DriverGym: Democratising Reinforcement Learning for Autonomous Driving

Parth Kothari
Level 5, Woven Planet

Christian Perone
Level 5, Woven Planet

Luca Bergamini
Level 5, Woven Planet

Alexandre Alahi
EPFL

Peter Ondruska
Level 5, Woven Planet

Abstract

Despite promising progress in reinforcement learning (RL), developing algorithms for autonomous driving (AD) remains challenging: one of the critical issues being the absence of an open-source platform capable of training and effectively validating the RL policies on real-world data. We propose DriverGym, an open-source OpenAI Gym-compatible environment specifically tailored for developing RL algorithms for autonomous driving. DriverGym provides access to more than 1000 hours of expert logged data and also supports reactive and data-driven agent behavior. The performance of an RL policy can be easily validated on real-world data using our extensive and flexible closed-loop evaluation protocol. In this work, we also provide behavior cloning baselines using supervised learning and RL, trained in DriverGym. Code and videos are available on the L5Kit repository.

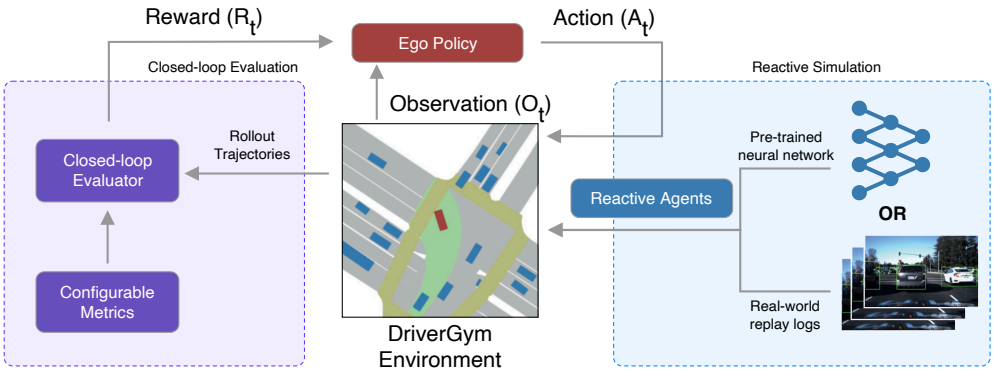

Figure 1: DriverGym: an open-source gym environment that enables training RL driving policies on real-world data. The RL policy can access rich semantic maps to control the ego (red). Other agents (blue) can either be simulated from the data logs or controlled using a dedicated policy pre-trained on real-world data. We provide an extensible evaluation system (purple) with easily configurable metrics to evaluate the idiosyncrasies of the trained policies.

1 Introduction

Recently, Reinforcement Learning (RL) has achieved great success in a variety of applications like playing Atari games [Mnih et al., 2013], board games [Silver et al., 2017], manipulating sensorimotor

Machine Learning for Autonomous Driving Workshop at the 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

Table 1: Comparison of various open-source RL simulation environments for autonomous driving.

Name	Gym Compatible	Evaluation Protocol	Simulator Expert Data	Real-world Expert Data	Agents Model
TORCS	✗	✗	✗	✗	Rule-based
Highway-Env	✓	✗	✗	✗	Rule-based
CARLA (Official)	✗	✓	14 hrs	✗	Rule-based
SMARTS	✓	✓	✗	✗	Rule-based *or* Data-driven
CRTS	✓	✓	✗	64 hrs	Real-world Logs
DriverGym	✓	✓	✗	1000 hrs	Data-driven *or* Real-world Logs

in three-dimensions [Martín-Martín et al., 2019]. However, developing RL algorithms for real-world applications such as autonomous driving (AD) remains an open challenge [Kiran et al., 2020]: with AD being an extremely safety-critical task, one cannot directly deploy a policy in the real world for data collection or policy validation.

One solution is to deploy the policy in the real world with a safety driver inside the car at all times. However, this process is time-consuming, and more importantly, not accessible to all of the research community. Therefore, to tackle this challenge, there is a dire need for an RL simulation environment that can (1) be used to easily train RL policies using real-world logs, (2) simulate surrounding agent behavior that is both realistic and reactive to the ego policy, (3) effectively evaluate the trained models, (4) be flexible in its design, and (5) inclusive to the entire research community.

