Thesis Summary - ReLight (Recurrent Traffic Light Control)

Peter Kjaer, Samuel Alexander Vall Andersen

Aalborg University

Over the last decade, cities worldwide is seeing a large influx of new cars on the road. More cars on decades-old road network infrastructure cause traffic congestion, which cause huge environmental and economical damage. The TomTom Traffic Index[1] found that traffic congestion in major cities like London is polluting up to 15 percent of the total yearly CO2 in their respective cities. The American Transportation Research Institute estimates that the U.S. freight sector loses around $74.1 billion annually due to traffic congestion[2] , 75 percent of which occurs in urban areas. One of the trivial solutions to this congestion is to increase the flow throughput in the urban areas. Since around 75 percent of the traffic congestion happens in urban areas, it is trivial to optimize traffic light control to allow for more flow and avoid unnecessary vehicle idling.

Conventional traffic light control in many modern cities today uses pre-defined traffic plans or cycles to control the light phases[3]. Conventional traffic light control, however, cannot perceive and react to real-time traffic patterns, making them unable to dynamically adapt to the changing spatial-temporal dynamics around the intersection.

To allow adaption towards spatial-temporal dynamics for an intersection, recent papers incorporate Reinforcement Learning techniques to perform traffic light signal control. Reinforcement Learning techniques outperform previous conventional traffic control methods big a significant margin and allowed for control adaption towards each individual intersection. These techniques, however, disregard the cycle phases utilized in conventional traffic control and only use flow density to predict the right action based on the current state observation. This may cause inaccurate decisions and non-adaptation towards dynamically prone accidents, roadworks, or special days like weekends or holidays.

To properly capture traffic cycle phases, the agent must be able to know the difference between rush/slow hours, and weekday/weekends. GPLight[4] adds an additional forecasting module to exploit future traffic information in Reinforcement Learning to capture the future short-term traffic flow prediction and use this latent context to combine the result with the Reinforcement Learning agent. This method, however, adds an additional loss since the forecasting module must learn from the state observations and perform a prediction of the short future. Realistic traffic flow is affected by a lot of environmental and temporal factors, which makes it very deterministic and produces random flow which makes time-series prediction a challenging and inaccurate problem. A high loss from this prediction

Email addresses: pkjar16@student.aau.dk (Peter Kjaer), sava17@student.aau.dk (Samuel Alexander Vall Andersen)

module may cause the agent to take wrong decisions and end up in a state which yields bad results.

To circumvent this limitation, we propose ReLight (Recurrent traffic Light control), which uses recurrent controllers to capture long-short term spatial-temporal patterns in a Partially observable Markov decision process (POMDP) environment. Where current Reinforcement Learning solutions are Markov Decision Process (MDP), they are limited by the Makorv Property where the future depends only on the present state and does not depend on history. With this limitation, it restricts MDP solutions from properly capturing nowcasting without an additional forecasting module. POMDP however, is dependent on the history of previous belief states to predict its current belief state. POMDP state is commonly obscured, and reliant on the spatial-temporal patterns from its history of beliefs. By utilizing this property, we can treat the environment as a POMDP, where we use the history to capture features from the short-term cycle phases. To combine the POMDP environment with Deep Reinforcement Learning, we use an LSTM to capture the long-short term dependencies to only carry the important long-term information through the environment, while still capturing the important short-term patterns. We use the LSTM hidden and cell state to carry on the features through the cycles, together with the general weights of the network to capture the long-term dependencies.

We take inspiration from other papers using a DQN in a POMDP control setting, and propose two strategies for environmental sampling and training which are customized to properly capture cycle phases. While all papers to our knowledge perform LSTM unrolling from the DQN replay memory during training, we claim that unrolling sequential hidden states in training is the same as sampling through the environment and zero the hidden state for a hyperparametric amount of steps. In this way, the model trains faster and is able to cope with recurrent model staleness in the environment.

We compare ReLight to both current conventional and Reinforcement Learning solutions, and the results show that ReLight performs better than the current state-of-the-art models without the need for an additional forecasting layer or storing excessive information. Hidden Chain Reset also performs on the same level as K-epoch unrolling, which confirms our theory of combating recurrent staleness in the environment performs the same while training significantly faster while still retaining the long-short term spatial-temporal patterns.

References

[1] M. Beedham, 'See the true environmental cost of inner-city congestion with TomTom Traffic Index', 2022.

[2] N. McCarthy, F. Richter, 'Infographic: Congestion costs U.S. cities billions every year', 2020.

[3] H. Wei, G. Zheng, V. V. Gayah, Z. Li, 'A Survey on Traffic Signal Control Methods', *{CoRR}* abs/1904.08117 (2019).

[4] X. Hu, C. Zhao, G. Wang, 'A Traffic Light Dynamic Control Algorithm with Deep Reinforcement Learning Based on GNN Prediction', *CoRR* abs/2009.14627 (2020).

ReLight: Capturing spatial-temporal context in Road Traffic Signal Control using recurrency in POMDPs

Peter Kjaer (20164808) and Samuel Alexander Vall Andersen (20175548)

***Abstract*—Traffic congestion in urban areas is a problem for the environment and the economy. One solution to minimize congestion is optimizing traffic lights. Traffic signal control is a challenging problem due to the complex traffic flow patterns. Conventional traffic control use pre-coded cycle pattern plans, which suffer from adapting to the complex flow dynamics. Reinforcement Learning allows for dynamic control but is unable to properly catch temporal-feature due to the Markov property. To solve this, recent papers propose incorporating prediction modules into Reinforcement Learning control, however, this suffers from additional loss and generalization.**

To circumvent these issues, we propose Recurrent Light (ReLight), which treats the environment as a Partially Observable Markov Decision Process which depends on the history of previous belief states. We utilize this dependency to capture spatial-temporal features and utilize an LSTM in the DQN network to capture important long-short term features through hidden states. To properly capture cycle phases, we propose two sampling and two training strategies. In our experiments, we demonstrate that ReLight outperforms state-of-the-art models on one, multi and city-wide datasets.

I. Introduction

Cities worldwide are seeing an influx of rising traffic congestion, and it has only been growing. The average U.S driver spends more than 290 hours behind the wheel each year, whereas people aged 30-49 drive on average 13,506 miles annually [1]. It is inevitable that the world's population is growing, thus adding more vehicles to the road. Traffic congestion harm both the environment and the economy. Based on TomTom Traffic Index [2], traffic congestion and inefficient traffic pollute about 2.2 megatonnes of CO2 emissions, adding up to 15% of the total CO2 usage yearly in London. One way to mitigate or avoid the immense CO2 emissions caused by traffic is to keep traffic moving and limit the number of vehicles in major cities. If we keep traffic moving and decrease the travel time, cars would spend less time on the road and emit less CO2. One fundamental way to increase flow throughout large cities is to optimize road traffic lights. Conventional traffic light control uses pre-defined fixed cycle length plans which is pre-programmed and controlled by an algorithm [3]. A cycle-phase is a cycle-based signal plan intersection used for different time sequences to allow for most vehicle throughput optimally through the intersection. Each traffic cycle lasts for a pre-defined length and is usually repeated in similar phase sequences throughout the day [4]. The fixed plans are not adaptive enough to cope with today's complex traffic and increasing flow.

Peter Kjaer, Samuel Alexander Vall Andersen are with Aalborg University, Denmark, e-mail: pkjar16@student.aau.dk, e-mail: sava17@student.aau.dk

One way to solve the problem of dynamic adjustment for traffic light control is to use Reinforcement Learning techniques. [[PressLight - 2019] [5], [CoLight - 2019] [6], [IntelliLight - 2018] [7]] have introduced the concept of Reinforcement Learning to control the traffic light sequence based on radar or video feed data. These papers have shown that Reinforcement Learning has a superior performance over traditional traffic light sequences since it learns directly from observing the environment and reinforces the algorithm towards the best possible actions.

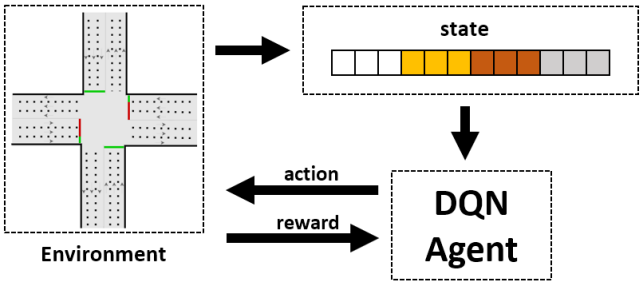

Fig. 1. Classic Deep Reinforcement Learning framework for Road Traffic Light Control. Each color in the state represent each inbound lanes.

To convert a traffic light into a Reinforcement Learning environment, we can use the radar or video feed data to infer the number of vehicles [4]. Figure 1 illustrates a traffic light in a Reinforcement Learning setting. The state-space consists of all inbound lanes towards the intersection, where each of the lanes represents the number of vehicles on each lane. Combined with Deep Reinforcement Learning algorithms like DQN, the agent learns to generalize the traffic features. The agent takes the state input and decides whether to keep the current light phase or change the lights. The agent then takes this action in the next step in the environment, where the environment outputs an immediate reward and a next state. With the camera or radar information, the reward function could be calculated using elements like Queue Length, Waiting time, or Speed [4].

