Learning a Safe Driving Policy for Urban Area from Real-world Data

Adil Mahmud[1] and Changcheng Huang[2]

***Abstract*— Driving safely on the urban roads is a major impediment in achieving level 5 autonomy. To achieve this, two main streams of approaches have been proposed: module-based and end-to-end. Module based solutions try to solve the problem by dividing the whole task of driving into separate modules and solving each one at a time. On the other hand, end-to-end approaches try to provide the control command directly from the sensor data input, like what a human driver does. Deep reinforcement learning (DRL) is one of the algorithm families that has received much attention recently to achieve end-to-end solutions. As this approach is challenging, almost all the related works use simulator generated data for training a policy network. However, synthetic data does not capture the complexity, variability, realism, and diversity of the real-world environment. A reinforcement learning (RL) policy trained on synthetic dataset necessarily makes it unreliable in real-world deployment. In this study, we propose an actor-critic DRL model to learn a driving policy from a real-world urban driving dataset. The policy enables the RL agent to keep safe distance from the leading vehicle, follow traffic light, and prevents the agent from going off-road. To optimize the policy we use proximal policy optimization (PPO), a state-of-the-art reinforcement learning algorithm. Simulation results show that the agent learns some of the basic safe driving requirements effectively.**

I. Introduction

Autonomous or semi-autonomous driving has attracted the research community, promising safer, cost-effective, and scalable transportation[1]. Despite advancements, urban settings remain a challenging focus of research due to complex multi-agent dynamics, traffic rules, map topology, and the need for scalability to new environments. While most of the solutions to the driving tasks focus on achieving a particular goal, which is quite plausible, grossly, they all fall short of scaling to unknown or new environments [2]. This is the reason why most of the leading self-driving car manufacturing companies aim for collecting as much data as possible in new environments to train neural networks. It might solve the generalization problem to some extent, but it is costly and may not offer a comprehensive solution for broader challenges.

The autonomous driving software systems can be categorized into two general groups: modular and end-to-end. Modular system divides the task into a pipeline of separate constituent modules that links sensory input to the control output. Though the modular approach has been the focus in the research community because of its flexibility and adaptability, the end-to-end approach has seen a recent surge due to its simplicity [1], [2], [3], [4].

To facilitate end-to-end driving, both imitation learning (IL) and deep reinforcement learning (DRL) algorithms have shown great performances. IL algorithms aim to mimic human expertise by learning the way human driver drives from real world data [5]. This is a very promising approach but may work poorly in unknown environments, especially while driving on roads with different statistics or geometry compared to the dataset the model was trained on. We argue that DRL can be a better candidate in this regard because, with DRL, an agent learns the best action to take in a situation, commonly known as environment in reinforcement learning (RL) domain. This is very similar to a human driver where the driver takes the next action based on what he/she sees around. Human driver can subconsciously comprehend the surroundings and take driving decision (accelerate or brake, steer straight, left or right). If the environment can be comprehensively described, DRL will be able to learn the best action to take as well.

The initial demonstrations that used DRL to solve driving task in an end-to-end manner have shown very encouraging results [6], [7]. The authors in [7] are pioneers in applying DRL to learn a driving policy, particularly focusing on the lane following task. They used a single monocular image as input to a convolutional neural network (CNN) feature extractor which gave a feature vector conveying a comprehensive understanding of the immediate environment. This feature vector was provided to an actor-critic algorithm for learning a lane following policy. Reference [8] achieved the same goal of lane keeping as that of [7] but on a car simulator for racing called TORCS. Reference [9] focused on lane change strategies by using proximal policy optimization (PPO), a popular deep reinforcement learning algorithm. Similarly, [10] proposed a Deep Q Network (DQN) based lane change decision making strategy. It can be noticed from the discussion that all these works focused on a specific driving goal instead of achieving a comprehensive and generalized driving task on the road.

