Appendix

In this appendix, we provide more details about the four experiments and some scenario examples from the three databases used in the experiments. Section A is on the scenario generation model; Section B is on the single-agent cross-dataset generalization experiment; Section C is on the multi-agent reinforcement learning and imitation learning; Section D shows the interface of ROS bridge and the qualitative results of running Openpilot test. Section E shows more rendered scenario examples from each database.

Codebase, documentation, videos, and a scenario gallery are available at https://metadriverse.github.io/scenarionet.

A Scenario Generation Model

We use all the databases (nuPlan, Waymo, PG) in scenarioNet for cross-dataset traffic scenario generation experiments. The goal is to show that scenarioNet contains diverse traffic scenario data for training deep neural networks. We conduct our experiments based on TrafficGen [18], a neural generative model previously developed for reactive traffic scenario generation.

A.1 Task Definition

A traffic scenario is denoted as $\tau = (\mathbf{m}, \mathbf{s}_{1:T})$, which lasts T time steps and contains the High-Definition (HD) road map $\mathbf{m}$ and the state series of traffic vehicles $\mathbf{s}_{1:T} = [s_1, ..., s_T]$. Each element $s_t = \{s_t^1, ...s_t^N\}$ is a set of states of N traffic vehicles at time step t. Given an existing scenario $\tau = (\mathbf{m}, \mathbf{s}_{1:T})$, the goal of TrafficGen is to learn to generate **new** traffic scenarios $\tau' = (\mathbf{m}, \mathbf{s'}_{1:T'})$ that have similar distribution with τ and different states $\mathbf{s}'$ and longer time steps T'. After training, TrafficGen takes $\mathbf{m}$ as input and generates τ' as a totally different scenario.

A.2 Model Architecture

Traffic Scenario Encoder. TrafficGen uses vectorization to encode map and vehicle information, representing lanes as sets of vectors, each vector representing a small region. A vector-based coordinate system is established for each small region, with each vector comprising a start point p_i^s, endpoint p_i^e, and information about vehicles in this region. Thus, a traffic snapshot τ at time step t is represented as $\tau_t = \mathbf{v} = v_i{}_{i=1}^{I}$. Cross attention mechanism is applied to the unordered set $\mathbf{v}$ to fuse information from different regions into a single context vector.

Decoder for Scenario Generation. TrafficGen places vehicles by generating a set of weights for all regions, which is then turned into a categorical distribution. The local position of a tentative vehicle is modeled by a mixture of K bivariate normal distributions, as are heading, speed, and size of the vehicle. Autoregressive sampling is used to create a traffic snapshot. A motion forecasting model is used as the trajectory generator.

A.3 Experiment Setting

We train a scenario generation model TrafficGen with mixed data. Specifically, we randomly sample 30% data from nuPlan, Waymo, and PG. We filter out the scenarios with less than 8 cars and crop a rectangular area with a side length of 120m centered on the ego vehicle. Each of the 20s scenarios is split into 10 traffic snapshots with 2s intervals. The training is executed on servers with 8 x Nvidia 2080TI and 256 G memory in Distributed Data-Parallel (ddp) mode. We set the feature size to be 1024, and use 3-layer MLPs with hidden dimensions of [1024, 512, 256] for attribute modeling. The training takes 16 hours for 100 training epochs and the learning rate decays by 20% at every 30 epochs. The detailed hyperparameters are shown in Table 4.

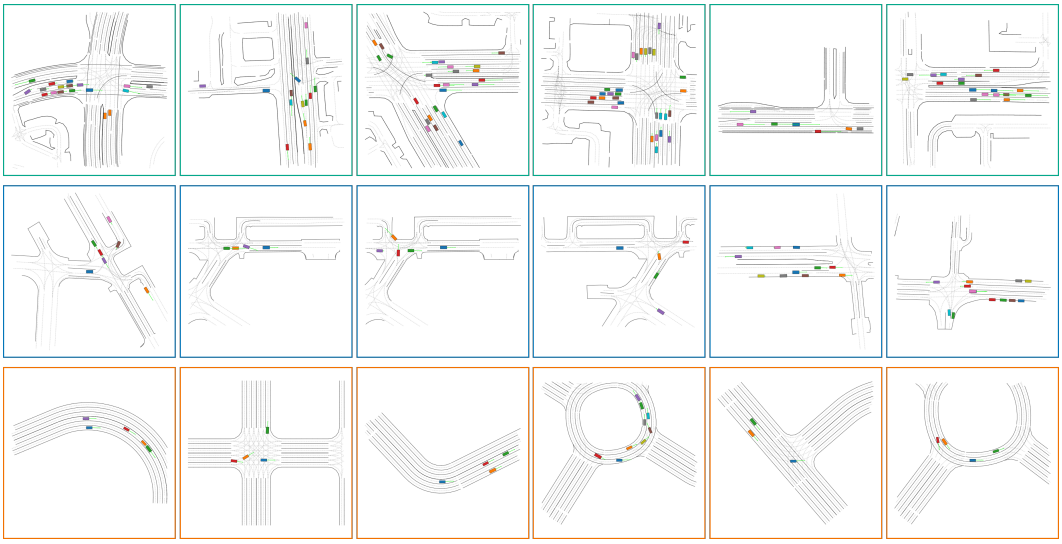

Figure 6: Traffic scenarios generated from the model trained on different databases: Waymo (□), nuPlan (□), and PG (□).