By counting the vehicles in each lane over time, we can create a time series of the traffic flow for the entire intersection. Converting environmental information into a time-series problem, the agent can use the temporal features from the state over time to infer the best action. Figure 2 illustrates how state information can represent multivariate time-series for each lane in an intersection.

GPlight [Hu et al. 2020] [8] incorporates future short-term traffic flow prediction through time-series forecasting to

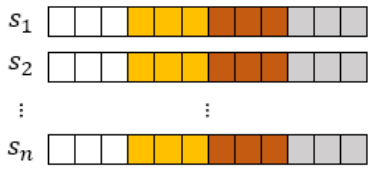

Fig. 2. States (s) over time can be represented as a time-series problem.

allow the agent to make better decisions based on the spatial movement of traffic flow. In this way, the agent may prepare in advance for rush hour traffic based on the knowledge from the multivariate forecasting features. By implementing a forecasting module alongside the Reinforcement Learning agent, however, have some fundamental issues :

1) **The loss from the time-series forecasting module may lead to inaccurate predictions, which makes the agent take the wrong actions.** Suppose the time-series forecaster does not train properly due to the lack of training data or if the patterns are too sporadic. In that case, the loss from the forecaster may feed inaccurate information to the Reinforcement Learning agent. Inaccurate information may cause the Reinforcement Learning agent to take wrong actions, where the agent may end up in states which do not infer optimal convergence. Societal and environmental effects like the 2020 Corona Pandemic, roadworks, and other factors may change the current traffic flow, making forecasting a continuous adaptive process. The forecasting module may rarely achieve highly accurate convergence since the target is constantly moving in sporadic intervals.
2) **It is rarely reasonable to assume that the causal pattern for some time sequence always matches the same cycle pattern for the specific intersection.** It can be easy to assume a causal pattern between each weekday and the same cycle phases sequencing through each weekday throughout the week. In reality, however, this may not always be the case in most scenarios. The causal pattern may not always be correctly identified and may not always match the current time sequence for the specific intersection. It is reasonable to assume that traffic within a specific hour is affected by many other factors like highway accidents, events, and events happening far away, affecting long commuters. When training with time-series data, the causal patterns within a specific cycle may not accurately match the actual cycle of that specific time sequence. An example could be a rush hour cycle of 8:00 AM - 9:00 AM, where the traffic moves around with almost the same flow throughout Monday-Wednesday. In case of an accident on a motorway on Thursday, this flow may be delayed or even receive more traffic pressure. It is reasonable to assume that no day is the same, thus assuming the causal patterns will not always match the proper cycle every time.
3) **Reinforcement Learning algorithms will adapt to a particular flow density, whereas in natural traffic environments, traffic densities will change constantly and unevenly [9].** The time-series forecasting module may lead to the agent converging towards the learned forecasting model rather than the cycle dynamics. If a forecasting model samples data for one year and uses this data to forecast the following year, the traffic patterns may not look the same or be completely different. It is essential to keep Reinforcement Learning from non-bias estimations like a forecasting agent since we want to learn Spatio-temporal patterns while still being able to adapt to the current cycle phase.

The control process in Reinforcement Learning is typically a Markov Decision Process (MDP). An MDP consists of a set of states, actions, probabilistic transition matrix, and immediate reward function of the actions [10]. One way to solve an MDP control process is to create an algorithm to find the best policy, which specifies the best actions to take for each state [11]. An iterative algorithm iterates through the environment, finding the best path based on its variant. One common variant is Value Iteration [Bellman 1957] [12], which uses the Bellman Equation to compute the optimal policy and its value at each state. One of the major constraints of MDPs is called the Markov Property. The Markov Property is where the future depends only on the present state and does not depend on history [13]. This constraint creates a time-series problem that is difficult to incorporate into Reinforcement Learning. To incorporate time-series forecasting in an MDP environment, it is necessary to create an additional module that feeds information alongside the state since the state can not depend on the history of previous states. To circumvent the Markov Property, we must look for a new approach to get the spatial-temporal features for each cycle without an additional forecasting module.

We propose Recurrent Light (ReLight), a new approach to Reinforcement Learning traffic light control, where we treat the environment as a Partially Observable Markov Decision Process (POMDP). POMDP is a generalization of MDPs, where POMDP assumes the system dynamics of an MDP; however, it can only observe partial or no information from the state environment. Where MDPs map state observations to actions, POMDP maps the history of observations (belief states) to the action [10]. The idea behind POMDP is to obscure the full observability of the current state, remove all certainty that the agent knows its current state is, and select an action based on the history and the current belief state. The underlying dynamics of the POMDP are still markovian. However, the agent must take action based on the belief state composed of the previous beliefs, which makes POMDP unique and a non-markovian problem [10]. The POMDP problem can be latently inferred as a time-series problem since it keeps track of the spatial-temporal features through time by keeping track of the history of previous obscured beliefs. Where papers using MDP for Reinforcement Learning like [Mnih et al. - 2015] [14] use four sequential observations to create a state to capture the direction and movement of elements in the environment. To partially obscure this environment to convert this problem to a POMDP control problem, the state could be separate,

only providing a single frame instead of a stack frame for the agent, or elements from the frames could be obscured [15]. To convert the traffic environment into a POMDP, we look into the possible state configurations for a traffic environment. The possible metrics for an MDP state could be the number of vehicles on the road, traffic movement, how fast each car is going, headway, queue length, or neighboring intersections. To provide minimal information, where the agent still can infer some information from the environment, we use the vehicle count per lane in a single intersection. The agent does not get the full state information, while LSTM can utilize this information and treat it like a time-series problem.

Previous papers [PressLight [16], FRAP [17], IntelliLight [18]] already run their state space in this configuration, with some getting this information from neighbouring traffic lights as well. Compared to the information available from the emulator, we claim that these papers are trying to run an MDP control process on minimal viable information of what is provided by the environment. We aim to learn from spatial-temporal features through time with hidden states using a history of belief states within the POMDP. This history of hidden states infer latent context from an LSTM, which takes the current observation together with previous hidden and cell state as input, and produces a new hidden and cell state. We pass the spatial-temporal features from the long and short-term traffic features through the environment by iteratively passing the previous hidden state to an LSTM and outputting the next hidden state. We aim to capture the traffic cycle phases and their latent context by passing on the hidden state.

To solve the inaccuracy of the forecasting module, we remove the need for an additional forecasting module altogether by converting the environment to a POMDP. By utilizing the POMDP features, we remove the direct need to infer time-series forecasting and only rely on the built-in latent features from the LSTM. Using a recurrent model will accurately sequence the time-series spatial features to replace the need for an additional forecasting module, thus still being Markovian.

The use of a forecasting module allows the forecasting module to catch general patterns from the data used for training the module. This data could be adapted continuously, thus continuously updating the gradients for the forecasting module during training. One problem, however, is the adaption to the short-term patterns of cycle phases. Cycle phases dynamically change for each time sequence and are heavily dependent on the surrounding factors like roadworks, queues, accidents, etc. Depending on conditions, the cycle phase can change from every one hour to every 5 minutes. To adapt to these rapidly changing and developing phases promptly, we propose to our knowledge; a novel training strategy called Hidden Chain Reset. This training strategy zeros the hidden state carried onward in the environment and tried to reconstruct the hidden state based on the state input and the weighted features from the LSTM. In this way, the agent learns to reconstruct the hidden state based on its current belief and the history of all previous beliefs. By rolling a few hidden and belief states through the neural network, the hidden state will be able to adapt to the current cycle phase, thus increasing its accuracy for each unique environment. One of the Hidden Chain Reset's major contributing features is its capability to prevent recurrent state staleness, thus retaining optimal performance. Hidden Chain Reset provides a significant lower convergence and training time compared to current methods like unrolling during training.

To prevent the agent from adapting to a particular density flow, we utilize the features from the LSTM unit in the DQN, to ensure capturing the long-term and short-term dependencies while only retaining the important information. It is important for the agent to learn the long-term patterns of a weekend and weekday together with a rush and night hours while still retaining the flexibility of the small cycle phases within those larger cycle phases. The forget gate of the LSTM retains only the important information, discarding the rest to the output hidden state. In this way, the agent does not learn a particular pattern but also weighs the short-term patterns in its decisions.

We take inspiration from previous POMDP Reinforcement Learning papers [[Hausknecht et al. 2015][15], [Kapturowski et al. 2015][19]], and modify these strategies for efficient cycle sampling. For ReLight, we propose two environment sampling strategies (Hidden Chain Reset and Episode-based Reset) and two training strategies (Random Sampling and Burn-in K-Unrolling). With these strategies, we aim for the agent to learn to reconstruct states for each cycle phase to capture spatial-temporal sequences adequately.