End-to-end approach is very appealing because of its simplicity compared to modular approach, but it faces challenges due to high dimensional inputs and unnecessary information in the raw sensor data. Therefore, recent studies have preferred intermediate representation of data, *e.g.*, perception module's output, instead of raw sensor data [3], [11], [12], [13], [14], [15], [16]. However, all the mentioned works, except [13], use simulator generated data for policy learning, which could present challenges when implementing these

[1]A. Mahmud is with the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada, adilroman at cmail.carleton.ca

[2]C. Huang is with the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada, changcheng.huang at carleton.ca

policies in real-world driving scenarios.

In this work we use Woven Planet Level 5 prediction dataset[17], a large real-world urban driving dataset, to train a driving policy by using PPO, an actor-critic DRL algorithm. We vectorize the perceived environment and then transform the representation into corresponding graph network. Finally, we apply multi-head attention to the intended agent to extract a feature vector that would encompass the interactions of the agent with the surrounding vehicles and traffic elements. The following points summarizes our contributions:

- This is one of very few works that tried real-world urban driving dataset to train an RL algorithm for learning driving policy.
- We developed a reward function incorporating simplified yet critical safe driving criteria, such as maintaining safe distance and target speed, preventing collisions, and obeying traffic signals, to evaluate naturalistic driving actions taken by an RL agent, such as acceleration, braking, and steering movements, within the environment.
- We developed a simulator on top of the closed loop simulator (simulated agent follows the predicted path instead of the ground truth) provided by Woven Planet (*Lyft*) [17] to incorporate all the vehicles for simulation. Thus, each vehicle in a scene is a potential simulation agent. This approach increases the number of training samples to a greater magnitude.

II. Learning Policy from Real-world Data

Policy learning with DRL is an error and trial process where the agent takes an action and the environment replies with a reward. A key question in learning a driving policy with DRL is whether it should be learned in a real-world driving setting or a simulated environment. The prior option is the most effective because the policy will be learned directly on the road where the agent will drive eventually. A very good example to support this claim can be found in [7], where an agent successfully learns to follow a lane in a day of training on an empty road. However, real-world driving involves interacting with other road-participants. Deploying such a learned policy to a road with other participants would be catastrophic. The only viable solution is to train a policy in a simulated environment to avoid risks during the learning process. However, transferring the learned policy from simulated to real world environment, know as sim-to-real transfer, appears as another challenge[3], [4]. The challenge is posed due to the significant differences in the dynamics and characteristics between a real-world and a simulated environment.

We argue that learning a driving policy from a real-world dataset, instead of simulated data, has the potential to significantly mitigate sim-to-real transfer problems. In fact, simulated dataset helps build a robust policy by exposing the agent to a richer and more exploratory data distribution so that it can tackle as many situations as possible, but it lacks in capturing complexity and nuances of the real-world. As a result, a policy trained on simulated data has the potential to a degraded performance in real-world environment. In contrast, real-world data captures the dynamics of the real-world environment and gives the flexibility of error and trial learning if the policy is learned in a simulator. If a vehicle is chosen from the dataset and allowed to simulate in a closed-loop setting while following the RL framework, a policy could be learned in the presence of real-world road-users. By adhering to standard traffic rules, the agent will be able to learn to navigate safely within the natural context.

III. Actor-critic Reinforcement Learning with PPO

Reinforcement learning enables agents to learn optimal actions by interacting with the environment to maximize cumulative rewards. Actor-critic RL is one of the variants of RL that has received much attention recently. It is a family of RL algorithms that use both value (critic) and policy-based (actor) methods. While the policy network learns to provide the optimal actions, the value network helps measure how good an action is. Proximal policy optimization PPO [18] uses an actor-critic approach and has proven its efficiency in handling complex environments while also being simple in deployment. In this work we use PPO for policy learning.