We propose DriverGym, an open-source gym-compatible environment specifically tailored for developing and experimenting with RL algorithms for self-driving (see Fig. 1). DriverGym utilizes one of the largest public self-driving datasets, *Level 5 Prediction Dataset* [Houston et al., 2020] containing over 1,000 hours of data, and provides support for reactive agent behavior simulation [Bergamini et al., 2021] using data-driven models. Furthermore, DriverGym provides an extensive and extensible closed-loop evaluation system: it not only comprises a variety of AD-specific metrics but also can be easily extended to incorporate new metrics to evaluate idiosyncrasies of trained policies. We open-source the code and pre-trained models to stimulate development.

In this work, we provide the following contributions:

- An open-source and OpenAI gym-compatible environment for autonomous driving task;
- Support for more than 1000 hours of real-world expert data;
- Support for logged agents replay or data-driven realistic agent trajectory simulations;
- Configurable and extensible evaluation protocol;
- Provide pre-trained models and the corresponding reproducible training code.

2 Related Work

To replicate the success of the OpenAI Gym framework [Brockman et al., 2016], many simulation environments have been developed in the context of autonomous driving [Espié et al., 2005, Leurent, 2018, Dosovitskiy et al., 2017, Lopez et al., 2018, Quiter and Ernst, 2018]. Table 1 provides a comparison amongst commonly used RL simulation environments including DriverGym. Racing simulators like TORCS [Espié et al., 2005] offer limited scenarios of driving. Highway-Env [Leurent, 2018] provides a collection of gym-compatible environments for autonomous driving. However, it lacks important semantic elements like traffic lights, an extensive evaluation protocol and expert data.

Traffic simulators like CARLA [Dosovitskiy et al., 2017], SUMO [Lopez et al., 2018] supports flexible specification of traffic conditions for training and testing. However, they are synthetic simulators that utilize hand-coded rules for surrounding agents' motion that tends to be unrealistic and display a limited variety of behaviors. Crucially, they lack access to real-world data logs. SMARTS [Zhou et al., 2020] overcomes the former issue by providing *Social Agent Zoo* that supports data-driven agent models while CRTS [Osinski et al., 2020] tackles the latter providing access to 64 hours of real-world logs within the CARLA simulator. DriverGym solves both these challenges: it enables simulating reactive agents using data-driven models learned from real-world data, and provides access to 1000 hours of real-world logs to initialize episodes or simulate agents.

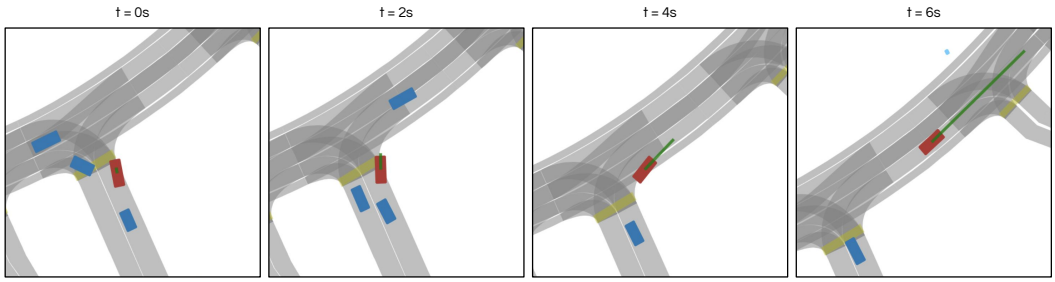

Figure 2: Visualization of an episode rollout (ego in red, agents in blue) in DriverGym. The policy prediction (green line) is scaled by factor of 10 and shown at 2 second intervals for better viewing.

3 DriverGym

DriverGym aims to foster the development of RL policies for self-driving by providing a flexible interface to train and evaluate RL policies. Our environment is compatible with both SB3 [Raffin et al., 2019] and RLlib [Liang et al., 2018], two popular frameworks for training RL policies. Our code is open-source with Apache 2 license. We describe the components of our environment below.