II. Related work

This section will first introduce works on conventional traffic light control, then works on Reinforcement Learning integrated into traffic light control, and lastly conclude with POMDP works.

A. Conventional Traffic Light Control

Today, many modern cities rely on traffic signal control systems like SCATS [20] and SCOOT [21], where both systems are designed and rely on manually crafted traffic signal panels [22]. The pre-crafted traffic plans are then selected by the current traffic volume detected by loop sensors or other detection tools for an intersection. These systems, however, cannot perceive and react to real-time traffic patterns [22]. Greenwave [23] and Maxband [24] generate cycle-based signal plans for the individual intersection and aims to optimize the offsets to reduce the number of vehicles stopping (calculating speed from sensors) in one direction. Greenwave only optimizes for unidirectional traffic and requires all intersections to share the same cycle, which is only optimal if all intersections share the same traffic patterns and cycles [22]. Actuated Control [25] and Self-organizing Traffic Light Control (SOTL) [9] dynamically change the next traffic phase in real-time according to pre-defined rules and data. SOTL claims that Reinforcement Learning algorithms adapt towards a particular flow density, while the real-world traffic density changes constantly and is uneven [9]. Actuated Control and SOTL, however, use a static ruleset for controlling the traffic phases, making it non-adaptive to find traffic patterns and prepare lanes in advance for high or low-density traffic [22].

B. Reinforcement Learning Traffic Light Control

Reinforcement Learning-based algorithms like PressLight [16], FRAP [17], IntelliLight [18] all use the Deep Q-Network (DQN) Reinforcement Learning algorithm to achieve impressive results, outperforming all previous methods on multiple datasets for a both multi- and single intersections. One way to achieve coordination and traffic control between multiple intersections is to train one agent which jointly takes action for all intersections; however, it has trouble learning from the dimensions of the action space. [[Chu et al. 2019] [26], [Tantawy et al. 2013] [27], [Wiering 2013] [28]] propose to train Reinforcement Learning agents separately for each intersection and use neighboring information to train the agent. However, these methods concentrate information from all intersections together and treat each with equal importance, which leads to a flawed design since [Wei et al. 2019] [29] argues that upstream intersections could have a more significant influence than downstream intersections. [Wei et al. 2019] [29] address this issue by leveraging the attention mechanism to learn different weights of all neighboring intersections to influence their importance. This method, however, still does not extend to oncoming and outgoing traffic from highways and any oncoming traffic outside the neighboring traffic light state. While non-Reinforcement Learning-based algorithms use cycle phases of weekday and holiday settings, their adaptive capabilities are limited to specific ruleset or preprogrammed plans. Current Reinforcement Learning-based algorithms use flow density and Max pressure [30] to optimize the algorithm to get the lowest travel time; however, they ignore the cycle phases from weekdays, nights, and holidays. Recent works [Hu et al. 2020] [8] solve this problem with an graph neural network module to predict the future short-term traffic at each intersection. The forecasting module is then combined with the Reinforcement Learning algorithm for the agent to decide based on the traffic density. This method, however, adds an additional loss function to the traffic volume forecasting module. Realistic traffic flow is typically very indeterministic and random, making time-series forecasting in this domain challenging and inaccurate. If the loss of the forecasting module is high, the Reinforcement Learning agent may be fed inaccurate data, unnecessarily feeding the Reinforcement Learning agent wrong information about the environment.

C. Partially observable Markov decision process

Both [Hausknecht et. al 2015] [15] and [Narasimhan et. al 2015] [31] use temporal-difference update to train a DQN agent by jointly training both convolutional and LSTM layers, while still achieving impressive results in Atari2600 and text-based fantasy games.

For an agent to perform well in a POMPD environment, an Reinforcement Learning agent requires a state representation that encodes information from its state-action trajectory through the environment, which is used as additional information to its current state observation. One of the common ways to get this representation is to use a Recurrent Neural Network (RNN) [32], which captures features through a temporal sequence. However, RNNs suffer from a vanishing gradient problem when the recurrent sequence gets too big, forgetting information early in the recurrent chain. [Hochreiter & Schmidhuber, 1997] [33] proposes the Long Short Term Memory (LSTM) architecture to solve this problem, which allows the network to remember inputs over a long period and capture important features from all parts of the recurrent chain. The hidden states of the LSTM architecture is then used as a part of the agent's state encoding.

To train an RNN agent directly from replay memory while allowing it to capture relevant long-term dependencies, requires whole state-action trajectories to be stored in the agents replay memory and used for updating the networks gradients. [Hausknecht et al. 2015] [15] compare two strategies for training an LSTM architecture directly from replay memory.

1) At the beginning of the sampled sequence, the agent initializes a zero start state of hidden states to initialize the network, which fills during environment sampling. An entire episode is then randomly selected from the replay memory, where the update starts from the beginning of the sampled episode to the end.
2) At the beginning of the sampled sequence, the agent initializes a zero start state of hidden states to initialize the network, which fills during environment sampling. The agent then randomly selects a point within an episode and samples a pre-defined sequence from this point that is unrolled and selected for gradient updating.

Where storing and replaying an entire trajectory may seem like a simple task, it may become infeasible for very long episodes. Very long episodes may also suffer from recurrent state staleness since the agent only performs infrequent updates on the same episodes with a new network. [Kapturowski et al.2019] [19] argue that zeroing the initial start state allows the recurrent network to learn to recover meaningful predictions from an initial recurrent state mismatch. Since a zeroed initial state is not a part of the recurrent chain, it has to rely on the network's weights to re-create the hidden state. It may, however, limit the network's ability to rely on the initial state and fail to capture some long-term dependencies. [Hausknecht et al. 2015] [15] results observed an insignificant difference between the two strategies. To solve these problems, [Kapturowski et al. 2019] [19] propose two new strategies for training the network:

1) **Stored State** - The recurrent state is stored in replay memory during the environment sampling and used to initialize the network during gradient updating.
2) **Burn-in** - During gradient updating, the network will randomly sample a sequence from replay memory, where the sample will get a 'burn-in period'. The network will use a portion of the sampled sequence for unrolling the network to produce a start state, where the network will be updated on the remaining part of the sequence.

Our paper defines the environment as a Partially Observable MDP to remove the Markovian property constraints and uses LSTM to capture long, short-term spatiotemporal features from each lane at all intersections and isolate the effects of

the recurrence properties. By leveraging the benefit of carrying the hidden state in the environment, our model can capture the features from the temporal sequence of the traffic flow without additional modules. Our contribution also leverages POMDPs and LSTMs to use faster sampling and training strategies, which converge faster and, in most cases, better than other state-of-the-art methods.

III. Preliminaries

A. Partially Observable Markov Decision Process

Partially Observable Markov Decision Process (POMDP) differs from the original Markov Decision Process (MDP) [10] by the state space S not being fully observable from the environment. Instead of the agent receiving full information about a state from the environment, the agent only receives an observation as an indicator for the actual state in the system. We can define POMDPs formally as a 6-tuple $(S,\ A,\ T,\ R,\ \mathcal{Z},\ O)$ where:

- S, A, T, R is the states, actions, transitions and rewards as in MDPs.
- $\mathcal{Z}$ is the observation space. Set of observations.
- O is the observation model. Contains conditional observation probabilities.

Since we do not get the full information from the environment, the agent has a new task of approximating or reconstructing the environment. It does so by updating its belief state $b(s)$ while it interacts with the environment [34]. A belief state is a probability of the agent being in state s according to its previous history of actions and observations [35]. In a POMDP environment, the agent does not have direct access to the current state of the environment, so taking a decision requires keeping track of the history of the process, which makes a POMDP a non-Markovian problem. Non-Markovian problems like POMDPs often requires substantial domain knowledge to define a set of hidden states and observation probabilities [36]. One way to circumvent this requirement is with deep learning. Deep learning can be applied to represent and track hidden states without a substantial amount of domain knowledge [36]. In works by [[Narasimhan et al. - 2015] [37], [Hausknecht & Stone - 2015] [15] and [Bakker - 2002] [38]], RNN or LSTM is used to represent Q-functions. These works propose two models for combining recurrent neural networks with Reinforcement Learning. The LSTM architecture should have a significant advantage for non-Markovian problems, since the current belief state is depended on the previous history of actions and observations compared to the feed-foward network.

IV. Problem Definition

This section will go through the notation of our environment. All symbols can be looked up in Table I. We define an intersection in the environment as:

$$l_1, l_2..l_n \in rl$$
$$rl_1, rl_2 \ldots rl_n \in I$$

[1]Since CityFlow is designed on the American traffic model, all right turn traffic will always be green, thus not generating an action.