PPO is known for its capability of balancing sample efficiency, stability, and ease of implementation. It proposes a clipped surrogate loss function that constraints the policy from being changed too much compared to the previous policy during training. This is achieved by using a simple loss function as below, which is defined as [18]

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \quad (1)$$

where θ represents parameters of the policy function, $\hat{E}[...]$ represents expectation over a finite batch of trajectories, $r_t(\theta)$ is the ratio of the probability of taking action a_t at state s_t in the current policy divided by the old policy, as shown below:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \quad (2)$$

$\hat{A}_t$ is the advantage function, which is a measure of how much better or worse an action is compared to the average action in a given state. The PPO algorithm uses fixed-length trajectory segment and a truncated version of generalized advantage estimation. This gives the following equation of advantage function:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \quad (3)$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, γ is discount factor, and λ is a hyperparameter that controls the trade-off between bias and variance in the estimate. The clipped objective function in (1) can be further augmented by adding an entropy bonus term to ensure sufficient exploration. Thus, the objective function becomes

$$L_t^{CLIP+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) + c_2 S_{\pi_\theta}(s_t)] \quad (4)$$

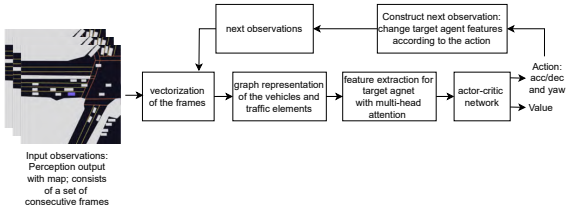

Fig. 1. Schematic diagram of state-action-next state loop of the proposed DRL system. A frame refers to an observed scene at a time, capturing information from the road-participants and the road itself.

where π_θ is a stochastic RL policy and S denotes the entropy of the policy distribution π_θ evaluated at state s_t. Furthermore, if the actor and critic neural networks share parameters, the loss function should combine the loss surrogate and a value function (represents expected cumulative reward) error term, which is reflected in the following equation:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S_{\pi_\theta}(s_t)] \quad (5)$$

where c_1 and c_2 are coefficients, L_t^{VF} is the squared-error loss of the value function $(V_\theta(s_t) - V_t^{targ})^2$. For the complete PPO algorithm, we suggest to consult reference [18].

IV. System Design

This section provides details of our proposed DRL system. In a typical DRL system, an agent follows a policy to take actions based on its current observation. The environment provides a reward and a subsequent observation based on the action taken. We start by providing a schematic diagram in Fig. 1 that shows how a state-action-next state loop works in our case to set up a context for the detailed explanations on state, actions, and reward. The process starts by taking a group of instantaneous intermediate representations, referred to as frames in the dataset's terminology or observations in the context of reinforcement learning. These representations depict the scene surrounding the vehicle under consideration, known as the *target agent*. Inspired by the works in [19] and [20], the intermediate representations are vectorized, and then the traffic elements, such as vehicles, lane-lines, crosswalks are transformed into graph structures. Subsequently, a feature vector is generated by using multi-head attention to capture interactions between the *target agent* and the surrounding vehicles along with the traffic control elements, such as traffic light status and crosswalks. This is how some real-world observations are transformed into a RL state that will be easier to be interpreted by the actor-critic network in the next step. The actor network provides actions, which are acceleration or deceleration, and yaw. On the other hand, the value network assesses the performance of the actor by indicating how favorable or unfavorable its actions are. The following subsections provide descriptions on the state, action and reward of the proposed DRL system.

A. State

This subsection illustrates the steps to prepare the state for the proposed DRL system.

- **Vectorization** The representation of a state vector at time t involves observation at time t and its three predecessors, represented by $O_{t-3}, O_{t-2}, O_{t-1}, O_t$. Each observation includes vehicles, and traffic elements like lanes, traffic light status (red, green, or yellow), and crosswalks. To vectorize these observations, we adopt a method similar to [20], using points instead of vectors as in [19]. Lane-line, crosswalk boundary lines, and locations of vehicles are represented by coordinate points.
  Table I shows the list of features that describes each element. The velocity of the *target agent* or any other vehicle can be obtained from their locations in consecutive observations, if they are not provided in the dataset. For example, to obtain the velocity of the target agent at time $t-1$, v_{t-1}, we take the difference of its positions in observations O_{t-3} and O_{t-2} and divide it by the time difference between the two observations. As a result, for each vehicle state we get velocities for three observations: O_t, O_{t-1}, and O_{t-2}. Since the locations of the *target agent* and other elements are known, finding distances between the *target agent* and the traffic element points and other vehicles are straightforward. While representing the crosswalks and lane-mid lines, we limit the number of points to 20, but this number can be varied. The left side of Fig. 2 illustrates a vectorized representation of a sequence of observations.
- **Graph representation** Similar to the approach presented in [20], we transform the vector representation to a graph representation. The number of nodes in the graph of an element is equal to the number of points the element has in its vector representation. Accordingly, each vehicles has three nodes, and the crosswalks and lane-mid lines have twenty nodes each (equal to the number of points used to represent the lines). These graphs are called sub-graphs, which are in turn part of a global graph as shown in Fig. 2. The global graph is used to represent the interactions among the elements in the scene. In the final step, we apply multi-head attention to extract a feature vector for the *target agent* that is expected to hold the latent interaction features with the traffic participants and the traffic control elements.