3.1 State Representation

The state representation captures the context around the ego agent, in particular, the surrounding agents' positions, their velocities, the lanes and traffic lights. We encode this information in the form of a 3D tensor that is the birds-eye-view (BEV) raster image of the current frame. DriverGym supports all the rasterization modes provided by L5Kit [Houston et al., 2020] (see Fig 3 in Appendix). Compared to Atari environments, DriverGym requires more time to generate observations as the latter has to load real-world data and subsequently render high-resolution raster images.

3.2 Action Spaces

The action produced by the RL policy is used to control the motion of the ego agent. The action is propagated as (x, y, yaw) to update the state of the ego. Still, DriverGym does not make any strict assumptions on the policy itself which can, for instance, output $(acceleration, steer)$ and use a kinematic model to decode the next-step observation.

3.3 Reactive Agents

An important component of the DriverGym environment is to model the motion of the surrounding agents. DriverGym allows flexibility in this aspect and currently support two ways of controlling the behavior of surrounding agents: *log replay* and *reactive simulation*.

In *log replay*, during an episode rollout, the movement of surrounding agents around the ego is replayed in the exact same manner as it happened when the log was collected in the real world. In *reactive simulation*, the agent behavior is both reactive and realistic. Motivated by [Bergamini et al., 2021], DriverGym allows simulating agent reactivity using data-driven models that learn agent behavior from real-world data, *i.e.,* users can provide neural-network-based agent models trained on real-world data, to simulate agent behavior.

3.4 Rewards

The rewards in the environment quantify the performance of a driving policy during a rollout and subsequently guide the training of the policy using reinforcement learning. DriverGym, through the Closed-Loop Evaluation (CLE), supports a variety of AD-specific metrics that are computed per-frame, and can be combined to construct the reward function. This system is described in the section below.

Table 2: Evaluation of different training strategies using the CLE protocol in DriverGym. Lower is better. **SL**: Supervised learning using L2 imitation loss, **SL + P**: SL plus trajectory pertubations, **PPO**: RL using PPO, with imitation-loss based reward. More information about metrics and validators in Appendix A.1. Metrics and validators are in the format: average (std. deviation).

Method	**Metrics**		**Validators**				
	Average Displacement	Final Displacement	Final Displacement (>= 30.0m)	Distance To Reference (>= 4.0m)	Front Collision	Side Collision	Rear Collision
SL	32.4 ± 2.7	74.5 ± 5.5	9.5 ± 4.9	26 ± 0.0	12.5 ± 3.5	19 ± 5.6	16 ± 1.4
SL + P	**13.4 ± 1.4**	**25.5 ± 3.5**	5.3 ± 0.6	**9.7 ± 1.5**	9 ± 2.6	12.3 ± 6.8	**7 ± 0.0**
PPO	18.7 ± 2.3	46.4 ± 7.7	**4.0 ± 2.0**	12.7 ± 3.2	**4.3 ± 2.5**	**6 ± 3.5**	27 ± 5.0

3.5 Closed-Loop Evaluation Protocol

Having an extensive closed-loop evaluation (CLE) protocol is a necessity to correctly assess the performance of RL policies before deployment in safety-critical real-world scenarios. Our CLE framework comprises insightful AD-specific metrics: the first set of metrics, specific to imitation learning, are distance-based metrics. The second set of metrics, specific to safety, capture the various types of collisions that occur between ego and surrounding agents. These include front collision, side collision and rear collisions. More importantly, our CLE framework can be easily extended to incorporate new metrics that can help to test various properties of the trained policy. An in-depth description of our CLE and its flexibility is provided in the appendix section.

4 Experiments

We evaluate three different algorithms using DriverGym to compare the effectiveness of these training strategies. The first one is an open-loop training baseline using L2 imitation loss (**SL**). Naive behavioral cloning is known to suffer from distribution shift between training and test data [Ross et al., 2011]. We compare it with a stronger baseline, inspired by ChaufferNet [Bansal et al., 2019], that alleviates distribution shift by introducing synthetic perturbations to the training trajectories (**SL + P**).

We also evaluate an RL policy, namely Proximal Policy Optimization (PPO) [Schulman et al., 2017] implemented in the SB3 framework [Raffin et al., 2019]. We choose PPO as it not only demonstrates remarkable performance but it is also empirically easy to tune [Schulman et al., 2017]. All the experiments have been performed on 2 Tesla T4 GPUs. The details of the model architectures, training strategies, hyperparameters used and experimental setup are provided in the appendix.

