean wait times. Against traditional signal timings, we aim for up to a 50% improvement in mean wait time.
    - Max wait time: The max wait time, an approach more closely-nit to our intended experimentation, measures the maximum wait time a vehicle may experience in a given intersection. This serves as a basis of measurement for the unrealistic priority a model may provide for a given lane of the intersection.
        - While an interesting evaluative statistic, we reserve this for experimentation if time allows – as for the time being, we'll be focused on mean wait time improvement. Regardless, similar to the mean wait times, if evaluated, we can aim to meet or offer marginal improvements over existing models, for up to a roughly 10% improvement in average max weight times. Against traditional signal timings, we aim for up to a 50% improvement, to be paired alongside the improvements noted in mean wait time.
- Finally, we evaluate the obtained outcomes to verify whether we have accomplished our objective. If we fail to meet our goal, adjustments to our models may be made in an attempt to enhance the model's performance relative to our evaluations.

3. Build the Model

3.1 Parameter setting

- When setting our parameters, there are a few notable factors that directly affect the rate at which the model may learn at each "point":
    - Learning rate: The importance drawn to new results – the higher the learning rate, the increased impact of the new result.
    - Discount rate: The value applied to reward values after each iteration – a higher discount rate leading to a more rapid decline in anticipated reward as time progresses.
    - Exploration rate: The percentage of "random" choices applied to the given model – aiding in loop avoidance while incentivizing more experimental actions for long-term gains.
- These parameters help to differentiate the 'exploring vs exploiting' phenomenon. While exploring, we tend to have high exploration value in an attempt to learn – where we try to develop the initial "thinking" process. While exploiting, the exploration value is low, and the learning rate decreases in an attempt to learn and adapt to specific situations – it's worth noting that here, discount rates are often increased in an attempt to reach good solutions as fast as possible.

- While not necessarily applicable as direct parameters, it's best to note some further configurations we may offer more model adjustments and improvement:
    - Reward Function: The reward function, incorporating state to drive incentive for selection of preferred actions – in our case, this will be on the basis of minimizing average wait times, for which inverting the cumulative wait time is common practice/punishment.
    - Network Architecture: The neural network architecture used by each of the RL algorithms. Because of our limited compute and small action space, we keep the neural networks small (only a few fully connected layers at most). This is often used for policy selection, as is the case with DQNs.
- And, as was noted in section 1 of the report, while not directly model parameters, further parameters of state, including vehicle speed distributions and turning ratios/counts must be recognized as configurations. Their intended behaviors have since been outlined in our explanation of scope.