TABLE I
Elements and their features

Element	Features
Target agent	`velocity_x` (V_x), `velocity_y` (V_y), yaw (Y)
Other vehicles	`velocity_x`, `velocity_y`, Euclidean distance to *target agent*
Lanes	Dx, Dy: Distances (in x and y direction) between points on the mid-line and the *target agent*
Crosswalks	Dx, Dy: Distances (in x and y direction) between points on the crosswalk boundaries and the *target agent*
Traffic light	Traffic light status for each lane

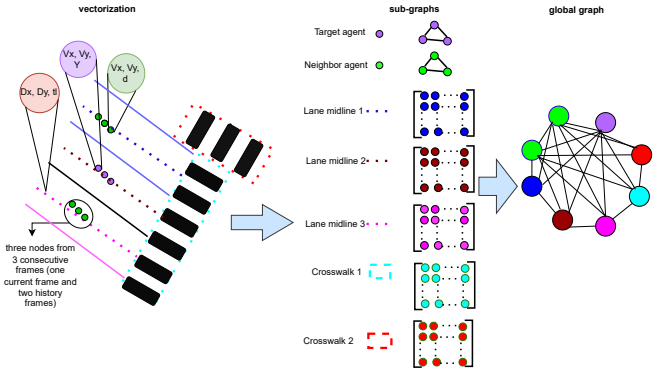

Fig. 2. Vectorization and graph representation of a frame. On the left, each element is represented by a set of coordinate points, where the feature set for an element is provided inside a circle. In the middle, each vehicle and road elements are presented as sub-graph, and on the right all the elements and vehicles form a fully connected global graph. The global graph models higher-order interactions.

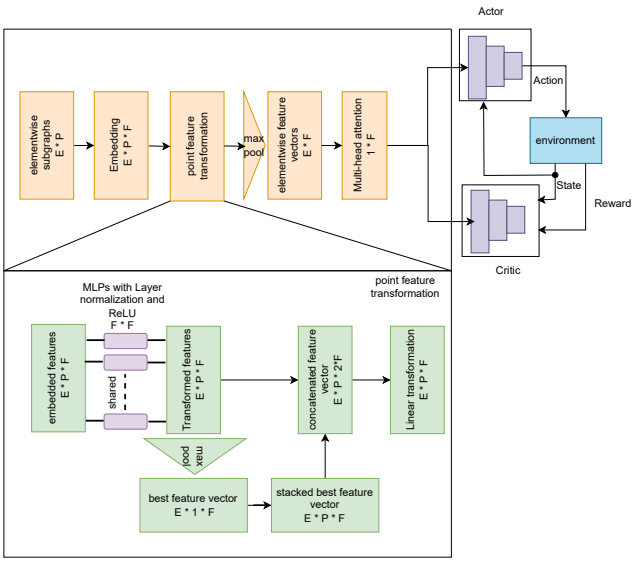

Fig. 3. Actor-critic DRL model with feature extraction steps. Each block of feature extraction is self-explanatory, provided with specific dimensions. E stands for the number of elements, P stands for the number of points used to describe an element, F stands for the size of feature vector. The output of Multi-head attention block is the *target agent's* feature vector, provided to the RL network.

- **Feature extraction** Feature extraction involves multiple steps as shown in the left side of Fig. 3. The goal is to discover an effective representation of the *target agent's* characteristics and those of its surrounding environment. It takes element-wise *sub-graphs* as input for all the elements present in the scene, and finally extracts a feature vector for the *target agent* by applying multi-head attention on the *global-graph*. Since each element is represented as a set of points, they are treated as point-features and a transformation is applied to them following the method presented in [21].