The performance of three runs (different seeds) of the three models on 100 real-world test scenes is reported in Table 2. Based on distance-based metrics, **PPO** is similar to **SL + P** in terms of ADE, however it suffers from high FDE. **PPO** showed fewer front and side collisions, however, it showed a much higher number of rear collisions, which can be explained by the passiveness of the ego vehicle. Finally, **SL** is the worst and corroborates the expectations.

5 Discussion

We believe DriverGym is an important step towards solving the task of planning for autonomous driving. Thanks to its gym-compatible interface, it allows to easily train and evaluate RL policies for self-driving. Furthermore, surrounding agents can be controlled via a model trained on real-world data to improve their reactivity towards the ego. A current weakness of DriverGym is the time complexity of policy rollouts, which can be reduced through faster observation generation and mitigation of inter-process communication.

One avenue for future work is to provide fine-grained policy evaluation for different scene categories (e.g. restarting from an intersection controlled by a traffic light). We hope that DriverGym will provide a common ground for policy evaluation that is extensible, and will drive the improvement of the next generation of planning algorithms.

6 Acknowledgements

We would like to thank everyone at Level 5 working on data-driven planning, in particular Sergey Zagoruyko, Alborz Alavian, Oliver Scheel, Yawei Ye, Moritz Niendorf, Stefano Pini and Maciej Wołczyk.

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

D. Silver, T. Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, A. Guez, Marc Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

Roberto Martín-Martín, Michelle A. Lee, Rachel Gardner, S. Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1010–1017, 2019.

B. R. Kiran, Ibrahim Sobh, V. Talpaert, P. Mannion, A. A. Sallab, S. Yogamani, and P. P'erez. Deep reinforcement learning for autonomous driving: A survey. *ArXiv*, abs/2002.00444, 2020.

J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. https://level-5.global/level5/data/, 2020.

Luca Bergamini, Y. Ye, Oliver Scheel, Long Chen, Chih Hu, Luca Del Pero, Blazej Osinski, Hugo Grimmett, and Peter Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations. *ArXiv*, abs/2105.12332, 2021.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, J. Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *ArXiv*, abs/1606.01540, 2016.

E. Espié, Christophe Guionneau, Bernhard Wymann, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. 2005.

Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

A. Dosovitskiy, G. Ros, Felipe Codevilla, Antonio M. López, and V. Koltun. Carla: An open urban driving simulator. *ArXiv*, abs/1711.03938, 2017.

Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL https://elib.dlr.de/124092/.

Craig Quiter and Maik Ernst. deepdrive/deepdrive: 2.0 (2.0). https://doi.org/10.5281/zenodo.1248998, 2018.

Ming Zhou, Jun Luo, Julian Villela, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, Aurora Chongxi Huang, Ying Wen, Kimia Hassanzadeh, Daniel Graves, Dong Chen, Zhengbang Zhu, Nhat M. Nguyen, Mohamed Elsayed, Kun Shao, Sanjeevan Ahilan, Baokuan Zhang, Jiannan Wu, Zhengang Fu, Kasra Rezaee, Peyman Yadmellat, Mohsen Rohani, Nicolas Perez Nieves, Yihan Ni, Seyedershad Banijamali, Alexander Cowen Rivers, Zheng Tian, Daniel Palenicek, Haitham Ammar, Hongbo Zhang, Wulong Liu, Jianye Hao, and Jintao Wang. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *ArXiv*, abs/2010.09776, 2020.

Blazej Osinski, Piotr Milos, Adam Jakubowski, Pawel Ziecina, Michal Martyniak, Christopher Galias, Antonia Breuer, Silviu Homoceanu, and Henryk Michalewski. Carla real traffic scenarios - novel training ground and benchmark for autonomous driving. *ArXiv*, abs/2012.11329, 2020.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *ICML*, 2018.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *ArXiv*, abs/1812.03079, 2019.

J. Schulman, F. Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Bokeh Development Team. *Bokeh: Python library for interactive visualization*, 2018. URL https://bokeh.pydata.org/en/latest/.