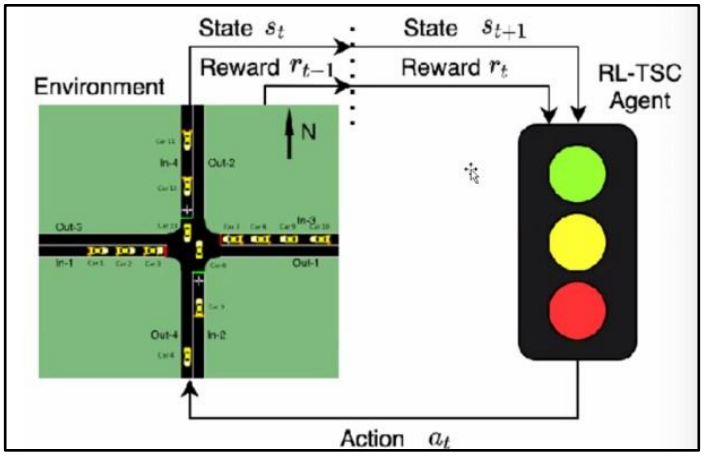
3.2 Models

- For our model development, we used Python with the aid of libraries including *TensorFlow*, *Keras*, *rllib*, and *sumo-rl* – baseline projects were also referenced and utilized in our experimentation, including *Intellilight* for initial prospects in using a DQN model – later DQN implementations were more in reference to the aforementioned use of *rllib* and *sumo-rl*, due to their increased workflow and compatibility.
    - We utilized *rllib* more specifically, for its DQN, PPO, and SAC model implementations and integrations. These implementations will be noted in our model descriptions.
- The system's performance should be predicted using a model that is built to learn from how the environment, which includes traffic patterns, intersection layouts, and other external variables, interacts with the traffic light control system.
- In reinforcement learning, the agent interacts with an environment by taking actions and receiving rewards (positive or negative). The goal of the agent is to learn a policy, and illicit rules for choosing actions based on the state environment.
    - Normally with reinforcement learning, we have policy-based and value-based approaches: as noted prior, with a policy-based approach we try to learn a policy by setting the situation to an action, normally via a function, while with the value-based approach, we look for the option that maximizes the reward that can be expected. In our case, we will set our policy via a neural network that takes into account the number of cars stopped.

3.3 Model description

- We have 3 principal elements: the agent, the environment/state, and the iterations between them. In this case, the traffic light controller is the agent, and everything else is part of the environment.
- The overall goal of the model is to find the optimal policy that maximizes the return. The return is a function of a sequence of rewards weighted by γ (discount rate). In order to predict the expected return, two different functions are necessary. The state-value

function $V\pi\ (s)$ estimates how beneficial it is to be in a particular state (s) under policy π. On the other hand, the state-action function $Q\pi\ (s, a)$ estimates how beneficial it is to take a particular action (a) in state s under policy π. These functions help the agent to choose the most appropriate actions in order to maximize the expected return.



- We utilized *rllib*, a library for implementing reinforcement learning models as noted prior, to construct our models which consist of Neural Networks or other functions that interact with the reward. While *rllib* does not provide complete control, it offers various forms of tuning and adjustment, including the ability to choose between multiple Neural Networks, alongside how rewards are distributed within a certain range.
- In our models, we explored a few different reward functions:
    - Cumulative vehicle delay (default): $r_t = D_{a,t} \quad - D_{a,t+1}$
        - This reward represents the change in total vehicle delay from the previous time step. It is more exploitative, with no long-term rewards
    - Vehicle queue reward function: $r_t = -\Sigma C_{stop}$
        - This reward represents the total number of vehicles that are waiting in the intersection as a negative reward.
- As for the models we chose to explore, we've worked with the following three:
    - PPO - Proximal Policy Optimization is a policy gradient reinforcement learning algorithm – a subset of the policy-based methods we introduced earlier in the report. This means it seeks to optimize the policy directly, rather than estimating a value function to start. To optimize the policy, policy gradient methods utilize gradient descent with respect to the expected return. PPO improves the stability of training by limiting the changes made to the policy per step. It is simple to implement, easy to train, and works well on discrete action spaces. PPO is an on-policy algorithm, meaning it learns based off of the current policy the agent is using. We'd expect PPO to perform well on this task, with improved performance over DQN.
        - Implemented with a 2-layer fully-connected neural network, and experimented with both cumulative vehicle delay, and vehicle queue reward functions.

    - DQN - Deep Q-Networks are the deep learning version of Q-learning, utilizing neural networks to estimate the q-values for all states and actions. A q-value is the expected reward given a state and action. DQN is a value-based method, meaning it uses this estimated value to determine the best policy. DQNs are simple to implement – being an off-policy method, it doesn't utilize the current policy to learn and can reference previous policies from a replay buffer. We'd consider this as our baseline reinforcement learning model.
        - Implemented with a 2-layer fully-connected neural network, and experimented with both cumulative vehicle delay, and vehicle queue reward functions.
    - SAC - Soft Actor-Critic is a variant of the Actor-Critic model that is sample efficient and generalizes well. Actor-Critic models learn both a value and policy function, with the actor learning the policy and the critic learning the value function and critiquing the actor's policy. Soft Actor-Critic modifies the objective function and is off-policy, meaning it can use previous experiences from a replay buffer, similar to the DQN. It seeks to maximize entropy, meaning it maximizes the randomness of actions while still completing the task. We'd expect SAC to generalize well to other intersections compared to our other models.
        - Implemented with a 2-layer fully-connected neural network, and experimented with both cumulative vehicle delay, and vehicle queue reward functions.
- Our final objective is to create our own system for model implementations, establishing all relationships and enabling us to make final decisions about our results. This will allow us with the following:
    - Modifying specific parameters to control how the learning or discount rate updates, which, given our limited configurations in this aspect, is a significant change – as the library is expected to make optimal choices. Nevertheless, we wanted to observe how it influenced our selections..
    - Specify varied reward functions – for instance, we could use a wait-time polynomial to derive the reward function, aiming to minimize both the average and maximum wait times (decreasing in the process a bit the average), as the current interface imposes limitations on us in this regard. This exploration is more broadly noted in our reevaluation of parameters.
- We have attempted this final step, through the use of *Intellilight* as a baseline, but have encountered significant issues. Although we were able to run it, we ran into problems during the training process, and we need to investigate whether it was due to an error or a parameter setting that exceeds our hardware limitations. We can not use Purdue virtual machines, such as MC19, due to problems integrating SUMO and user restrictions for the installation of binaries. For the time being, our current implementations and integrations, while limiting in some configurations, have been suitable for answering our project scope. Re-evaluation of this final, more configurable system, would be ideal with further developments beyond our current scope if time was to allow.