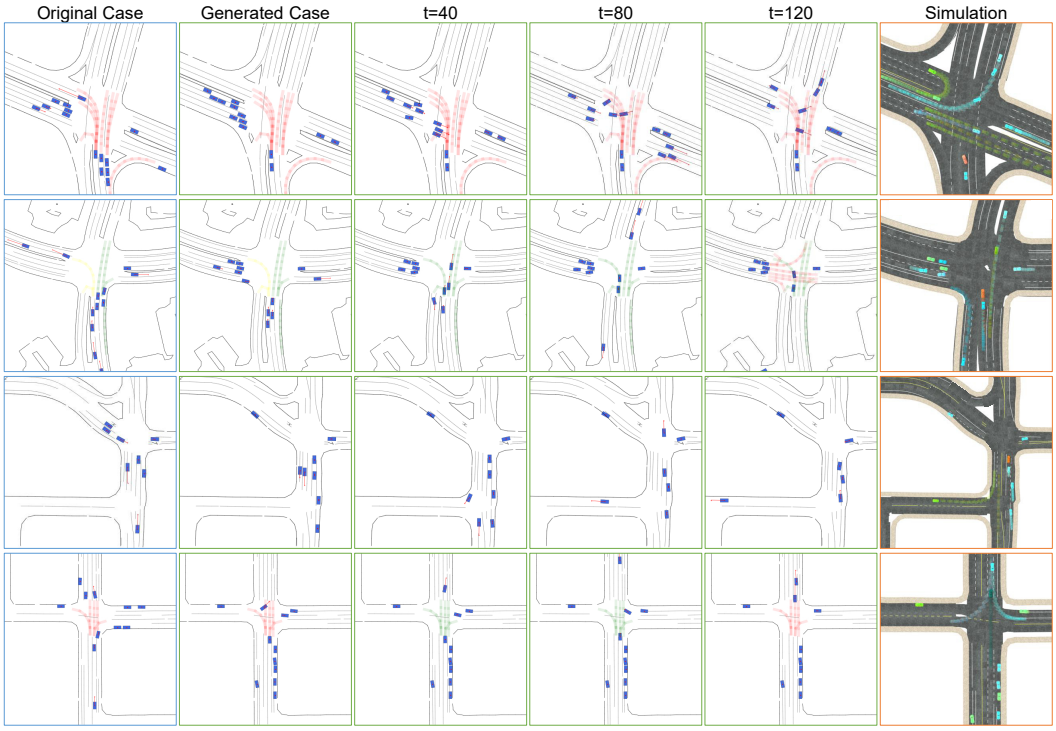

Figure 7: Dynamics of the generated traffic scenarios. The first column is the original case. The middle columns show the generated scenarios at different timesteps. The last column shows the corresponding scenarios imported in the simulation. The green and red dashed lines indicate the traffic light status of this lane at the intersection.

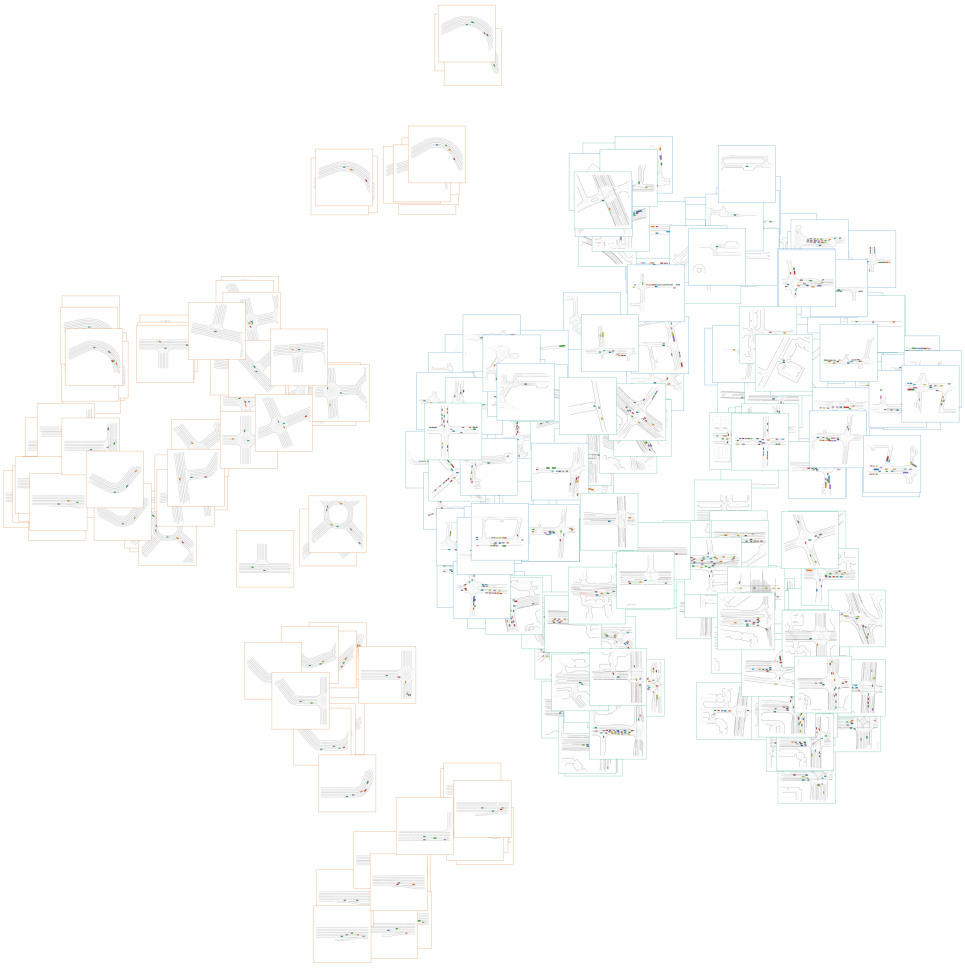

Figure 8: t-SNE visualizations of 3000 scenarios. PG scenarios (□) are located mostly on the left side, while the scenarios from nuPlan (□) and the Waymo scenarios (□) are scattered on the right side.

We plot some generated traffic scenarios in Fig. 6 and the dynamics in Fig. 7. It shows that the models trained on the processed scenario data from ScenarioNet can generate realistic and diverse traffic behaviors and interactions.

A.4 t-SNE Visualizations

The trained model's encoder is used to extract feature embeddings of a given scenario sample. The embeddings are then visualized with t-SNE method to show the similarities and differences. The detailed hyperparameters of t-SNE are shown in Table 5.

The t-SNE result is plotted in Fig. 8. The clustering results show that there is a large domain gap between real-world scenarios (Waymo and nuPlan) and synthetic scenarios (PG). Besides, domain gap exists even in real-world datasets. One prominent feature of the Waymo dataset is the complex crossroad (shown in the lower right corner), which the nuPlan dataset is lacking. The t-SNE visualization reveals the differences between different traffic datasets.