B. Action

Since our goal is to design the system as naturalistic as possible, we choose *acceleration* and *yaw* as the actions. While driving, a human driver controls throttle, brake, and steering. The throttle and brake controls *acceleration* and *deceleration* respectively, while steering controls *yaw* changes. Both *acceleration* and *yaw* action variables are continuous values ranging from -1 to 1. Negative *acceleration* signifies *deceleration*. On the other hand, a negative *yaw* value indicates left steering while a positive value represents right steering. The predicted action values within the range [-1,1] requires scaling to actual values. We determined the maximum and minimum values for *acceleration* (and *deceleration*) to be approximately ± 5.54 m/s^2 and for *yaw* to be approximately $\pm$ 45°, from the dataset. Further, the scaled value was used in the unicycle kinematic model that gave us the state value upon the predicted action.

C. Reward

The reward function should guide the *target agent* to drive in a way a regular vehicle drives. The driving patterns of regular vehicles are clearly street dependent. Vehicles will drive differently on different streets. Two key factors that decide the patterns are the *target speed*, S_T and the *safe distance*, d_s between the *target agent* and the leading vehicle. The *safe distance* is a function of the vehicle's driving speed, v_t, i.e., $ds = f(v_t)$, where s is the vehicle's current speed. This function has been established by various regulation rules. The *target speed* varies depending on the situation. It can be equal to the posted speed limit if there are no obstacles or red traffic lights, or less than that otherwise. In the event of an obstacle or a red traffic light, the instantaneous target speed for a frame will depend on the current distance to the obstacle or the stop line. This distance is known as the stopping distance. In such a case, the target speed is determined using kinematic laws and stopping distance.

It can be anticipated that most vehicles drive around their *target speed*. Therefore, the reward function should encourage learning algorithms to vary around the *target speed*. However, vehicles are also required to maintain a *safe distance*. The reward function should punish vehicles that are getting too close. In addition to that, since we are allowing the *target agent* to maintain a *safe distance* and the *target speed*, it tends to change lanes when necessary. Therefore, we must restrict the *target agent* from going outside the road or to a wrong lane. Moreover, we need to restrict the *target agent* from colliding with other agents. To this end, we propose the following reward function:

$$R = \min\left(\frac{d}{d_s}, 1\right) \exp\left(-\frac{(s - S_T)^2}{\beta}\right) + L_r + Y_r \qquad (6)$$

where d is the distance to the leading vehicle, L_r is *lane reward* and Y_r is the *yaw reward*. If $d \geq d_s$, a maximum reward of 1 is allocated for the distance maintained, indicating a good selection of action that resulted keeping a safe distance. In contrast, a value less than 1 when $d < d_s$ signals that the *target agent* failed to keep a safe distance. The *lane*

reward is set to a high negative reward to discourage going off-road, while *yaw reward* is the difference between the predicted yaw and ground truth yaw value.

V. Experiment

In this section we are going to evaluate the proposed method, specifically how it works in terms of keeping safe distance from the leading vehicle, abstaining from collisions, and going off-road.

A. Simulation Setup

We use Woven planet prediction dataset [17] to train the driving policy. It contains real-world driving scenes from urban routes located in Paolo Alto, California, USA, and hosts a numerous number of real-world driving situations with different degrees of complexity. The scenes include driving in multi-lane traffic to complex intersections, with or without traffic lights, and a variety of road geometry. While driving on the selected routes, a fleet of vehicles, called ego vehicles, tracked other vehicles, cyclists, and pedestrians around, and monitored traffic light status. In our simulation we only considered the vehicles and disregarded cyclists and pedestrians. We call the vehicles agents. The length of tracking information for agents, also known as the trajectory length, varies as it was gathered by ego vehicles while in motion. Therefore for simulation, we selected agents having a minimum track record length of 30 frames, where each frame is approximately equal to 0.1 second, according to the dataset. This also means that the trajectory length of a selected agent may be more than 30 frames. To ensure consistency among agents, we selected 30 frames from the beginning of each corresponding trajectory. In this way we get a set of eligible agents to choose from in each episode.