4. Assess Model

4.1 Model assessment

- We will look at the final results obtained after a short period of training, on this case we will not be able to apply certain statistics such as on classification but only see how well we have developed:

Reward Function	Algorithm	Avg accumulated wait time	Avg speed	Avg stopped
Vehicle Delay*	PPO	9953.829	0.2534363	72.37255
	PPO EP4	6508.9	0.28	62.62
	DQN	8757.469	0.3116416	60.31016
	DQN EP13	6279.17	0.374	50.03
	SAC	21262.1	0.1536264	86.00357
	SAC EP11	14941.8	0.189	61.6
Queue	PPO	10746.65	0.2279387	64.11408
	PPO EP51	8311.144	0.2710836	73.82175
	DQN	11206.79	0.434218	57.19251
	DQN EP8	3619.561	0.2711	34.285
	SAC	15953.56	0.2523465	64.27273
	SAC EP9	10600.34	0.281	76.647

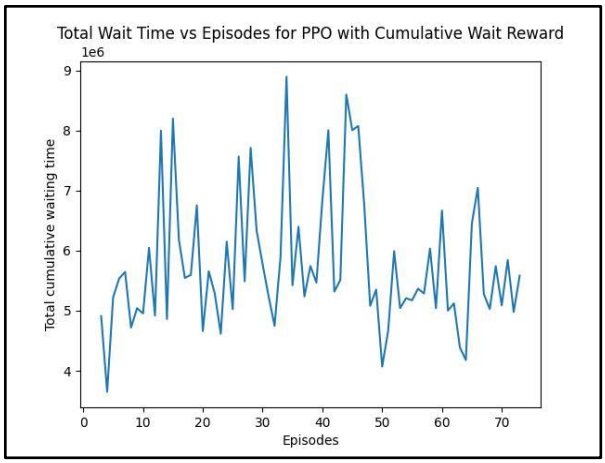
** The models trained on the Vehicle Delay reward function were limited to speed distributions declared by the default config of SUMO, and remain unchanged due to time constraints – those in the Queue reward function are not exposed to this issue.*

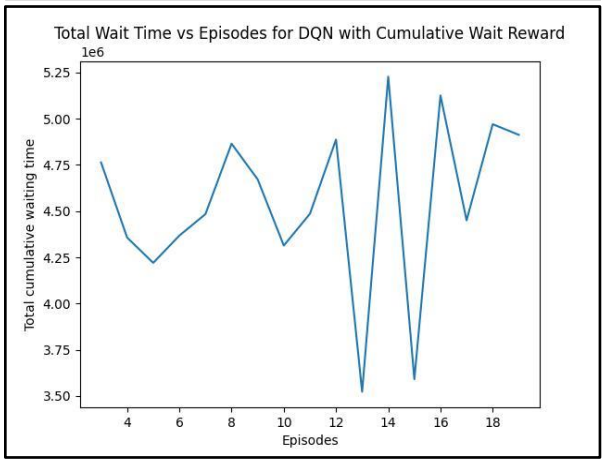
Note: the evaluation statistics addressing specific episodes represent the best performers for a given model – while not necessarily being the last one.