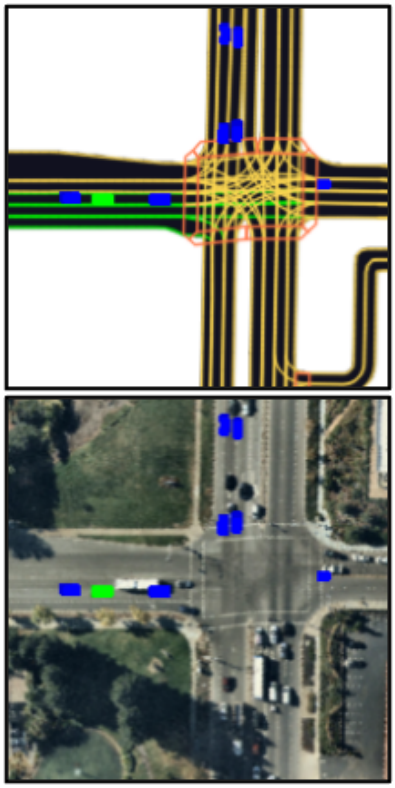

Figure 3: Example Rasterization Modes

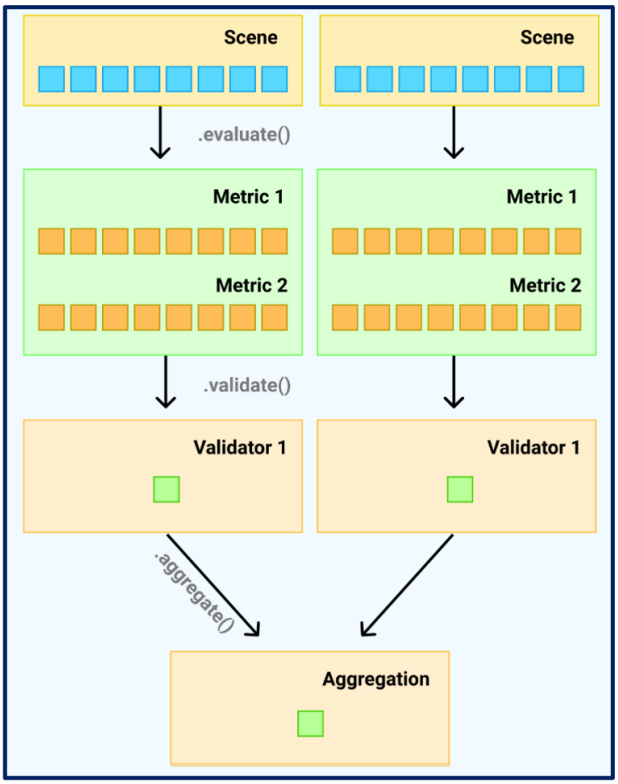

Figure 4: Evaluation Plan

A Appendix

A.1 Closed-Loop Evaluation Protocol

Our CLE works on top of the simulation outputs provided by episode rollouts in DriverGym. An *evaluation plan* (Fig. 4) is defined that comprises (1) metrics that are computed per frame (e.g. L2 displacement error) (2) validators that enforce constraints on the metrics per scene (L2 displacement error $\leq$ 4 meters), and (3) composite metrics per scene, that can depend both on the output of metrics

Table 3: Description of the various metrics provided in closed-loop evaluation protocol.

Name	Type	Description
Average Displacement	Distance-Based	Computes the L2 distance between predicted ego centroid and ground-truth ego centroid averaged over the entire episode
Final Displacement	Distance-Based	Computes the L2 distance between predicted ego centroid and ground-truth ego centroid at the last timestep of the episode
Distance to Reference	Distance-Based	Computes the L2 distance between the predicted centroid and the closest waypoint in the reference trajectory (ground-truth ego)
Front Collision	Safety-Based	Computes whether a collision occurred between the front of ego and any another agent
Side Collision	Safety-Based	Computes whether a collision occurred between a side of ego and any another agent
Rear Collision	Safety-Based	Computes whether a collision occurred between the rear of ego and any another agent

and validators (e.g. passed driven miles). An example plan is provided in Listing 2. Note that our CLE supports both reactive and non-reactive agents.