At the beginning of each episode, we randomly select one of the eligible agents – designated as the *target agent*. Subsequently, we find all the agents within 35 meters radius keeping the *target agent* at the center. Additionally, we obtain crosswalk and lane locations from high-definition (HD) map associated with the dataset. Furthermore, we acquire the traffic signal state pertaining to each lane. These features collectively constitute observations for the *target agent*. During each interaction between the target agent and the environment in the reinforcement learning loop, a set of four consecutive observations are grouped together to form a state, as described in section IV(A). Subsequently, predictions are made for each frame starting from the fifth frame in the first iteration, and continuing sequentially for subsequent frames. Given that the trajectory length of a *target agent* is 30 frames, we obtain 26 consecutive predictions in each episode. Therefore, the episode length is equivalent to 26 frames.

The interactions work iteratively, where each iteration is called a step. In each step, the RL policy takes the current state from the environment, predicts actions, and the environment evaluates the actions by providing a reward along with the next state. The next state is prepared by taking the *target agent*'s predicted actions (acceleration/deceleration and yaw rate) into consideration. These actions are translated into features like velocity, location, and yaw using the unicycle kinematic model. The *target agent* then adopts the newly forecasted location, while the remaining agents adhere to their ground truth features.

In our work, we employed the Stable-Baselines3 [22] PPO implementation. The simulation was conducted by using the parameters in Table II.

TABLE II
Simulation parameters

Parameter	Value
Episode length	26 Frames (= 2.6 seconds)
Learning rate	$3 \exp -4$
Batch size	512
Number of epochs	10
Discount rate	0.8
Entropy coefficient	0.01

B. Evaluation Metrics

We used the following metrics to evaluate the performance of the proposed model.

- `Distance_to_leading_vehicle`. This metric represents the ratio of actual distance to the leading vehicle compared to the safe distance. The optimum result would be 1 and a value close to 1 is considered to be a good distance being upheld.
- `Off_road_percentage`. The frequency, expressed as a percentage of episodes, the *target agent* goes outside the permissible lane markings. This includes instances where the agent goes outside the road boundaries or enters a lane designated for vehicles travelling in the opposite direction.
- `Collision_percentage`. Percentage of episodes collision happened between the *target agent* and its neighboring agents. A collision is recognized if the bounding boxes of the *target agent* and a neighboring agent intersect to any extent.

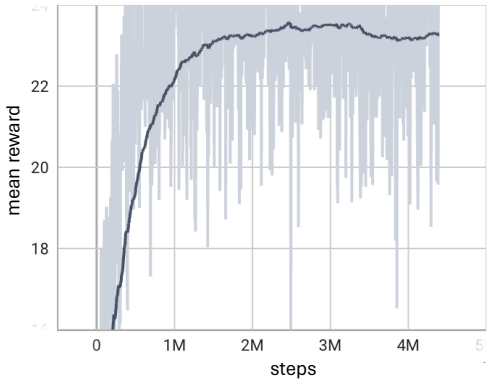

Fig. 4. Mean reward over simulation steps. The reward plateaued after approximately 2.5 million steps.