Table 4: TrafficGen

Hyper-parameter	Value
Batch Size	256
Feature Size	1024
Training epochs	100
Learning Rate	3e−4
Activation Function	"relu"
MCG Layers	5

Table 5: t-SNE

Hyper-parameter	Value
Components Number	2
Init	pca
Learning Rate	auto
Perplexity	30
Early Exaggeration	12

B Single-agent Cross-dataset Generalization Experiment

We use the PG database and nuPlan database for cross-dataset generalization experiments. The goal is to investigate how the sim-to-real gap affects the generalizability of the learning-based vehicle controller. To this end, we train agents on synthetic PG scenarios [28] and nuPlan [6] scenarios respectively, and test them on the same held-out real-world test set.

B.1 Task Setup

To be specific, the task is to follow the trajectory of the data collection car and drive as fast as possible while avoiding collisions.

Observation. The observation of the RL agents is as follows:

- A 120-dimensional vector denoting the Lidar-like point clouds with $50m$ maximum detecting distance centering at the target vehicle. Each entry is in $[0, 1]$ with Gaussian noise and represents the relative distance of the nearest obstacle in the specified direction.
- A vector containing the data that summarizes the target vehicle's state such as the steering, heading, velocity, and relative distance to the trajectory to follow.
- The navigation information that guides the target vehicle toward the destination. Concretely, it consists of 10 points sampled on the future trajectory and the distance between two consecutive points is $2m$. The points will be projected to the vehicle coordinates.
- A 12-dimensional vector denoting the Lidar-like point clouds with $50m$ maximum detecting the boundary of the drivable area, like the solid lines or sidewalks. (Optional)

As the vehicle for collecting nuPlan data in Boston sometimes drives out of the drivable area for bypassing the cones or barriers, crossing the drivable area boundary usually happens. Therefore, we didn't use the boundary detector in our experiments, and crossing the boundaries like solid lines won't terminate the episode nor penalize the agent.

Action. The driving policy is a fully end-to-end model and directly controls the low-level throttle and steering angle. The action a is a continuous two-dimensional vector with entries in $[-1, 1]$. By multiplying coefficients and clipping the extreme value, the action will be converted into the engine force and steering angle for changing the vehicle states.

Reward and Cost Scheme. The reward function is composed of four parts as follows:

$$R = c_1 R_{disp} + c_2 P_{smooth} + c_2 P_{collision} + R_{term}. \tag{1}$$

The *displacement reward* $R_{disp} = d_t - d_{t-1}$, wherein the d_t and d_{t-1} denotes the longitudinal movement of the target vehicle in Frenet coordinates of the target trajectory between two consecutive time steps, providing a dense reward to encourage the agent to move forward. The *smooth penalty* $P_{smooth} = min(0, 1/v_t - |a[0]|)$ incentives the agent to drive smoothly and avoid a large steering value change between two timesteps, especially, when the velocity is high. v_t and $a[0]$ denote the current velocity and the steering value respectively. In addition, if a collision with a vehicle, human, or object happens at timestep t, the agent will receive a *collision penalty*. The penalty is set to $P_{collision} = 2$ when colliding with a human or vehicle and $P_{collision} = 0.5$ for colliding with an object like cones and barriers. We also define a sparse *terminal reward* R_{term}, which is non-zero

only at the last time step. At that step, we set $R_{disp} = R_{speed} = 0$ and assign R_{term} according to the terminal state. R_{term} is set to $+10$ if the vehicle reaches the destination, -5 for being $2.5m$ away from the reference trajectory. We set $c_1 = 2$, $c_2 = 1$ and $c_3 = 1$.

Termination Conditions and Evaluation Metrics. The episode will be terminated only when: 1) the agent drives $2.5m$ away from the reference trajectory 2) the agent arrives at the destination and 3) the agent can not finish the episode in $recorded\_episode\_length + 50$ steps. For each trained agent, we evaluate it in the held-out test environments and define the ratio of episodes where the agent arrives at the destination as the *success rate*. The definition is the same for *out of road* and *Timeout*. For evaluating the driving behavior, the mean velocity in each scenario is collected and then averaged across all scenarios. Also, a metric similar to the *success rate* is measured and called *route completion*, which is the ratio of moving distance to the length of the whole reference trajectory. Since each agent are trained across 5 random seeds, this evaluation process will be executed for 5 agent which has the same training setting but different random seeds. We report the average and std on the metrics mentioned above.

B.2 Curriculum Training System

In the single-agent experiments, 40,000 scenarios are used for training agents in both PG scenarios and nuPlan scenarios. Loading each scenario from scratch costs a significant amount of time, and thus we would like to buffer the scenarios in RAM for repeated use. However, the memory consumption is nonnegligible, especially when we train 5 policies concurrently and launch 20 workers to collect rollout for each policy. Assuming each scenario consumes about 10MB of memory, buffering 40,000 training scenarios in each worker process consumes 40,000$\times$100$\times$10MB=40,000GB=40TB of memory, which requests a powerful and expensive cluster to train agents in large-scale scenarios. We adopt **curriculum training** scheme for overcoming this issue, which reduces the memory by 99.9%.

Curriculum Training. We first sort the training scenarios according to the difficulty score calculated by: $track\_length \times cumulative\_curvature$. $track\_length$ is the moving distance of the data collection car. A greater moving distance corresponds to a higher velocity, which in turn indicates a higher difficulty score. This value then will be multiplied by a weight $cumulative\_curvature$ which quantifies the level of bending in the trajectory. After this, scenarios can be divided into 100 levels with 400 scenarios in each level. Only when the *Success Rate* reaches 75%, the worker will move to the next level and release the memory used to store scenarios of the previous level. To further reduce memory usage, we split the scenarios in each level into 20 subsets, and each worker only loads scenarios from the corresponding subset. Finally, only 0.05% scenarios (20 scenarios) are actually loaded in each worker, which saves memory by a large margin.

This curriculum training scheme not only makes it possible to finish training on a single server but boosts training efficiency and performance. We conduct an ablation study without the curriculum training, which costs a long time to train. This training scheme releases every scenario after using and reloading it when needed. The inferior performance highlights the necessity of having curriculum training.

