MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning

Quanyi Li[§*], Zhenghao Peng[†*], Lan Feng[‡], Qihang Zhang[†], Zhenghai Xue[†], Bolei Zhou[♠]
[§]Centre for Perceptual and Interactive Intelligence, Hong Kong SAR, China,
[†]The Chinese University of Hong Kong, Hong Kong SAR, China, [‡]ETH Zurich, Switzerland,
[♠]University of California, Los Angeles, USA.

Abstract—Driving safely requires multiple capabilities from human and intelligent agents, such as the generalizability to unseen environments, the safety awareness of the surrounding traffic, and the decision-making in complex multi-agent settings. Despite the great success of Reinforcement Learning (RL), most of the RL research works investigate each capability separately due to the lack of integrated environments. In this work, we develop a new driving simulation platform called MetaDrive to support the research of generalizable reinforcement learning algorithms for machine autonomy. MetaDrive is highly compositional, which can generate an infinite number of diverse driving scenarios from both the procedural generation and the real data importing. Based on MetaDrive, we construct a variety of RL tasks and baselines in both single-agent and multi-agent settings, including benchmarking generalizability across unseen scenes, safe exploration, and learning multi-agent traffic. The generalization experiments conducted on both procedurally generated scenarios and real-world scenarios show that increasing the diversity and the size of the training set leads to the improvement of the RL agent's generalizability. We further evaluate various safe reinforcement learning and multi-agent reinforcement learning algorithms in MetaDrive environments and provide the benchmarks. Source code, documentation, and demo video are available at https://metadriverse.github.io/metadrive.

Index Terms—Reinforcement Learning, Autonomous Driving, Simulation.

1 Introduction

Great progress has been made in reinforcement learning (RL), ranging from super-human Go playing [48] to delicate dexterous in-hand manipulation [2]. Recent progress shows the promise of applying RL to real-world applications, such as autonomous driving [16], [18], [50], the power system optimization in smart building [29], the surgical robotics arm [40], and even nuclear fusion [9]. However, generalization remains one of the fundamental challenges in RL. Even for the common driving task, an agent that has learned to drive in one town often fails to drive in another town [10]. There is the critical issue of model overfitting due to the lack of diversity in the existing RL environments [8]. Many ongoing efforts have been made to increase the diversity of the data produced by the simulators, such as procedural generation in gaming environments [7] and domain randomization in indoor navigation [23]. In the context of autonomous driving (AD) research, many realistic driving simulators [4], [10], [28], [46], [61] have been developed with their respective successes. Though these simulators address many essential challenges in AD, such as the realistic rendering of the surroundings in CARLA [10] and the scalable multi-agent simulation in SMARTS [61], they do not successfully address the aforementioned generalization problem in RL, especially the generalization across different scenarios. Since the existing simulators mostly adopt fixed assets such as hand-crafted traffic driving rules and maps, the scenarios available for training and testing are far from enough to catch up with the complexity of the real world.

To better benchmark the generalizability of RL algorithms in the autonomous driving domain, we develop MetaDrive, a driving simulation platform that can compose a wide range of diverse traffic scenarios. MetaDrive holds the key feature of compositionality, inducing a diverse set of driving scenarios from procedural generation (PG), real data importing, and domain randomization. It also supports versatile research topics like safe RL and multi-agent RL. Every asset in MetaDrive, such as the vehicle, the obstacle, or the road structure, is defined as an interactive object with many configurable settings restricted in the parameter spaces, which can be combined, managed, and actuated to compose new interactive driving scenarios. As shown in Fig.1 **A.** and Fig.1 **B.**, through the procedural generation or the real data importing, interactive and executable scenarios can be generated to train and test learning-based driving systems.

Another challenge for generalizable RL research is the scalability issue [7]. Conducting experiments to benchmark the generalizability of RL agents requires massive computing resources and a long training time [8]. To tackle this, MetaDrive incorporates a trade-off between the visual rendering and the physical simulation. With the resource saved from photorealistic rendering, a single MetaDrive instance with 100 MB size can run realistic physical simulation up to 300 FPS on a standard PC. The developed platform is also compatible with common RL training frameworks such as RLLib [25] and Stable-Baseline3 [38], which can launch more than 100 instances to generate batch data in parallel. Furthermore, MetaDrive can be installed with a single command line and interact with OpenAI Gym API in the Python environment. With