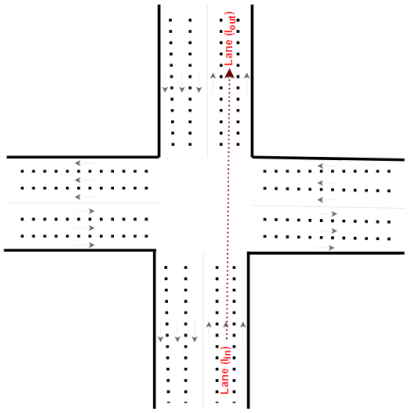

Fig. 3. State design from an 3x4 intersection with an action space of 8[1].

where a road link rl have l lanes inside an intersection I. The lanes can be inbound l_{in} , entering the environment towards the intersection, and outbound l_{out}, having passed the intersection and exiting the environment. Vehicles spawn from l_{in} lane and drive towards one of the l_{out} lanes based on its pre-scheduled route. Once a vehicle has passed the intersection and entered an l_{out} lane, the vehicle should no longer be considered a part of the state. The vehicle is then passed to another intersection, where the next Reinforcement Learning agent will consider it as its l_{in} , and the vehicle will be passed to the new Reinforcement Learning agents respective state space. Figure 3 visualizes the environment with 3 inbound and 3 outgoing lanes, in all four directions. l_{in} represents a distinct traffic movement from an inbound lane towards an outgoing lane l_{out}.

TABLE I
Symbols and Notation

Symbol	Description
E	Environment
$i \in I$	Intersection
rl	Road Link
$l \in i$	Lane
l_{in}	Inbound Lane
l_{out}	Outbound Lane
s	State
a	Action
r	Reward
q	Q-value
t	Timestep
f	Flow
TT	Travel Time
m	Vehicle count on a specific lane

We represent the state space as an array with the size $(1 \times N)$, where N is the number of lanes $l_1, l_2..l_n \in I$ in the environment. In the intersection on Figure 3, the number of inbound roads has 3 lanes x 4 roads, which results in a state size of (1×12). We count the number of vehicles currently on the respective lane for each lane in the state representation. We can count vehicles with equation 4.1.b.

$$l_{count} = m(l)(t) \tag{4.1.b}$$

where m is a function that counts the number of vehicles on a

given lane l at timestep t. One timestep unit in the environment is represented as one second. If there is a high count for a lane in the environment, the headway is minimized, thus creating higher traffic queues and delays.

$$\sum_{t=1}^{t_{\max}} l_{count}(\forall l_{in} \in I, t) \tag{4.1.c}$$

We can calculate the total amount of vehicles in the environment with 4.1.c, where t_{max} is the maximum time to simulate the environment in seconds. When using realistic datasets, equation 4.1.c must be equal or close to the daily ADDT for that intersection for t_{max} for one day.
We can define the flow (or spawn interval) of the cars of each lane as a function f where one car enters l_{in} and exits through l_{out} as equation 4.1.d at timestep t.

$$flow = f\left(l_{\text{in}}, l_{\text{out}}\right)(t) \tag{4.1.d}$$

We can extend 4.1.d to get the total flow (4.1.e) of the intersection within a given simulation time t_{max}. A higher flow value on a low t_{max} generates more pressure on the intersection. Flow is commonly low at night and off-peak hours at most traffic distributions.

$$\sum_{t=1}^{t_{\max}} flow((\forall l_{in} \in I, \forall l_{out} \in I), t) \tag{4.1.e}$$

This state representation shows how well the Reinforcement Learning agent controls vehicles through the traffic light. A higher vehicle count in each lane means a long queue, which results in a low traffic flow. The agent can use this state representation to see how many vehicles are in each lane and take the appropriate action to maximize traffic flow.

One common way to evaluate the traffic situation in the environment is through the average travel time. We can define the travel time (4.1.f) for one vehicle by the time the vehicles enters l_{in} and exits l_{out}.

$$TT = (l_{out})(t) - (l_{in})(t) \tag{4.1.f}$$

We can then use travel time to find the average travel time (4.1.g) for the environment.

$$\frac{1}{t_{\max}}\left(\sum_{t=1}^{t_{\max}} TT((\forall l_{in} \in I, \forall l_{out} \in I), t)\right) \tag{4.1.g}$$

When the environment contains multiple intersections, 4.1.c, 4.1.e, 4.1.g is extended to $I_1, I_2..I_n \in E$, where I is each intersection in the environment E. Each Reinforcement Learning agent can only see information for their respective intersection; however, this measurement allows us to measure the interpolation of nearby intersections and compare results with other multi-intersection papers.

V. Method

A. Framework Overview

Since the MDP environment requires comprehensive observation of the state (like in Atari games, where you can observe the movement of 4 frames [14]), POMDP only requires partial state information. In our environment, a state observes the number of vehicles in each lane at the observation time. To isolate the effects of recurrency, we combine LSTM with Deep Reinforcement Learning to better capture long-short term temporal patterns between traffic cycle phases. We utilize the hidden and cell states to construct a tuple of a single hidden state from the output of one iteration through the LSTM cell. We use the LSTM to unroll the history of previously observed belief states for the agent to guess the current belief state where the agent is inside the environment. Recurrent controllers benefit from robustness against missing information, even trained with complete state information [15]. Since the number of vehicles on each lane is sparse, information from the environment can assist the agent in reconstructing the hidden state based on the history of previous hidden states. By only providing the agent state information of the number of vehicles at each lane, the LSTM benefits from the latent temporal features from the state observation and uses this to learn temporal patterns. The goal of the LSTM is to learn features for the specific intersection cycles with the short-term patterns while still retaining patterns from the general traffic pattern.

Figure 4 visualizes the environment sampling process. At the beginning of each episode, we sample an initial observation from the environment and a zero-initialized hidden state containing a tuple of a hidden and cell state. The DQN network (visualized on Figure 5) then inputs the initial observation together with the hidden state and produces a next hidden state and a Q-value for each action. The agent performs the action with the highest Q-value in the environment and produces the following state observation with an immediate reward. The agent then samples $(s, s_{next}, a, r, hidden, hidden_{next})$ to the replay buffer and input the next state observation together with the next hidden state into the DQN network to continue the value iteration algorithm. The replay buffer stores the following information, which is used for training:

1) s is the current environment observation, containing the number of vehicles per lane at a specific time.
2) a is the action sampled by the DQN network (Figure 5)
3) r is the environment reward. Our paper measures the reward function by the waiting count of vehicles in each lane. Longer queues will output less reward to the agent.
4) $hidden$ is the current hidden state
5) $hidden_{next}$ is the next hidden state sampled together with the action from the DQN network.
6) s_{next} is the net environment observation after an action is taken.

B. Sampling Strategy

We propose ReLight that extends [Kapturowski et al. 2019] [19] two strategies, Stored State and Burn-in, which aims to eliminate recurrent state staleness without storing large trajectories while maintaining long-term dependencies. Our goal with the new modifications is to capture cycle phases within the environment and use the hidden state to carry the temporal features for each phase. To capture relevant long-term dependencies in a traffic environment, we must encapsulate the phase cycles and learn about each of the cycles and their

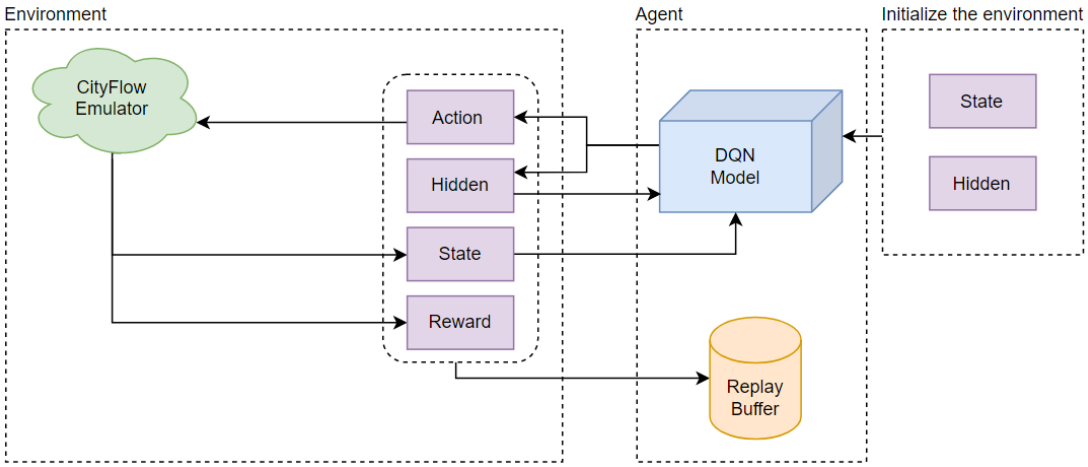

Fig. 4. POMDP Reinforcement Learning overview

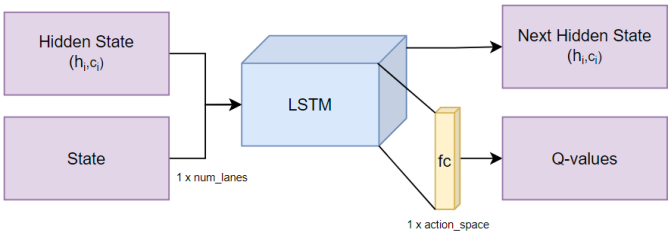

Fig. 5. DQN model with an LSTM. The model takes the current observation as input together with a tuple consisting of a hidden and cell state.

differences. The night cycle with no to low traffic consists of half of the cycles for an entire day, and most traffic patterns expects rush hour cycles in the morning and the afternoon. An overview of where each element is applied can be found on Algorithm 1. We propose two strategies for environment sampling:

1) **Hidden Chain Reset** - We propose a novel hyperparameter addition to capture the cycles phases adequately. Instead of zeroing the hidden state at the start of each episode, we zero the hidden state during an episode at every hyperparametric amount of steps. By zeroing the hidden state during environment sampling, the agent tries to re-create the hidden state from the spatial-temporal cycle to increase correctness and combat the recurrent state staleness by old model parameters.
2) **Episode-based Reset** The second strategy is to zero the hidden state at the beginning of an episode and then pass the hidden state onwards through the entire episode. The hidden state should capture the features from an entire day (or defined episode length) instead of short cycles within a day.