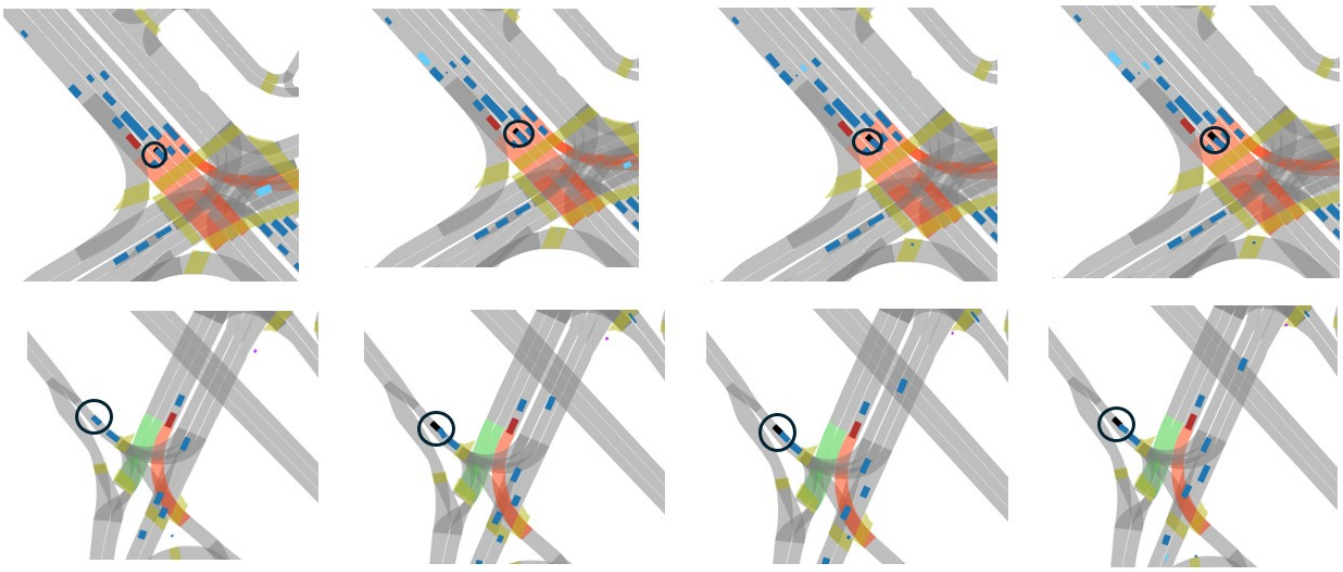

Fig. 5. Qualitative results demonstrating performance in two episodes. The upper row shows the first episode, and the lower row shows the second. Each episode includes four Frames (0, 10, 15 and 25) with the *target agent* indicated by a circle. In the first episode, the *target agent* stops at a junction and before crosswalk with a red light. In the second episode, the leading vehicle is stopped, and the *target agent* approaches and stops keeping a safe distance from the leading vehicle.

TABLE III
ILLUSTRATION OF PERFORMANCE ENHANCEMENTS ON SAVED MODELS ACROSS TIME

	1	2	3	4	5	6	7	8	9	10
Dist_ratio	.789	.861	.861	.872	.941	.963	.946	.954	.956	.961
Off_road incidents (%)	15	8	6	2	3	1	1	1	1	0
Collision (%)	22	30	27	25	17	10	21	18	18	9

C. Results

The reward mean graph shown in Fig. 4 illustrates the performance of the trained model over the course of more than 4 million training steps. Specifically, the agent's policy is updated every 15,360 training steps, and an average reward of 10 episodes is then calculated by evaluating the updated policy. The policy update interval is a hyperparameter and is determined based on the batch size and episode length in our case. For clarity, the line has been smoothed. We can see that after approximately 2.5 million steps, the reward reaches a plateau.

To measure the effectiveness of the proposed system, we provide results in Table III using the evaluation metrics mentioned in sub-section *B* across 10 saved checkpoints. These checkpoints, captured at various intervals during training, offer insights into the model's evolving performance over time. The results show excellent performance in the mean distance to the leading vehicle, which gradually converges to approximately 1. It indicates that the model effectively learned to maintain a safe distance to the leading vehicle. The off-road percentage steadily diminishes to 0, signifying improved adherence to safe driving policy. While the collision rate initially exhibits a high frequency, it gradually reduces, though not to the extent to be considered as excellent performance. The collision rate might be higher due to the fact that we primarily focused on maintaining a safe distance from the leading vehicle, and ignored the need to maintain safe distances from potential vehicles in adjacent lanes or rapidly approaching vehicles from behind.

Qualitative results are presented in Fig. 5 for two scenarios where the agent successfully adheres to traffic rules. The first scenario depicts the agent appropriately stopping at an active red light, halting before the crosswalk. In the second scenario, the agent maintains a safe distance from a leading vehicle.