[*]*Quanyi Li and Zhenghao Peng contribute equally to this work.*

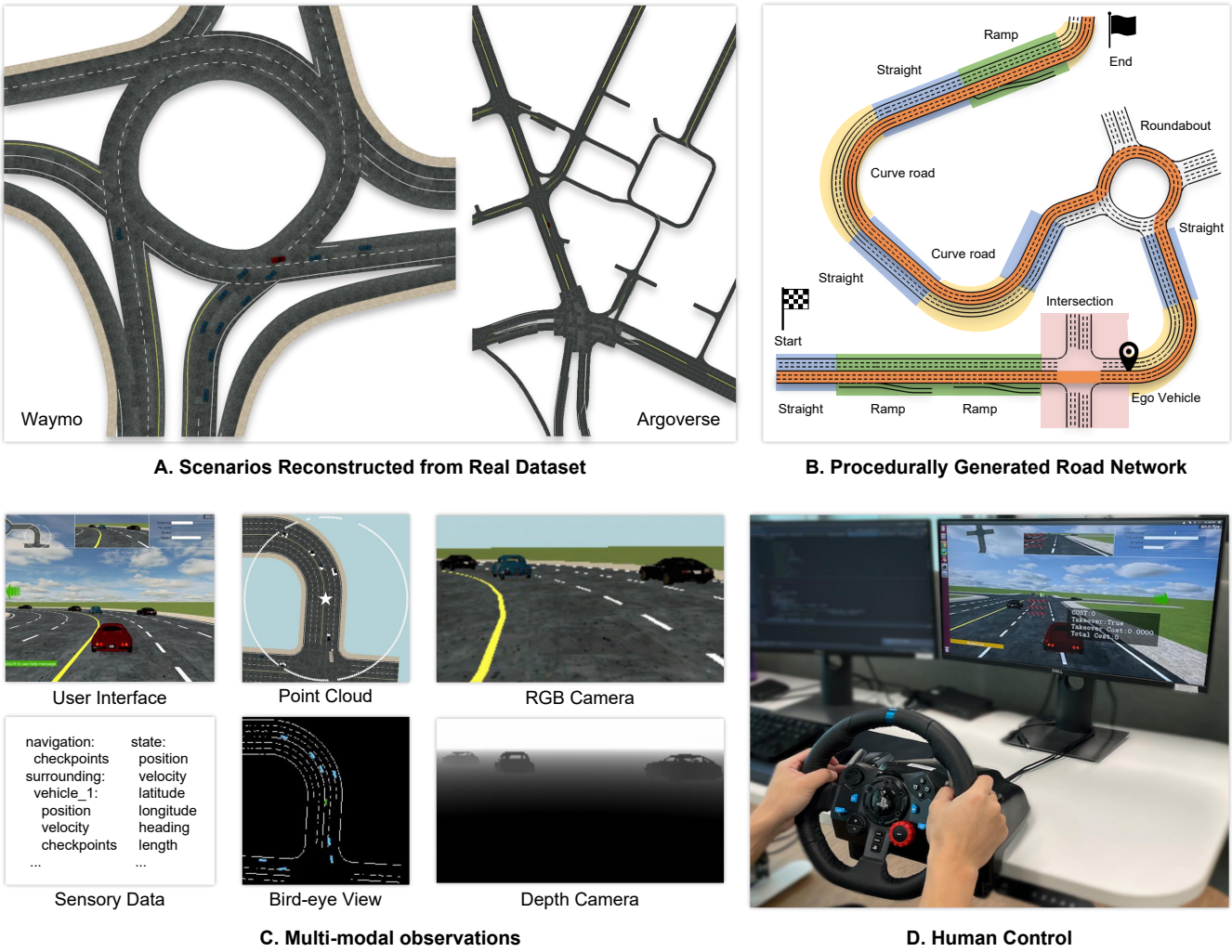

Fig. 1. **A.** MetaDrive supports importing maps and traffic flow from real-world dataset. **B.** A road map procedurally generated from elementary road blocks. **C.** Multi-modal observations provided by MetaDrive, including Lidar-like point clouds, RGB / depth camera, bird-view semantic map, and scalar sensory data. **D.** MetaDrive supports the control and demonstration from human subject.

those features, MetaDrive aims to facilitate the development of generalizable RL algorithms.