An episode can be a pre-defined hour or an entire day/week for the traffic environment, and each step within the environment is a second. For an environment spanning one hour (3600 steps), the agent zeros the hidden state 14 times an hour with a hidden chain reset of 256.

At the beginning of an episode, we initialize an empty

Algorithm 1 ReLight Algorithm

Initialize replay memory D to capacity N
Initialize Q-function Q with random weights θ
Initialize target Q-function $\hat{Q}$ with weights $\theta^{-} = \theta$
for episode = 1, M **do**
  Initialize step number t to be 0 and T as episode length
  Zero initialize hidden state h_t
  while $t < T$ **do**
    if with probability ϵ **then**
      Sample random action a_t and h_{t+1}.
    else
      Sample $a_t, h_{t+1} = argmax_a Q(s_t, h_t, a; \theta)$
    end if
    Execute action a_t and observe r_t, s_{t+1}
    Store transition $(s_t, a_t, r_t, s_{t+1}, h_t, h_{t+1})$ in D
    if Burn-In K-Unrolling **then**
      Sample J sequential transitions of
        $(s_j, a_j, r_j, s_{j+1}, h_j, h_{j+1})$ from D
      Zero initialize first transition with $h_j = 0$
      Update $h_{j+1} = Q(s_j, h_j; \theta)$ for burn-in steps
      Update next transition $h_j = h_{j+1}$
      Update $h_{j+1} = Q(s_j, h_j; \theta)$ for the remaining
        sequence and only pass on this part
      Update next transition $h_j = h_{j+1}$
    else
      Sample a random minibatch of
        $(s_j, a_j, r_j, s_{j+1}, h_j, h_{j+1})$ from D
    end if
    $Set\ y_j = \begin{cases} r_j & if\ t = T\ at\ step\ j+1 \\ r_j + \gamma\ max_a \widehat{Q}(s_{j+1}, h_{j+1}, a`; \theta^{-}) \end{cases}$
    Update gradients w.r.t $\mathcal{L} = (y_j - Q(s_j, h_j, a_j; \theta))^2$
    Update $h_t = h_{t+1}$ and $s_t = s_{t+1}$
    if Hidden State Reset **then**
      Every G-steps $h_t = 0$
    end if
  end while
end for

hidden state $hidden = 0$. The hidden state contains a tuple of:

$$hidden = (h_t, c_t)$$

where:

- h_t is the hidden state
- c_t is the cell state

The agent then either samples a random action (depending on epsilon greedy or initial random sampling from [14]), or infers an action and a hidden state from the DQN network:

$$q, hidden_{next} = DQN_\theta(s, hidden)$$

where:

- q is an array with all the action values (also referred to as Q-values). We select the action by $a = argmax(q)$.
- $hidden_{next}$ is the next hidden state
- DQN is the Deep Q-Network subjected to some parameter θ with the state and hidden state as input.

The agent then executes the action in the environment and take an environment step to generate the next state and an immediate reward. The $(s, a, r, s_{next}, hidden, hidden_{next})$ is then appended to the replay buffer, together with updating the hidden state and state for the next environment step:

$$hidden = hidden_{next}$$
$$state = state_{next}$$

During training, we inspire by [Kapturowski et al. 2019] [19] two proposed strategies. Since we use a strategy to prevent recurrent state staleness in the environment sampling, it should not be required to store large sequences in the replay memory for iterative unrolling during training. By utilizing Hidden Chain Reset in the environment while carrying the hidden state during the environment step, we can create an update strategy to avoid storing sequential transitions in the replay buffer and perform sequential unrolling updates altogether. We claim that there is no difference in managing the hidden chain in environmental sampling or training, as long as the hidden and cell state is passed throughout the environment, carrying their parameters.

C. Training Strategy

We propose two different strategies for training:

1) **Random Sampling** - In random sampling, the model samples a batch size amount of transitions from the replay memory and performs a gradient update at each transition inside the randomly selected batch. The random sampling does adhere to the DQN random sampling policy; however, it relies on the hidden states inferred directly from the replay memory, which may be prone to recurrent state staleness even when using Hidden Chain Reset in the environment. Finding a good hyperparameter for the Hidden Chain Reset is essential for random sampling. If the hidden state is zeroed too frequently in the environment, the LSTM may have trouble learning long-term dependencies. In contrast, we may suffer from staleness and capturing dependencies not relevant to the current cycle if it is too big.

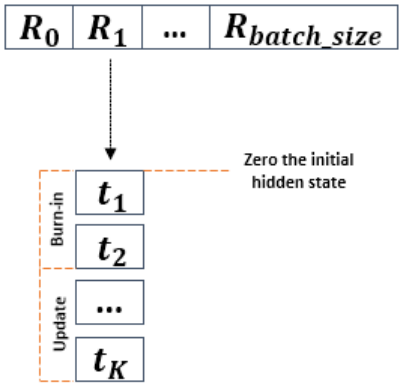

Fig. 6. Unrolling from replay memory, where t is a transition and R is an element randomly sampled from the replay buffer.

2) **Burn-In K-Unrolling** - In this strategy, the environment samples a hyperparametric K amount of sequential transitions and stores the sequential transitions into one element in replay memory. When the agent trains, the agent samples a batch size of sequential transitions and performs hidden-state unrolling on each of the sequential transitions starting with a zero start state and a hyperparametric burn-in phase before updating the gradient on the remaining part of the sequence (Figure 6). One of the significant benefits of this update strategy is that all updates happen with recently updated model parameters, which is prone to eliminate recurrent state staleness. This strategy may become a problem for long-term dependencies for a traffic environment since it is computationally costly to save a large sequence inside a replay memory as a single element. This model also only partially adheres to DQN's random sampling policy since it updates on the sequential transitions after the burn-in period. While the starting point is random, the transitions are unrolled in sequential order from the starting point.

VI. Evaluation

During evaluation we will try to answer the following questions:

- **(Q1)** How do the different sampling and training strategies affect the traffic environment and experimental results?
- **(Q2)** How does Hidden Chain Reset strategy compare to the unrolling strategy?
- **(Q3)** How does our proposed method perform compared with other state-of-the-art methods?
- **(Q4)** Is ReLight scalable enough to control a city-level traffic signals without neighbouring information?

We will first introduce the datasets and the methodology behind training. We will then answer each question in their independent section. Each question is answered in a section highlighted by the header.

TABLE II
BEST HYPERPARAMETER RESULTS FOR WEEKDAY (WD) AND WEEKEND (WE) ON GRANNY (LEFT) AND ROSA (RIGHT) IN AVERAGE TRAVEL TIME (SECONDS). BEST RESULTS HIGHLIGHTED IN BOLD.

Dataset	Param	Size	1h	3h	6h	12h
WD	Chain Length	1800	**166.43**	664.89	1413.55	**2497.84**
WE			**75.21**	**81.11**	139.92	762.94
WD	Hidden Dims	30	168.32	**654.25**	**1411.46**	2524.64
WE			75.33	81.51	**143.52**	**769.95**
WD	Burn-in	30	**165.04**	662.43	**1379.97**	2493.90
WE			75.94	**81.88**	**134.09**	682.00
WD	Kepochs	160	**161.93**	664.63	1388.25	**2392.54**
WE			**77.29**	82.35	**125.39**	**661.10**

Dataset	Param	Size	1h	3h	6h	12h
WD	Chain Length	3600	58.44	**177.31**	**143.05**	**113.90**
WE			57.06	58.18	**58.902**	**65.80**
WD	Hidden Dims	256	**58.24**	185.61	**139.86**	**114.80**
WE			**57.05**	**58.09**	**58.83**	65.79
WD	Burn-in	30	58.61	189.36	163.71	**124.90**
WE			**56.81**	**58.27**	59.057	**67.54**
WD	Kepochs	30	**58.64**	184.77	**155.90**	**127.35**
WE			**56.87**	**58.33**	59.03	68.21

A. Datasets

In this paper, we evaluate two different types of datasets. The first type is actual intersection data fetched from different cities in the United States and China. We use the city-wide New York 16x3, Hangzhou 4x4 and Jinan 3x4 datasets available from the CoLight Github[2]. These datasets only have a length of one hour. Since we need to measure cycle phases and how the models work in datasets longer than one hour, we need to generate realistic datasets with more hours.