B.3 Results

As shown in Fig. 9, we present experiment results for three training settings: *nuPlan with curriculum training*, *nuPlan without curriculum training*, *PG with curriculum training*. The three used databases are *PG-train*, *nuPlan-train*, and *nuPlan-test*, comprising 40,000, 40,000, and 5,000 scenarios respectively. Considering *nuPlan with curriculum training* as the baseline, our analysis focuses on two factors: the data source and the inclusion of curriculum training. The evaluation of real-world scenarios underscores the significance of training with in-distribution real-world data and employing a curriculum training system.

Actually, the poor performance of *nuPlan without curriculum training* is also exposed at the training stage. In spite of increasing the *data coverage*, the agents under this setting always have a poor *training success rate* and *training route completion* throughout the whole training process. However,

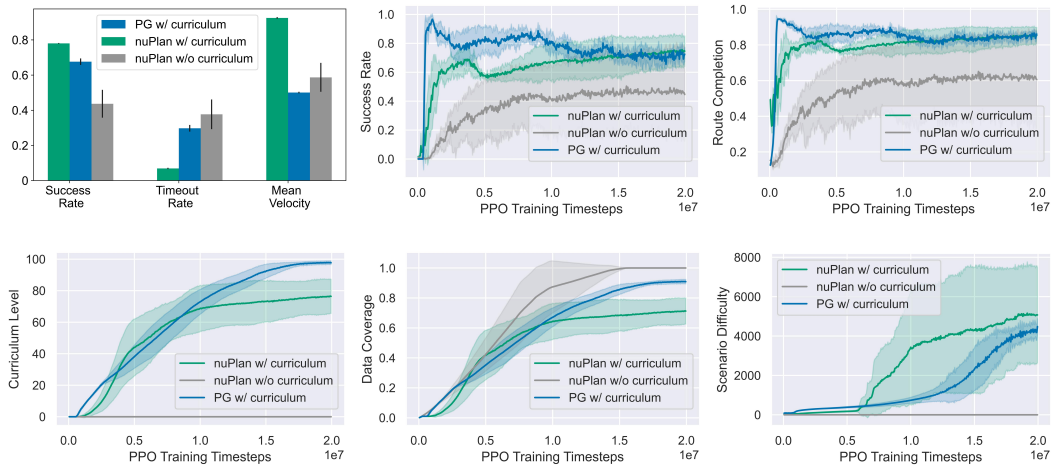

Figure 9: The upper left figure shows the evaluation results for three sets of agents trained in different settings. The other two upper figures show the learning dynamics including *training success rate* and *training route completion*. The bottom figures show the status of the curriculum training system. The *curriculum level* is calculated as the average worker level across 5 seeds, with 20 workers per seed. *Data coverage* refers to the proportion of scenarios in the training set that the agent encounters throughout the entire training process. A value of 1.0 indicates that the agent has visited all scenarios. *Scenario difficulty* is determined by calculating the average difficulty score across all scenarios collected during a PPO optimization epoch.

agents trained in *nuPlan with curriculum training* setting are more stable and have an increasing *training success rate* as they can fully exploit the scenarios which properly match their current driving ability.

For investigating the influence of data sources, the conclusion can be drawn only from the test results which show that agents trained under *PG with curriculum training* can not generalize to real-world settings well. And the failure mode is that agents trained in synthetic scenarios can not learn to drive at high speed. And they don't move until other vehicles move far from them. This behavior is reflected on the *test mean velocity* as well.

Another interesting phenomenon is the drop of *training success rate* when the *scenario difficulty* and *curriculum level* increase, which is also reported in [8] as well. And this value finally coverage to the test *success rate*. Therefore, we infer that given enough training scenarios, the test performance of learning-based agents can be roughly reflected by the training-time performance.

The hyper-parameters of the RL training are listed in Table 6

Table 6: PPO

Hyper-parameter	Value
KL Coefficient	0.2
λ for GAE [39]	0.95
Discounted Factor γ	0.99
Number of SGD epochs	20
Train Batch Size	50,000
SGD mini batch size	200
Learning Rate	1e$-$4
Clip Parameter ϵ	0.2
Activation Function	"tanh"
MLP Hidden Units	[512, 256, 128]
MLP Layers	3

C Multi-agent Policy Learning

C.1 Experiment Setting

We can load the real-world dataset into MetaDrive simulator and create multi-agent interactive policy environment. We first instantiate agents in the environment where their initial states are loaded from the real-world dataset. The initial states include position, heading, velocity and the size. Then, we assign `EnvInputPolicy` to all agents and allow them to be controlled by external RL policies.

The ground-truth (GT) trajectories are not accessible to the learning agents but serve as the supervision via reward function (in RL) or observations (in IL).

Concretely, we create the reward function similar to Eq. 1 with slightly different weights:

$$R = R_{disp} + P_{collision} + R_{term}. \tag{2}$$

The *displacement reward* $R_{disp} = d_t - d_{t-1}$, wherein the d_t and d_{t-1} denotes the longitudinal movement of the target vehicle in Frenet coordinates of the **target trajectory** between two consecutive time steps, provides a dense reward to encourage the agent to move forward. In addition, if a collision with a vehicle, human, or object happens at timestep t, the agent will receive a *collision penalty*. The penalty is set to $P_{collision} = 1$ when colliding with a human or vehicle. We also define a sparse *terminal reward* R_{term}, which is non-zero only at the last time step. At that step, we set $R_{disp} = R_{speed} = 0$ and assign R_{term} according to the terminal state. R_{term} is set to $+10$ if the vehicle reaches the destination, -1 for being $10m$ away from the reference trajectory. We transform the GT trajectory into the simulator to create the target trajectory and use the displacement reward as the primary supervision from the dataset.

On the other hand, in multi-agent imitation learning we use the GT trajectory to form a dataset of observations and follow the setting of learning from observations or say action-free imitation learning. Specifically, for each frame in the scenario, we load the states of all actors at this frame into the simulator and utilize the sensor simulation functionality of MetaDrive to simulate the observations of each actor. The observation follows Sec. B.1. We form the dataset of observation sequences of all actors in the dataset.