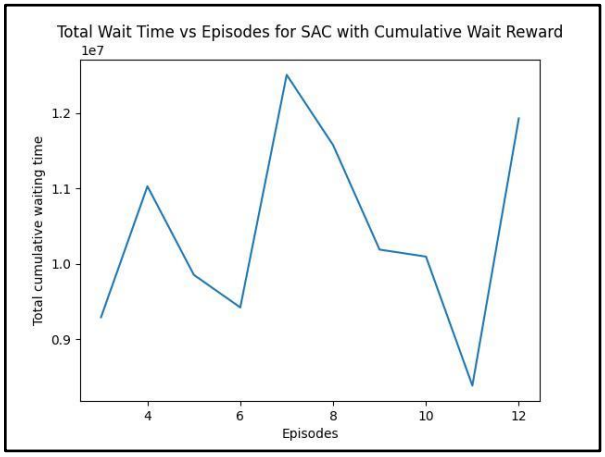
- We can see how the DQN model generally worked better, given a value function of vehicle queues relative to the other models on the basis of average accumulated wait time – holding the best case performance under that regard, while still maintaining competitive performance in maximizing average speed, and minimizing average stops. As for some of the worst performers, SAC performed among the worst when operating with a value function of vehicle delay, offering the worst average accumulated wait times, average speeds, and average stops, among all experimented models.
- It is also interesting to note how the delay reward got a bit better results on the NN that perform the best which can be attributed both to the own reward function (which is the standard use in this field) and the fact that the distribution was different – noting that we will need to solve the computational issues before comparing final results.
- Also, some insightful conclusions can be drawn from the results on the relation of the evaluative statistics, as we see how most of the variables are correlated – like the highest mean speeds alongside the lowest wait times, or the lowest mean wait times with the lowest average total cars stopped.
    - In some cases we see lower volumes of cars stopped but higher wait times, something that could be because in some cases we see how the policy tends to let some of the cars wait for longer periods of time in exchange for a lower overall average wait time.
- Given the complexity of the final policies, alongside the black-box nature of the weights for the neural network decision-making processes, such values are not displayed.
- We still have not reached the best-case expected result – one of the main reasons is training, for which we are aware of extended parameters and training times capable of increasing performance; however, given both hardware and time constraints, potential improvements with changes of configuration are drawn into question.
- It's important to note that results may not be directly indicative of the models' overall performance – as some of the models tend to perform better with exploration, while others may be in the exploitation phase.
- We do not address model generalizability in this report.

4.2 Revised parameter settings

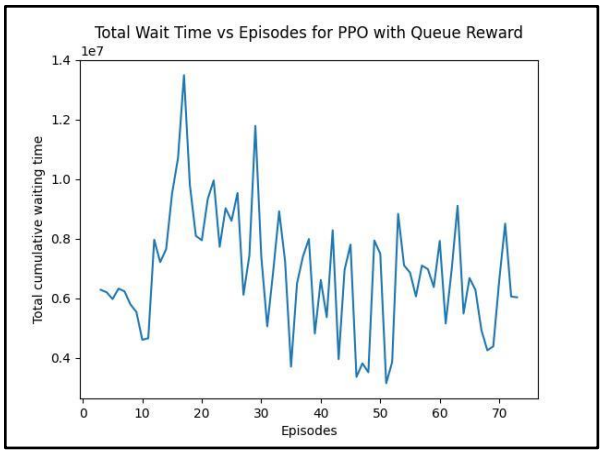
- Now, let's take a look at the reward progressions of the given models, in terms of total rewards per episode:
    - Waiting Time Rewards:

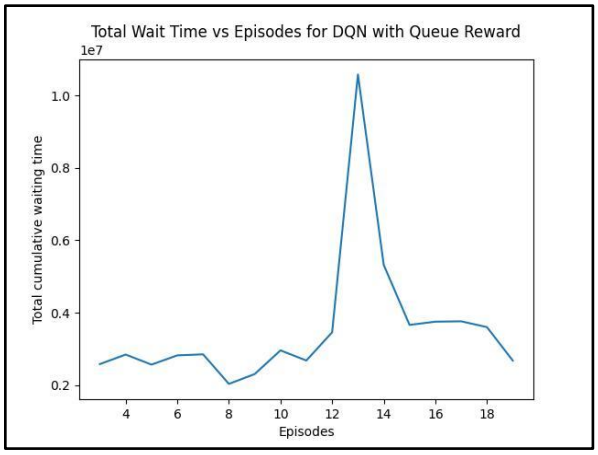


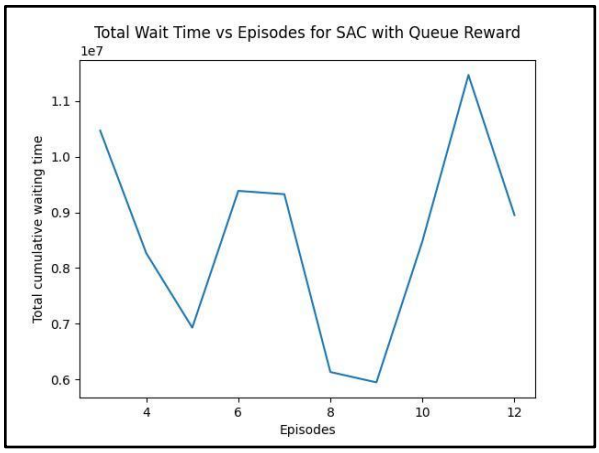




- Queue Rewards:







Note: episodes 1 and 2 from each of the above model expressions have been excluded, due to rllib's initialization behaviors.

- The difference in the number of episodes between model types is due to *rllib*'s implementation of these models. All models were run for 10 iterations, but PPO would make more updates and therefore have more episodes.
- One thing to note when evaluating our above reward progressions, is our limited convergence. We believe, given our limited training times, that the model is not given an

adequate chance to develop its' policy enough for convergence to appear in model performance and evaluations. The constraints and aspirations of such training times have been previously noted in section 4.1.
    - It's also important to recognize PPO's behavior given its extension of episodes relative to the other models, demonstrating better convergence and more "reasonable" results – which further attributes to our assumption drawn on the impact of episodes or general training times on the convergence of total rewards.
    - Also, good results on the first iterations may come from the hard exploration phase, where the model gives a lot of value to the randomness parameter. It looks like the approach of more or less random changes in light patterns performs relatively well on this type of problem.
- As for hyperparameter revisions, apart from state, we are limited in our adjustments to those alluded to prior; however, we could experiment with adding new layers to our best model. One potential approach is to incorporate a recursive neural network, which may improve learning by taking into account the repetitive nature of the actions and environment; the neural networks themselves, apart from their depth, offer limited hyperparameters.
- Another way we may look to revise our models for better evaluative performance, would be in the realm of adjustments to the reward functions. We have some already set up rewards functions, as demonstrated in the model descriptions, being that of a cumulative average wait time, and average vehicle queue. These reward functions, while providing insight and remaining a relatively common standard, could be optimized for improvements towards what we believe would be maximum wait times for vehicles. An adjusted reward function, such as a cumulative average wait time, where the accumulator applies an exponent to vehicles' individual wait times, could leverage a balance between optimizing overall average wait times, alongside maximum wait times for individual vehicles – something that would resemble a more realistic and desired environment.
- Changes towards our environmental configurations could also offer some insight:
    - Vehicle flows: Adjustments to have probabilistic models rather than constants – introducing both moments of high and low flow for testing the generalizations of the model.
        - Here it is important to reattribute our note on the reduction of scope, from the four intersections to only one. If time allows, exploring the possibility of a shared state, alongside an independent state of models for each intersection along the boulevard would be an interesting attribution; as it could have a direct impact on more realistic flows for a given intersection, alongside, of course, the inherent performance gains. Prior works, including *Colight*, are good references for diving into this phenomenon.
    - Vehicle speeds: Seeing if constant speeds from cars would bring up better results – which could be interesting to take into consideration due to the possibility of this fact in the future years with automobile cars.