To generate as realistic datasets longer than one hour as possible, we fetch real-world traffic Annual Average Daily Traffic(ADDT) from the Tennessee Transportation Data Management System [39]. We then fetch four ADDT for each road leading from/to an intersection in an urban environment to represent the traffic volume for one intersection. We select four intersections in Tennessee, United States to generate a flow and road network file:

- **Church:** Church St and Easley St
- **Granny:** Granny White Pike and Harding Pl / Battery Ln
- **Old Hickory:** Edmondson Pike and Nolensville Pike
- **Rosa:** Rosa L Parks Boulevard

Church, Granny, and Old Hickory consist of two intersections in each representative dataset, whereas Rosa only consists of one intersection. Since ADDT does not tell anything about the distribution of cars during a day, we calculate a flow probability distribution (Equation 7.1) on the traffic flow on the highway 20EB Bridge St in California, extracted from the California Department of Transportation (PeMS) database [40]. We use the PeMS dataset for distribution since, to our knowledge is the only database where we can find authentic time-series data of traffic flow over long periods. Since the PeMS does not include intersections, we can only use this dataset to approximate flow distribution.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{7.1}$$

We use the flow distribution from Wednesdays to represent a weekday and Sundays to represent weekends. To make a non-static distribution across datasets, we utilize the random binomial distribution (Equation 7.2) with the extracted probability distribution to generate unique weeks for each intersection.

$$P(N) = \left(\frac{n}{N}\right) p^N (1-p)^{n-N} \tag{7.2}$$

Our generated datasets start from 06:00 AM and span 1,3,6 and 12 hours since night traffic provides a low travel time for all hours. Figure 7 shows the vehicle distribution for a weekday and weekend and their different time cycles. Starting from 06:00 AM, the datasets will slowly move towards the 08:00 AM rush hour, which is the busiest cycle of the dataset. Weekday distribution show most people go to work around 8:00 AM and get off work at different periods around 3-4:00 PM. The weekend distribution shows no rush hour in the morning; however, the busiest time is around 12-2:00 PM, when most stores are still open. Table III infer the vehicle volume per hour, where mean is the mean average of vehicles driving through each respective intersection per hour.

TABLE III
DATASET STATISTICS FOR BOTH WEEKDAYS (WD) AND WEEKENDS (WE)

Dataset	Mean	Std	Min	Max
Rosa WD	3609.72	298.07	575.00	1336
Rosa WE	2405.13	299.22	125.66	1123.99
Granny WD	4946.79	452.07	323.09	2046.45
Granny WE	3279.45	326.1	124.74	1560.27
Church WD	1573.28	147.45	103.39	570.87
Church WE	1051.71	108.25	40.32	454.8
Old Hickory WD	5602.91	330.48	390.91	1556.97
Old Hickory WE	3743.48	267.15	156.8	1217.24

B. Evaluation Methodology

We simulate the traffic environment in the city-scale reinforcement-learning simulator CityFlow [41]. CityFlow is 20 times faster than other previous state-of-the-art emulators like SUMO and can support simulations of multiple intersections in the same environment [41].

Before comparing ReLight to other state-of-the-art models, we first optimize hyperparameters for the framework. We evaluate hyperparameters and other models with 1, 3, 6, and 12 hours in traffic intervals on an Nvidia Tesla V100 for 350 episodes.

To train the ReLight hyperparameters, we first want to learn the best hyperparameters for the environment without the

[2]https://github.com/wingsweihua/colight/tree/master/data

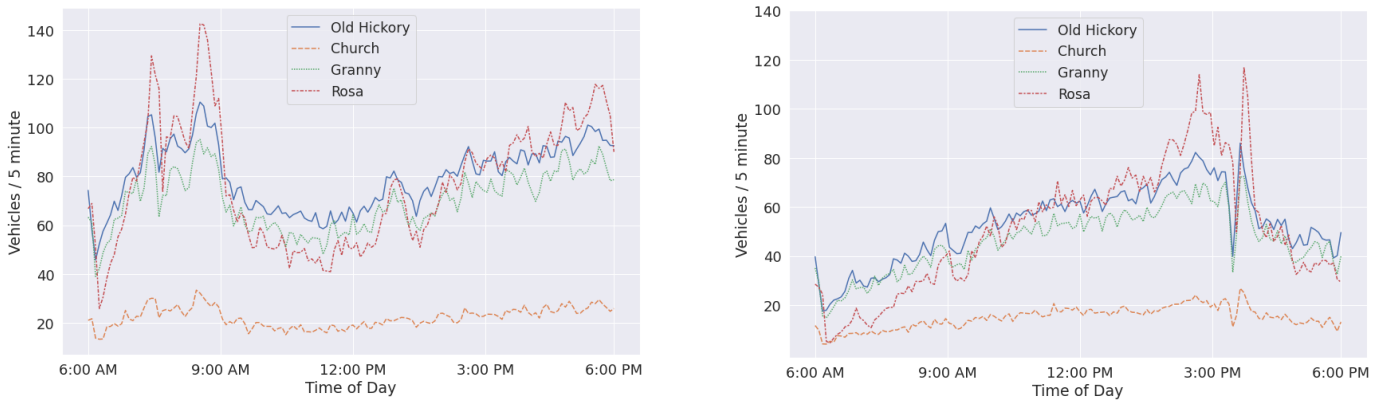

Fig. 7. Distribution for vehicles / 5 minutes for the weekday (left) and weekend (right)

TABLE IV
AVERAGE EPISODE TRAINING TIME MEASURED IN SECONDS ON THE ROSA DATASET BETWEEN 6:00-9:00 AM

	DQN	Colight	PressLight	ReLight	+Unrolling
WD	18.06	59.36	16.13	19.39	53.54
WE	14.37	43.40	12.49	14.94	48.87

unrolling strategy. We do this to fairly compare the training strategy on the same environment hyperparameters to get the best comparison. We prepare general baselines for hidden dimensions and hidden resets when training the environment, where both are 256, respectively. When we found the best hyperparameters for the environment, we used them for the two training strategies and the different hyperparameters within these strategies. When we found the hyperparameters, we used them to train to Relight on our four datasets and three large-scale one-hour datasets available on Github[3]. We then compare the results with both other conventional and Reinforcement Learning solutions.

C. Hyperparameters (Q1 + Q2)

In this paper, we propose two strategies for environmental sampling (Hidden Reset and Entire Trajectory) and training (Random and Unrolling). Entire Trajectory and Random sampling do not have any hyperparameters, and we can directly use them to compare the two other strategies.

1) **Hidden Reset** defines the frequency the hidden state gets zeroed to prevent recurrent state staleness and enhance the capabilities for the LSTM to capture cycle phases. We propose four different reset parameters (30, 256, 1800, 3600), where each parameter is in seconds. With an empirical analysis, we default our parameters 256 during hyperparameter training.
2) **Hidden dimension** is the dimensionality (unit parameter) of the output space for the LSTM layer. This hyperparameter defines the number of hidden units in the hidden layer. We propose three different hidden units (30, 256, 512), with the default parameter 256.
3) **Unrolling** defines the number of sequential transitions unrolled during training. [Kapturowski et al. 2019] [19] propose an unrolling length of 80 in their paper. We test with three different hyperparameters (30, 80 160). Unrolling lengths of 160 and larger are very slow since the model must sequentially infer 160 transitions each training step, which can be computational demanding.
4) **Burn-in** is a proposed extension of [Kapturowski et al. 2019] [19] strategy, where the agent gets to infer a new hidden state for a burn-in number of transitions before passing the remaining part of the unrolling sequence to training. [Kapturowski et al. 2019] [19] propose a burn-in number of 20. To mitigate the loss wasted from this extension, we decided to keep burn-in at 20 for all unrolling steps.

We present the results from the hyperparameter experiments in Table II. We measure the results in average travel time measured in seconds and highlight the best results in bold. We propose the following hyperparameters for one and multi-intersection:

1) **One Intersection**: We benchmark on the Rosa L Parks Blvd dataset for one intersection. In most cases, chain length got the best results with a hidden state to reset every one hour. The optimal hidden units are very clearly 256 units. One general observation of this dataset is that the unrolling and burn-in strategy performs worse overall than all the random sampling strategy parameters.
2) **Two Intersections**: We benchmark on the Granny White Pike and Harding Pl / Battery Ln dataset for multi-intersection. The best parameter is resetting the hidden state every 30 minutes, and the best hidden unit is 30. A general observation compared to the one-intersection is that unrolling with and without burn-in outperforms random sampling by a very slight margin. Figure IV shows the time taken for one episode compared across the different strategies and other models. Based on the time comparison for one epoch compared to the performance of the unrolling strategy, we will highly propose and use the Hidden Reset with the random sampling strategies instead, even though the results are minimally better for unrolling on multiple intersections.