The ground-truth trajectory can be used to measure the learned behaviors. The the route completion rate is the ratio between the length of projected agent trajectory and the length of GT trajectory. The average distance between agent trajectory and the GT trajectory is computed as follows:

$$\frac{1}{T}\sum_{t=1}^{T} ||pos_{agent,t} - pos_{GT,t}||. \tag{3}$$

where T is the minimum length of agent trajectory and the length of GT trajectory. Therefore, for the agents that terminate quickly after spawning, the average distance will be quite small. The final distance is computed as distance between the last position of GT trajectory and the last position of agent trajectory. The cost is the number of crashes of an agent in one episode. There are two terminal conditions. If the route completion rate exceeds 95%, the agent is marked successful. If the agent moves out of $10m$ aways from the reference trajectory, the agent is marked out of road and failed. We don't terminate agent's episode if it crash with other objects.

C.2 Baseline Details

MA-GAIL. We use GAIL [23] in multi-agent setting [43] but the discriminator distinguishes state-next state pair, instead of state-action pair [47]. We use PPO as the underlying RL algorithm in GAIL. The hyper-parameters are listed in Table 7.

MA-AIRL. We also train multi-agent Adversarial Inverse RL [59] (AIRL) with an additional inverse dynamics model for estimating the expert actions. The inverse dynamics model is trained concurrently with the AIRL policies and learns the action given state-next state pairs from the environment interactions. AIRL will learn a reward function and use the reward function to build a

discriminator as GAIL. We also use PPO as the underlying RL algorithm. The hyper-parameters are listed in Table 8.

MARL baselines. We train independent PPO [40, 38] and TD3 [21] agents as well as the Coordinated Policy Optimization (CoPO) agents [36] as MARL baselines. The hyper parameters are given in the next section.

C.3 Hyper-parameters

Table 7: Action-free Multi-agent GAIL

Hyper-parameter	Value
Discriminator LR	5e-5
Discriminator L2 Norm	1e-5
Discriminator SGD Num Iters	1
Discriminator SGD Minibatch Size	1024
PPO SGD Minibatch Size	512
PPO Batch Size	2000
PPO LR	1e-4
PPO SGD Num Iters	10
PPO Clip Parameter	0.2
PPO Lambda	0.95
PPO Gamma	0.99

Table 8: Action-free Multi-agent AIRL

Hyper-parameter	Value
Discriminator Loss LR	3e-4
Discriminator L2 Norm	1e-5
Discriminator SGD Num Iters	5
Discriminator SGD Minibatch Size	1024
Inverse Dynamics SGD Num Iters	100
Inverse Dynamics SGD Minibatch Size	512
Inverse Dynamics LR	1e-4
PPO SGD Minibatch Size	512
PPO Batch Size	2000
PPO LR	1e-4
PPO SGD Num Iters	10
PPO Clip Parameter	0.2
PPO Lambda	0.95
PPO Gamma	0.99

Table 9: PPO

Hyper-parameter	Value
PPO SGD Minibatch Size	512
PPO Batch Size	2000
PPO LR	1e-4
PPO SGD Num Iters	10
PPO Clip Parameter	0.2
PPO Lambda	0.95
PPO Gamma	0.99

Table 10: CoPO

Hyper-parameter	Value
LCF LR	1e-4
LCF Num Iters	5
Neighborhood Distance	40 m
PPO SGD Minibatch Size	512
PPO Batch Size	2000
PPO LR	1e-4
PPO SGD Num Iters	10
PPO Clip Parameter	0.2
PPO Lambda	0.95
PPO Gamma	0.99

Table 11: TD3

Hyper-parameter	Value
Critic LR	1e-4
Actor LR	1e-4
Tau for Target Update	5e-3
Train Batch Size	100

D AD stack testing

D.1 ROS bridge

As shown in Fig. 10, We provide a ROS bridge along with the *ScenarioNet*, allowing users from the ROS community to develop and test their systems with massive real-world data. As shown in Fig. 11, information like camera, lidar, and mid-level representations can be retrieved from the simulation.

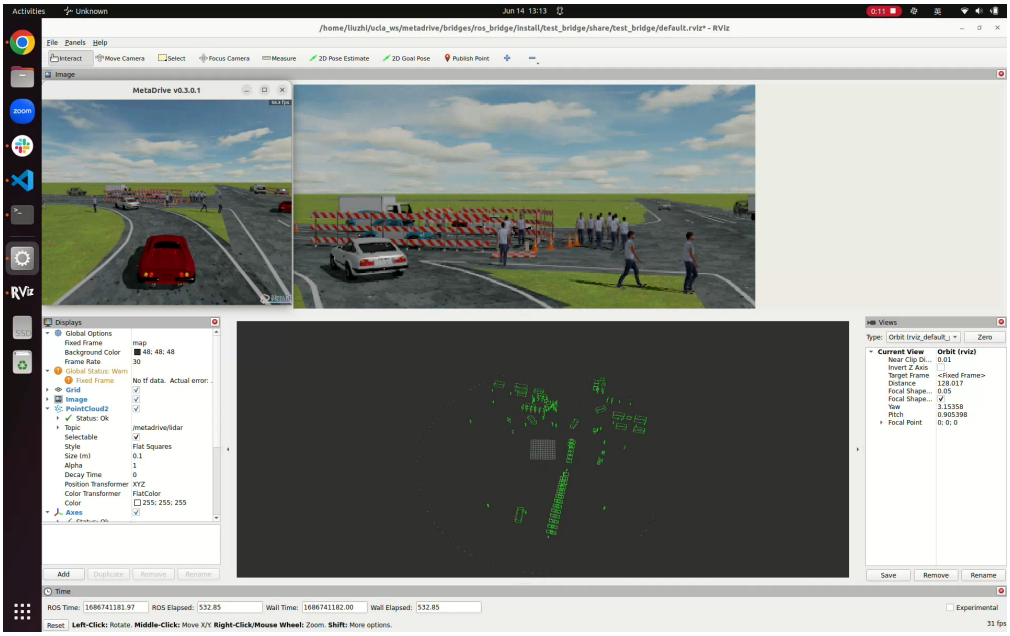

Figure 10: The interface of ROS bridge allowing connecting *ScenarioNet* and ROS community.