The evaluation protocol is flexible and new metrics can be easily incorporated to target specific cases of model failures. Within CLE, the user has access to all the simulation artifacts (trajectories, maps, *log replay* data of ego and agents) while designing a new metric. We hope the DriverGym evaluation protocol facilitates researchers to diagnose targeted behaviors of their policies.

A.2 Model Architecture

In our experiments, the backbone feature extractor is shared between the policy and the value networks. The feature extractor is composed of two convolutional networks followed by a fully connected layer, with ReLU activation. The feature extractor output is passed to both the policy and value networks composed of two fully connected layers with tanh activation. The open-loop baseline models have the same backbone architecture as above.

We perform group normalization after every convolutional layer. Empirically, we found that group normalization performs far superior to batch normalization. This can be attributed to the fact that activation statistics change quickly in on-policy algorithms (PPO is on-policy) while batch-norm learnable parameters can be slow to update causing training issues.

A.3 Training

The training data comprises 100 scenes (average frame length ~ 248) where the initial frame is randomly sampled. The open-loop baseline models are trained in a supervised manner where the L2 loss is calculated on the predictions of 12 future time-steps (1.2 secs).

We train the PPO policy in closed-loop (the surrounding agents are *log replayed*) for episodes of length 32 time-steps. PPO policy network predicts the mean and standard deviation values of a gaussian to represent its actions. Further, the policy is initialized such that the initial actions are independent of the observations. Therefore, we normalize the action space (zero mean) for faster training convergence. For further training stability, we incorporate a unicycle kinematic model at the policy output, *i.e.* the policy predicts the acceleration and steer.

For the PPO policy, we use an imitation loss-based reward. We define the reward as the negative of the L2 distance between the policy prediction and ego replay at every time-step. Reward clipping is performed for stability. Note that, DriverGym can also incorporate non-differentiable hand-crafted rules like collisions in the reward function to train different RL policies.

A.4 Hyperparameters

We train the PPO policy for $12M$ steps in which the learning rate is fixed to $3\text{e}{-4}$ for the first $8M$ steps and then decreased by a factor of 10 for the rest of the training. The discount factor is 0.80 and GAE is 0.90. 4 environments are rolled out in parallel for a total of 1024 time-steps before the model is updated for 10 epochs on the collected rollout buffer. The mini-batch size of the model update is 64 and the clipping parameter ϵ follows a linear decay schedule during training starting from 0.1. The reward clipping threshold is fixed to 15.

We train on 112×112 pixel BEV rasters centered around the ego. The raster image is generated by combining the semantic map (3 channels) and the bounding boxes of the various agents in the scene (top image in Fig 3). The past history and current bounding boxes of the agents (including ego) are incorporated via the channel dimension. We consider a history of 3 frames which along with the current frame leads to an additional 8 channels resulting in a raster image of size $112 \times 112 \times 11$. The raster image is transformed such that it is centered around the ego vehicle.

A.5 Visualizations

The DriverGym environment provides the user with the ability to visualize the output of episode rollouts (see example in Fig 2). The visualization is carried out using the Bokeh interaction visualization library [Bokeh Development Team, 2018].

A.6 DriverGym API Snippets

```python
from stable_baselines3 import PPO
import gym

env = gym.make("drivergym-v0")
model = PPO("CnnPolicy", env)
model.learn(n_steps=1000000)
```

Listing 1: Code snippet showing the user API for using DriverGym environment with Stable Baselines3 [Raffin et al., 2019].

```python
from l5kit.cle.metrics import SupportsMetricCompute

class L2DisplacementErrorMetric(SupportsMetricCompute):
    metric_name = "l2_displacement_error"

    def compute(self, simulation_output: SimulationOutputCLE) -> torch.Tensor:
        simulated_scene_ego_state = simulation_output.simulated_ego_states
        simulated_centroid = simulated_scene_ego_state[:, :2]
        observed_ego_states = simulation_output.recorded_ego_states[:, :2]
        return torch.norm(simulated_centroid - observed_ego_states_fraction)
```

Listing 2: Code snippet showing the flexibility of adding new metrics to closed-loop evaluator (CLE). In this example, we are defining a L2 displacement error metric.