[3]https://github.com/wingsweihua/colight/tree/master/data

TABLE V
OVERVIEW OF THE RESULTS FOR WEEKDAYS (WD) AND WEEKENDS (WE) IN TWO-INTERSECTION DATASETS. THE RESULTS IS IN AVERAGE TRAVEL TIME (SECONDS), WHERE THE BEST RESULTS IS HIGHLIGHTED IN BOLD.

Model		Granny 1h	Granny 3h	Granny 6h	Granny 12h	Church 1h	Church 3h	Church 6h	Church 12h	Old Hickory 1h	Old Hickory 3h	Old Hickory 6h	Old Hickory 12h
DQN	WD	598.21	2088.46	4706.60	10919.17	181.31	312.58	517.03	2501.46	179.29	527.60	824.02	3259.89
	WE	299.32	1034.74	2445.01	7981.25	183.76	191.95	244.97	4248.79	145.15	162.39	248.61	3186.29
SOTL	WD	380	1340.36	2989.40	5503.85	167.00	149.00	159.00	156.00	91.60	131.00	120.00	121.00
	WE	158.09	264.01	847.99	2987.02	286.00	246.00	211.00	194.00	138.00	119.00	106.00	100.00
Colight	WD	425.59	1378.68	3099.50	6318.06	437.23	1270.55	2837.02	5298.23	350.85	974.85	2164.69	4030.73
	WE	388.77	1057.35	2040.02	4576.03	480.70	1263.06	2387.72	5065.07	361.95	943.67	1786.04	3807.83
PressLight	WD	223.33	769.46	1947.22	2786.65	68.80	71.00	70.27	70.66	67.28	68.94	68.76	69.02
	WE	76.69	85.82	230.83	1182.62	68.09	67.66	69.74	69.88	65.79	67.18	67.93	68.24
ReLight	WD	**168.66**	**668.31**	**1415.38**	**2515.67**	**67.63**	**70.49**	**69.86**	**70.45**	**67.20**	**68.69**	**68.66**	**68.88**
	WE	**75.35**	**81.50**	**136.99**	**718.75**	**64.41**	**65.81**	**67.26**	**68.43**	**65.09**	**66.76**	**67.51**	**68.08**

TABLE VI
OVERVIEW OF THE RESULTS FOR WEEKDAYS (WD) AND WEEKENDS (WE) ON CITY-WIDE DATASETS. THE RESULTS IS IN AVERAGE TRAVEL TIME (SECONDS), WHERE THE BEST RESULTS IS HIGHLIGHTED IN BOLD.

	New York	Jinan	Hangzhou
DQN	1541.00	571.96	655.23
SOTL	1775.35	365.00	533.17
CoLight	361.49	347.42	338.85
PressLight	189.90	297.68	326.84
ReLight	**181.25**	**286.58**	**319.72**

TABLE VII
OVERVIEW OF THE RESULTS FOR WEEKDAYS (WD) AND WEEKENDS (WE) ON SINGLE-INTERSECTION DATASETS. THE RESULTS IS IN AVERAGE TRAVEL TIME (SECONDS), WHERE THE BEST RESULTS IS HIGHLIGHTED IN BOLD.

Rosa	Dataset	1h	3h	6h	12h
DQN	WD	**58.34**	247.48	215.24	168.05
	WE	**56.47**	**58.16**	59.03	**71.00**
SOTL	WD	118.00	566.00	723.00	608.00
	WE	181.00	126.00	139.00	248.00
CoLight	WD	277.40	847.11	2039.25	3505.60
	WE	282.85	713.95	1340.52	3126.02
PressLight	WD	58.51	269.15	240.19	188.63
	WE	58.54	261.21	261.48	209.86
ReLight	WD	58.44	**177.31**	**143.05**	**113.90**
	WE	57.06	58.18	**58.90**	118.16

D. Baselines and Metrics

We compare the proposed ReLight framework with other traditional and state-of-the-art methods. [4].

- **DQN:** A standard DQN [14] approach applied to traffic light intersections.
- **Actuated Control - SOTL:** SOTL [9] is a conventional traffic light control algorithm which uses a request-based phase control system to optimize traffic. If the number of vehicles in a queue at a lane surpasses a specified threshold, the traffic light receives a request to change light phases. If there are no requests for the current phase and a minimum green light phase has passed, the request is granted, and the signal changes. If there is still a request for the current phase due to traffic, the light signal will only change after passing a duration threshold.
- **CoLight:** Colight [6] use Graph Attentional Networks to enable cooperation between traffic signals in a reinforcement learning algorithm. They incorporate the temporal and spatial influences of neighboring intersections to the target intersection and build up index-free modeling of neighboring intersections.
- **PressLight:** Presslight [16] incorporates the state-of-the-art method max pressure in a reinforcement learning algorithm. In Max-pressure, the objective is to balance queue length between neighboring intersections by minimizing the "pressure" of the phases for an intersection. The pressure of a movement signal can be defined as the number of vehicles on incoming lanes minus the number of vehicles on the corresponding outgoing lanes.

[4]Since IntelliLight is using SUMO as an environment simulator, we omit this method from the results to compare other methods with the same datasets and environments properly.

E. Performance Comparison (Q3)

In this section, we evaluate the effectiveness of the different strategies while comparing ReLight to other traffic light control methods. The average training accuracy is shown in Table VII, Table V, and Table VI respectively.

For a single intersection (Rosa - Table VII), ReLight compares significantly better on all hour intervals than all other models. DQN and PressLight both compare in result for one hour, since it reaches the minimum time it takes for a car to drive through the environment. One thing to note is that 12 hours have a lower average travel time than 3 and 6 hours since the intersection got time to clear the rush hour queue and average the pressure with the additional low-pressure cycles. All models perform well on one hour but adding additional hours with more pressure, all models start to get larger travel times, however, DQN, Presslight, and Relight keep the travel time low. We expect Colight to perform badly on single and two intersection datasets like Rosa and Granny since their model relies solely on the graph attention mechanism [6] for

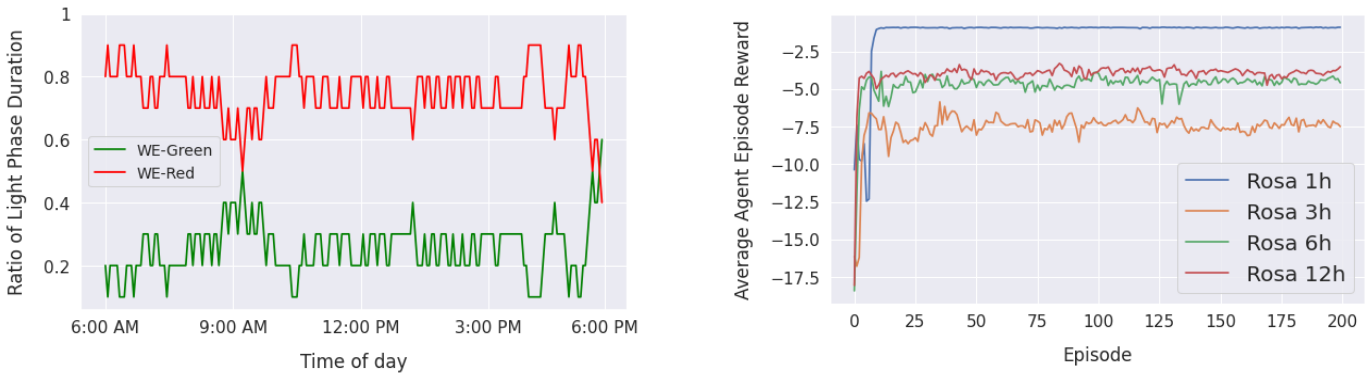

Fig. 8. Average time duration of two light phases (Green and Red) for West to East traffic on the Rosa weekday dataset from a converged policy (left) and episode reward for each episode for different time-periods for the Rosa weekday dataset (right)

each intersection. Figure 8 (right) shows the average episode reward for the four periods on the Rosa weekday dataset. For 1 hour, the agent gets the most reward from the environment since it is a period outside rush hour with low traffic. Both 6 and 12-hour both contain the same amount of rush and slow cycles, so they converge around the same environment reward. 3 hour is the most challenging period for the agent since it only sees the first hour with slow traffic and never alleviate traffic from the rush hour cycles. All hours converge to their periods' full convergence fast, with only a slight deviation in episode rewards later during training.

For two intersection datasets (Granny, Church, and Old Hickory - Table V), Relight significantly outperforms all other models on Granny, and is close but better for Church and Old Hickory than Presslight since the traffic pressure is low on both datasets. Overall it is clear that conventional traffic light control does not adapt well to pressure, and standard DQN-based algorithms perform well, but still fall short of ReLight.

In City-wide one-hour datasets (New York, Jinan, and Hangzhou - Table VI), Colight is comparably closer to Presslight and ReLight. Conventional control comes closer to the other results, however, still falls short on all datasets. Our ReLight model also slightly outperforms all other models on these datasets.