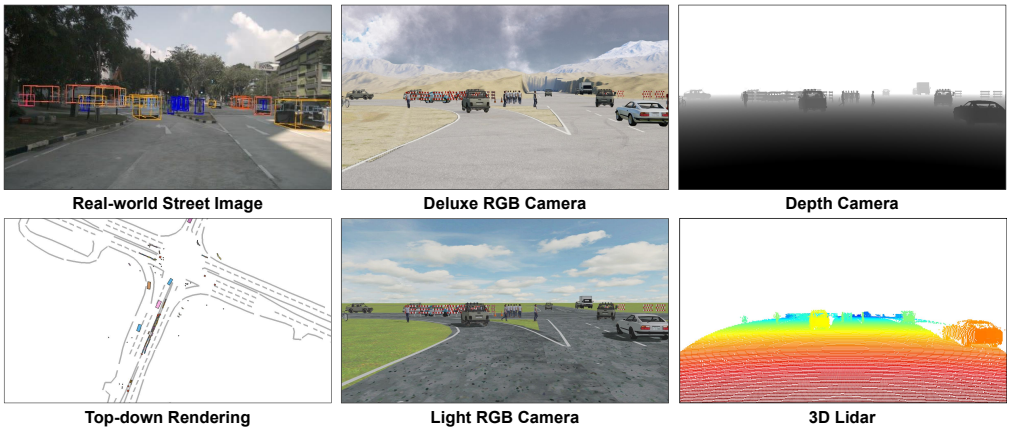

Figure 11: Multimodal sensory data provided by *ScenarioNet*.

ScenarioNet provides not only mid-level scenario representations but multiple sensor outputs like top-down view, RGB camera, depth camera, and cloud points. The deluxe RGB camera is supported by the deferred rendering pipeline. This figure shows the `scene-0061` from nuPlan-mini-split and its digital twin counterparts. It is worth investigating how to reconstruct meshes from the recorded lidar cloud points and images so that we can simulate the sensor output from new views different from the recorded ones. Besides, it is a promising topic to study the **closed-loop** sensor fusing and learning-based control together, which can be supported by *ScenarioNet*.

D.2 Qualitative Results of Openpilot test

Openpilot [9] is an end-to-end solution for driver assistance. Therefore, a platform providing 3D rendering is necessary, if one would like to study or test the system. *ScenarioNet* is the only one that provides both real-world scenarios and 3D graphics support. As its navigation module is still a beta version, we only test Openpilot in scenarios without diverged roads. We build a small database containing mainly lane-keeping scenarios for conceptually demonstrating that *ScenarioNet* can connect with commercial AD stack. As shown in Fig. 12, the Openpilot system is robust to common scenarios like turning right, lane-keeping, stopping at traffic lights, and passing traffic lights. The demo video is available at https://youtu.be/KjlPB0nCTvg

Figure 12: Openpilot manages to overcome four representative scenarios.

E Scenario Databases

In the database statistics Table, the *Intersection Ratio* is the ratio of scenarios having traffic lights for real-world data, and the ratio of scenarios containing intersections and roundabouts for synthetic data.

E.1 Waymo Database

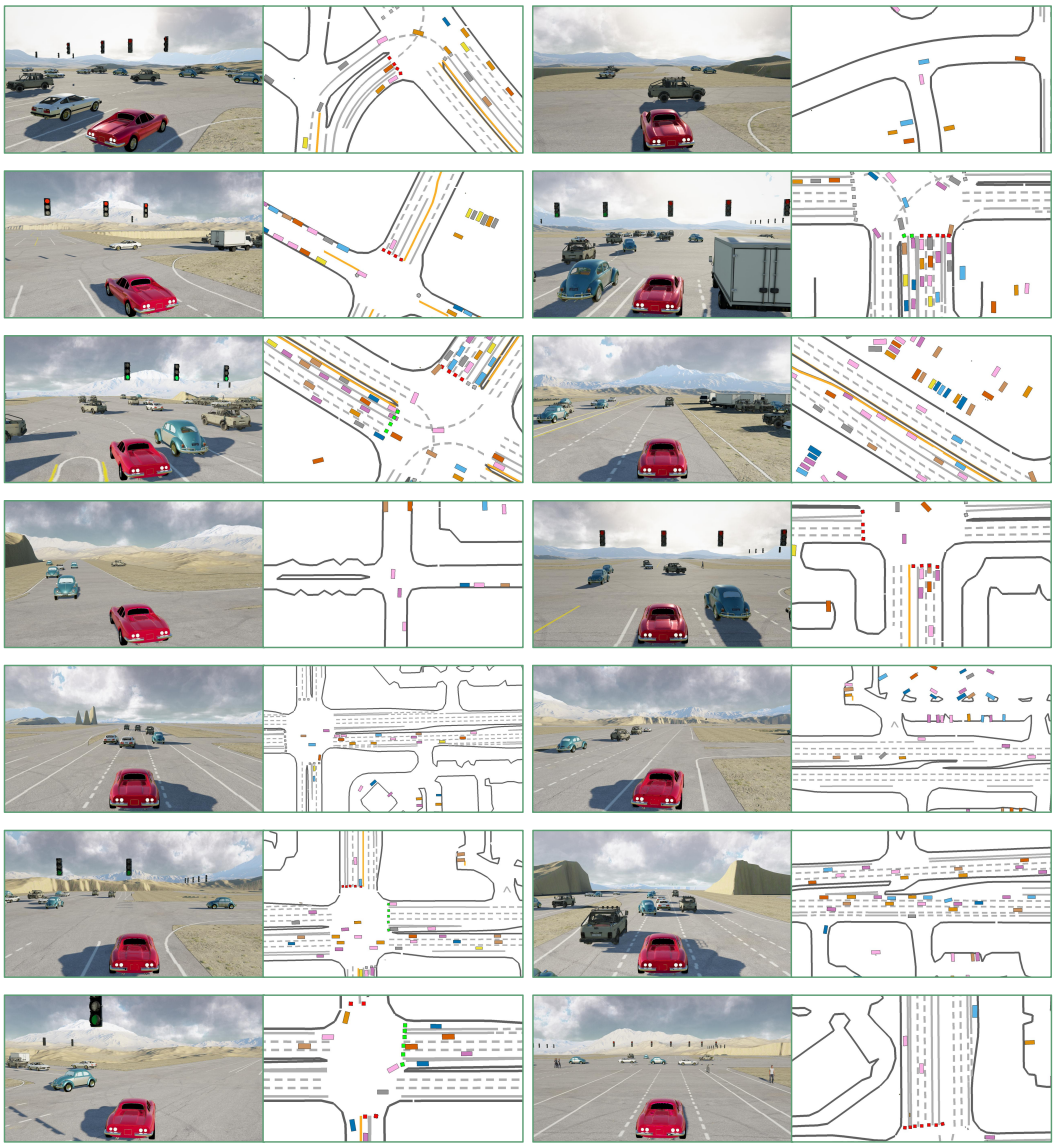

Figure 13: Rendered traffic scenarios and their top-down views from the Waymo database.