VI. CONCLUSION

In this study, we attempted a challenging task of developing a safe driving policy for urban areas by training an actor-critic deep reinforcement learning model with real-world data. This research stands out as one of the few works in literature that tried learning a policy by using deep reinforcement learning model on actual dataset, instead of using a simulated one. Both qualitative and quantitative results demonstrate the model's effectiveness in maintaining a safe distance from the leading vehicle and avoiding off-road incidents. While our focus was solely on maintaining a safe distance to the leading vehicle, the collision rate was not much satisfactory and hence enhancements are necessary, particularly by considering distances to the following vehicles and vehicles in the adjacent lanes.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Professor Halim Yanikomeroglu for generously dedicating his time and

expertise to review and provide valuable feedback on this research paper.

REFERENCES

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[2] D. Coelho and M. Oliveira, "A review of end-to-end autonomous driving in urban environments," *IEEE Access*, vol. 10, pp. 75 296–75 311, 2022.

[3] J. Chen, S. E. Li, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 5068–5078, 2021.

[4] P. Cai, S. Wang, Y. Sun, and M. Liu, "Probabilistic end-to-end vehicle navigation in complex dynamic environments with multimodal sensor fusion," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4218–4224, 2020.

[5] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *IEEE Int. Conf. on Rob. and Auto.(ICRA)*, 2018, pp. 4693–4700.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *Int. Conf. on Rob. and Auto. (ICRA)*, 2019, pp. 8248–8254.

[8] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "End-to-end deep reinforcement learning for lane keeping assist," *arXiv preprint arXiv:1612.04340*, 2016.

[9] F. Ye, X. Cheng, P. Wang, C.-Y. Chan, and J. Zhang, "Automated lane change strategy using proximal policy optimization-based deep reinforcement learning," in *IEEE Intell. Veh. Symp.(IV)*, 2020, pp. 1746–1752.

[10] J. Wang, Q. Zhang, D. Zhao, and Y. Chen, "Lane change decision-making through deep reinforcement learning with rule-based constraints," in *Int. Joint Conf. on Neu. Net. (IJCNN)*, 2019, pp. 1–6.

[11] Y. Wang, J. Wang, Y. Yang, Z. Li, and X. Zhao, "An end-to-end deep reinforcement learning model based on proximal policy optimization algorithm for autonomous driving of off-road vehicle," in *International Conference on Autonomous Unmanned Systems*. Springer, 2022, pp. 2692–2704.

[12] J. Li, X. Wu, J. Fan, Y. Liu, and M. Xu, "Overcoming driving challenges in complex urban traffic: A multi-objective eco-driving strategy via safety model based reinforcement learning," *Energy*, vol. 284, p. 128517, 2023.

[13] P. Maramotti, A. P. Capasso, G. Bacchiani, and A. Broggi, "Tackling real-world autonomous driving using deep reinforcement learning," in *IEEE Intell. Veh. Symp.(IV)*, 2022, pp. 1274–1281.

[14] H. Rathore and V. Bhadauria, "Intelligent decision making in autonomous vehicles using cognition aided reinforcement learning," in *IEEE Wireless. Comm. and Net. Conf.(WCNC)*. IEEE, 2022, pp. 524–529.

[15] P. Cai, H. Wang, Y. Sun, and M. Liu, "DQ-GAT: Towards safe and efficient autonomous driving with deep Q-learning and graph attention networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 21 102–21 112, 2022.

[16] T. Agarwal, H. Arora, and J. Schneider, "Learning urban driving policies using deep reinforcement learning," in *IEEE Int. Intell. Trans. Sys. Conf.(ITSC)*. IEEE, 2021, pp. 607–614.

[17] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," in *Conference on Robot Learning*. PMLR, 2021, pp. 409–418.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska, "Urban driver: Learning to drive from real-world demonstrations using policy gradients," in *Conference on Robot Learning*. PMLR, 2022, pp. 718–728.

[20] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "Vectornet: Encoding HD maps and agent dynamics from vectorized representation," in *IEEE/CVF Conf. on Comp. Vis. and Patt. Recog.*, 2020, pp. 11 525–11 533.

[21] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," in *IEEE Conf. on Comp. Vis. and Patt. Recog.*, 2017, pp. 652–660.

[22] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html