F. Spatial and Temporal distribution in Multi and City-Wide Datasets (Q4)

One of the most important parts of our contribution is that we can achieve get spatial-temporal information from neighboring intersections without directly inferring any information from these intersections. In PressLight and CoLight, both use independent methods to calculate network-level information to better predict pressure for each intersection. ReLight uses solely independent agents and single-intersection information and uses the latent context of spatial-temporal information from the LSTM features of the incoming traffic from other neighboring intersections. Figure 8 (left) shows the average time duration of two light phases (Red and Green for WE traffic) on the Rosa weekday dataset. This figure shows that traffic in the WE direction becomes green for longer during the rush hour cycles to alleviate traffic. Comparing the agent converged policy with the dataset distribution in Figure 7, it is clear that the agent learned to adapt its cycle phases to the specific flow for the individual intersection. For all multi and city-wide traffic datasets, ReLight outperforms other models in one to 48 intersections (New York), which indicates that ReLight may capture spatial-temporal information in some sense from neighboring intersections to properly coordinate city-wide traffic.

VII. Conclusion

In this paper, we propose a novel algorithm ReLight, where we treat the environment as a POMDP and use its properties to capture the short history of vehicle flow while using LSTM hidden state properties to capture spatial-temporal features across cycle phases. We conduct our experiments with real-world intersections for 1 to 12 hours and capture their different cycle phases through the morning, evening, and afternoon hours, and one-hour city-wide datasets. ReLight outperforms other state-of-the-art traffic control models in both single, multi, and city-wide intersections. In this paper, we only test the model with vehicle count state information, and for future work, we will suggest either adding or removing more state information or additional modules to the environment to experiment to compare the results. In future work, we also need to consider patterns of roadworks, accidents, and sudden high pressure together with applying our solution to other simulators or a real intersection, and try to capture patterns for an entire week instead of two separate days within a week.

References

[1] "Americans Spend an Average of 17,600 Minutes Driving Each Year," 03 2022. [Online]. Available: https://newsroom.aaa.com/2016/09/americans-spend-average-17600-minutes-driving-year/

[2] "The True Environmental Cost of Inner-city Congestion | TomTom Blog." [Online]. Available: https://www.tomtom.com/blog/traffic-and-travel-information/the-true-environmental-cost-of-inner-city-congestion/

[3] "Traffic Signal Timing Manual: Chapter 5 - Office of Operations." [Online]. Available: https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter5.htm#5.2

[4] H. Wei, G. Zheng, V. V. Gayah, and Z. Li, "A Survey on Traffic Signal Control Methods," *CoRR*, vol. abs/1904.08117, 2019.

[5] H. Wei, C. Chen, G. Zheng, K. Wu, V. Gayah, K. Xu, and Z. Li, "PressLight: Learning Max Pressure Control to Coordinate Traffic Signals in Arterial Network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1290–1298. [Online]. Available: 10.1145/3292500.3330949

[6] H. Wei, N. Xu, H. Zhang, G. Zheng, X. Zang, C. Chen, W. Zhang, Y. Zhu, K. Xu, and Z. Li, "CoLight: Learning Network-level Cooperation for Traffic Signal Control," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[7] H. Wei, G. Zheng, H. Yao, and Z. Li, "IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 2496–2505. [Online]. Available: 10.1145/3219819.3220096

[8] X. Hu, C. Zhao, and G. Wang, "A traffic light dynamic control algorithm with deep reinforcement learning based on GNN prediction," *CoRR*, vol. abs/2009.14627, 2020. [Online]. Available: https://arxiv.org/abs/2009.14627

[9] C. Gershenson, *Self-organizing Traffic Lights*. Complex Systems, 16 (2005) 29–53; 2005 Complex Systems Publications, Inc., 2005.

[10] A. R. Cassandra, "Background on pomdps." [Online]. Available: https://cs.brown.edu/research/ai/pomdp/tutorial/pomdp-background.html

[11] ——, "Brief introduction to markov decision processes (mdps)." [Online]. Available: https://cs.brown.edu/research/ai/pomdp/tutorial/mdp.html

[12] R. BELLMAN, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[13] F. Grabski, "1 - discrete state space markov processes," in *Semi-Markov Processes: Applications in System Reliability and Maintenance*, F. Grabski, Ed. Elsevier, 2015, pp. 1–17. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128005187000016

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[15] M. J. Hausknecht and P. Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs," *CoRR*, vol. abs/1507.06527, 2015.

[16] H. Wei, C. Chen, G. Zheng, K. Wu, V. Gayah, K. Xu, and Z. Li, "Presslight: Learning max pressure control to coordinate traffic signals in arterial network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1290–1298. [Online]. Available: https://doi.org/10.1145/3292500.3330949

[17] G. Zheng, Y. Xiong, X. Zang, J. Feng, H. Wei, H. Zhang, Y. Li, K. Xu, and Z. Li, "Learning phase competition for traffic signal control," *CoRR*, vol. abs/1905.04722, 2019. [Online]. Available: http://arxiv.org/abs/1905.04722

[18] H. Wei, G. Zheng, H. Yao, and Z. Li, "Intellilight: A reinforcement learning approach for intelligent traffic light control," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2496–2505. [Online]. Available: https://doi.org/10.1145/3219819.3220096

[19] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos, "Recurrent experience replay in distributed reinforcement learning," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=r1lyTjAqYX

[20] P. Lowrie, Roads, and T. A. of New South Wales. Traffic Control Section, *SCATS, Sydney Co-Ordinated Adaptive Traffic System: A Traffic Responsive Method of Controlling Urban Traffic*. Roads and Traffic Authority NSW, Traffic Control Section, 1990. [Online]. Available: https://books.google.dk/books?id=V4PTtgAACAAJ

[21] R. B. PB Hunt, DI Robertson and M. C. Royle, *The SCOOT on-line traffic signal optimisation technique*. Traffic Engineering Control 23, 4 (1982)., 1982.

[22] H. Wei, G. Zheng, V. V. Gayah, and Z. Li, "A survey on traffic signal control methods," *CoRR*, vol. abs/1904.08117, 2019. [Online]. Available: http://arxiv.org/abs/1904.08117

[23] E. S. P. Roger P Roess and W. R. McShane., *Traffic engineering*. Pearson, 2004.

[24] M. D. K. John DC Little and N. H. Gartner, *MAXBAND: A versatile program for setting signals on arteries and triangular networks*. Transportation Research Record Issue Number: 795, 1981.

[25] U. D. of Transportation, "Traffic signal timing manual - chapter 5." [Online]. Available: https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter5.htm

[26] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *CoRR*, vol. abs/1903.04527, 2019. [Online]. Available: http://arxiv.org/abs/1903.04527

[27] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): Methodology and large-scale application on downtown toronto," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1140–1150, 2013.

[28] M. Wiering, "Multi-agent reinforcement learning for traffic light control," 2000.

[29] H. Wei, N. Xu, H. Zhang, G. Zheng, X. Zang, C. Chen, W. Zhang, Y. Zhu, K. Xu, and Z. Li, "Colight: Learning network-level cooperation for traffic signal control," *CoRR*, vol. abs/1905.05717, 2019. [Online]. Available: http://arxiv.org/abs/1905.05717

[30] P. Varaiya, "Max pressure control of a network of signalized intersections," *Transportation Research Part C: Emerging Technologies*, vol. 36, p. 177–195, 11 2013.

[31] X. Li, L. Li, J. Gao, X. He, J. Chen, l. Deng, and J. He, "Recurrent reinforcement learning: A hybrid approach," 09 2015.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," p. 533–536, Oct 1986. [Online]. Available: http://dx.doi.org/10.1038/323533a0

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[34] M. Egorov, "Deep Reinforcement Learning with POMDPs," 11 2015. [Online]. Available: https://cs229.stanford.edu/proj2015/363_report.pdf

[35] H. A. M. Schafer, "Reinforcement Learning with Recurrent Neural Networks," 10 2008. [Online]. Available: https://d-nb.info/991415167/34

[36] X. Li *et al.*, "Recurrent Reinforcement Learning: A Hybrid Approach." [Online]. Available: https://arxiv.org/abs/1509.03044

[37] K. Narasimhan, T. D. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," *CoRR*, vol. abs/1506.08941, 2015. [Online]. Available: http://arxiv.org/abs/1506.08941

[38] B. Bakker, "Reinforcement learning with long short-term memory," in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14. MIT Press, 2001. [Online]. Available: https://proceedings.neurips.cc/paper/2001/file/a38b16173474ba8b1a95bcbc30d3b8a5-Paper.pdf

[39] T. D. of Transportation, "Annual average daily traffic (aadt) maps." [Online]. Available: https://www.tn.gov/tdot/driver-how-do-i/look-at-or-order-state-maps/maps/annual-average-daily-traffic-maps.html

[40] C. D. of Transportation, "Caltrans Performance Measurement System." [Online]. Available: https://pems.dot.ca.gov/

[41] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. Li, "Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario," *CoRR*, vol. abs/1905.05217, 2019. [Online]. Available: http://arxiv.org/abs/1905.05217