We construct the Waymo database from the **20 seconds** version [44] motion data with Google Cloud path at `waymo_open_dataset_motion_v_1_2_0/uncompressed/scenario/training_20s`. As Waymo data is collected with the altitude calibrated, we can filter the overpass scenarios by excluding the scenarios with significant changes in height. In addition, we exclude the scenarios where the ego car waits at the red light for a long time by selecting scenarios where the ego car moving distances are greater than 10 meters. We provide the rendered scenario examples from this dataset in Fig. 13. Top-down views show that Waymo scenarios contain diverse road structures and a large number of vehicles.

E.2 nuPlan Database

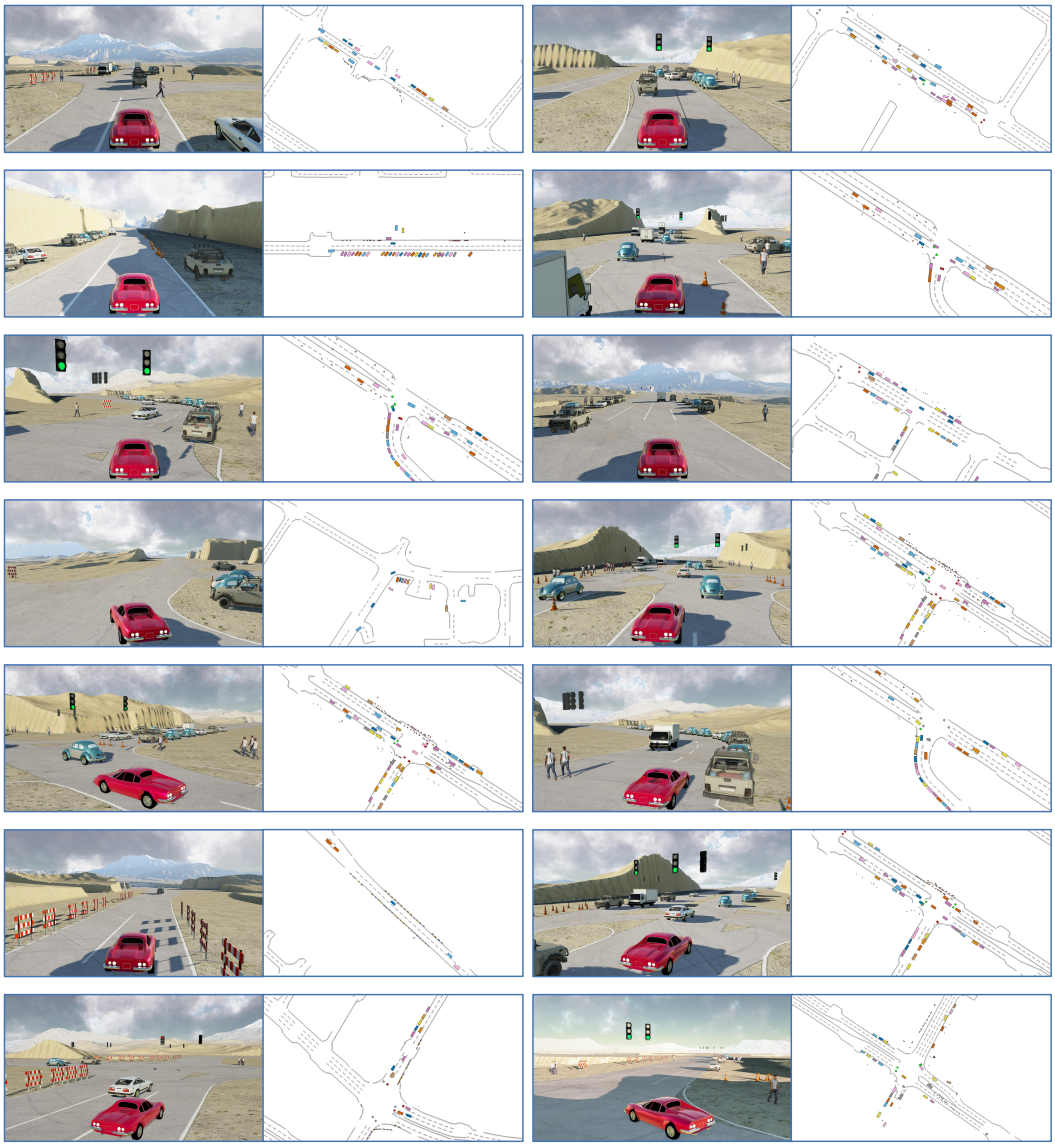

Figure 14: Rendered traffic scenarios and their top-down views from the nuPlan database.

According to https://www.nuscenes.org/nuplan, nuPlan has more than 1500 hours of driving data collected in 4 different cities: Las Vegas, Singapore, Pittsburgh, and Boston, here we only use the data collected in Boston as we want to keep all databases in experiments to have similar sizes. Data collected in other cities can also be converted to our scenario format.

We use the V.1.0 version data, which contains approximately 50,000 scenarios after excluding scenarios where the ego car moving distances are less than 10 meters. It is noticeable that nuPlan updated the data to version v1.1 recently (one week before the NeurIPS 2023 dataset track deadline), which may induce some differences if building a database from this new version. As shown in Fig. 14, the scenario examples reveal that *nuPlan Boston split* has cluttered scenes and contains many traffic cones and barriers besides vehicles. We additionally find that in some nuPlan scenarios, the ego car trajectory is out of the drivable area for sidestepping the barriers or cones.

E.3 PG Database

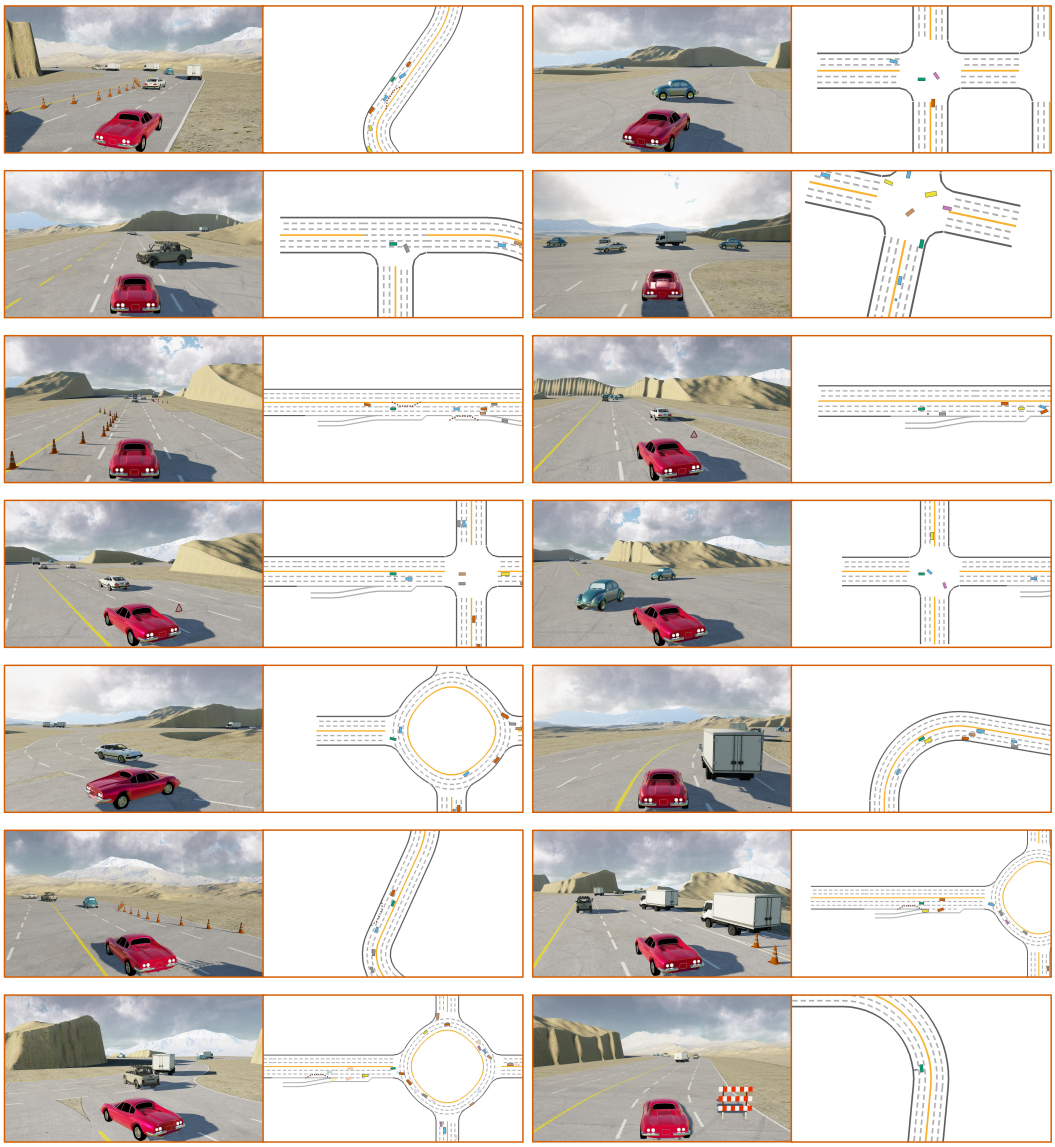

Figure 15: Rendered traffic scenarios and their top-down views from the PG database.

Unlike the previous two datasets collected in the real world, PG scenarios are synthesized according to a set of rules. For map generation, two blocks are sampled from a set of candidate roadblocks and connected to form a map. Those blocks include intersections, roundabouts, straight roads, curved roads, Ramp, and so on. Once the map is defined, a traffic generation rule will be applied to scatter vehicles and road objects like traffic cones on the map. The detailed scenario generation config such as the block distribution for sampling can be found at https://github.com/metadriverse/metadrive/blob/0a929f8130b34e4428067390f20f872d1d6d224a/metadrive/component/algorithm/blocks_prob_dist.py#L4. All vehicles choose a destination automatically and be actuated by IDM policy which can keep a proper distance from the front vehicle and perform a lane change when the front object is static. The scenario data is collected by an IDM policy as well. Some scenario examples are shown in Fig. 15.

F Discussion: how to improve visual fidelity?

The visual fidelity of the closed-loop simulation can be further improved with the collected raw sensor data. We already have some plans to improve it and will include this discussion in the paper for sharing our thoughts regarding realistic sensor simulation.

Currently, the sensor simulation including the camera and lidar is achieved in a Computer Graphics (CG) way, where people try restoring the mesh and texture for objects from real-world data and shading these models based on lights and materials to make them visually realistic. To improve the rendering results of *ScenarioNet*, we do plan to reconstruct the geometry data for traffic participants and objects from the driving videos. Besides, we also consider connecting *ScenarioNet* with Unreal Engine or Nvidia Omniverse in the future as they could provide better shading results.

On the other hand, NeRF is an alternative to improve the quality of sensor simulation. Through volume rendering, it can directly synthesize new camera views and point clouds from the driving videos when traffic participants and the ego car move with different trajectories and poses in the closed-loop training. This way is purely data-driven and can exempt the need for restoring 3D assets like objects and buildings. Recent results [55, 58, 54] already demonstrate its potential in terms of camera and lidar simulation. However, how to make the NeRF scene editable is still an open problem; hence, we plan to investigate this in the future.