MetaDrive supports a variety of reinforcement learning tasks because of its compositionality and extensibility. In the current development stage, we construct four standard RL tasks and implement their baselines in the context of driving research as below. The first three tasks are in single-agent setting while the last one is in multi-agent setting:

- **Generalization to unseen PG scenarios**. Based on procedural generation, our simulator composes a large number of diverse driving scenarios from the elementary road structures and traffic vehicles. Those maps and scenarios are further split into training and test sets. we then conduct baseline experiments to evaluate the generalizability of different RL methods with respect to the road network structures and traffic flows.
- **Generalization to unseen real scenarios**. MetaDrive has the APIs to import map data collected in real-world such as Waymo dataset [26], [51] and Argoverse dataset [5]. Thus we can benchmark the generalizability of RL agents in real traffic scenarios. These real-world cases are closely examined and selected by humans to ensure the data fidelity, then they are divided into standard training and test set. Agents trained in scenes generated by PG can also be evaluated in real traffic cases to verify the effectiveness of PG, since the scenes are composed under a unified structure.
- **Safe exploration**. We study the safe exploration problem in RL, where the agent has to drive under safety constraints. Besides the surrounding vehicles, obstacles like broken-down vehicles and traffic cones are randomly scattered on the road and a cost is yielded if a collision happens. We test several safe RL algorithms to measure their applicability to the safety-critical scenarios.
- **Multi-agent traffic simulation**. In five typical traffic scenarios such as roundabout and intersection, we study the problem of multi-agent RL for dense traffic simulation. There are 20 to 40 agents in a scene, where each vehicle is actuated by a continuous neural control policy. Collective motions can emerge from the coordination of the agents.

2 Related Work

RL Environments. A large amount of RL environments have been developed to benchmark the progress of different RL problems. Arcade Learning Environments [3] and the MuJoCo Simulator

TABLE 1
Comparison of representative driving simulators.

Simulator	Unlimited Scenarios	Vehicle Dynamics	Multi-agent Support	Custom Map	Lidar or Camera	Real Data Importing	Realistic Visual Rendering	In Active Maintenance	Lightweight
CARLA [10]		✓	✓	✓	✓	✓	✓	✓	
SUMMIT [4]		✓	✓	✓	✓	✓	✓	✓	
MACAD [34]		✓	✓	✓	✓	✓	✓	✓	
GTA V [28]		✓			✓		✓		
Highway-env [22]								✓	✓
TORCS [57]		✓		✓	✓				✓
Flow [55]				✓		✓			✓
CityFlow [60]			✓	✓		✓			✓
Sim4CV [31]		✓			✓		✓		
Duckietown [6]					✓	✓		✓	✓
SMARTS [61]		✓	✓	✓	✓			✓	✓
AIRSIM [47]		✓			✓	✓	✓	✓	
SUMO [27]				✓		✓		✓	✓
MADRaS [42]		✓	✓	✓	✓				✓
Udacity [53]		✓		✓	✓				
DriverGym [20]						✓		✓	✓
DeepDrive [52]		✓			✓		✓		
MetaDrive	✓	✓	✓	✓	✓	✓		✓	✓

[54] are widely studied in single-agent RL tasks, and in multi-agent RL the Particle World Environment [30] and SMAC [41] have become the standard testing grounds. Meta-World [59] is developed for multi-task and meta reinforcement learning. To apply RL algorithms in the real world, one has to consider their safety and robustness in previously unseen scenarios. Safety Gym [39] and ProcGen [7] are proposed to benchmark the safety and generalizability of RL algorithms, respectively. Similar to ProcGen [7] which uses procedural generation to generate distinct levels of video games, our MetaDrive simulator can also procedurally generate an infinite number of driving scenes. As shown in the following sections, most of the aforementioned RL tasks can be implemented and studied in the proposed MetaDrive simulator because of its flexibility for extension and customization.

Driving Simulators. Because of public safety concerns, it is difficult to train and test the driving policies in the physical world on a large scale. Therefore, simulators have been used extensively to prototype and validate autonomous driving research. Apart from MetaDrive, there are lots of existing driving simulators that support RL research. Table 1 presents a comprehensive comparison between different simulators. The simulators GTA V [28], Sim4CV [31], AIRSIM [47], CARLA [10] and its derived project SUMMIT [4], MACAD [34] realistically preserve the delicate appearance and transition of the physical world, such as lighting conditions, weather system, even the time of a day shift. In contrast to the high-fidelity rendering of these simulators, MetaDrive trades off the visual appearance quality for its high sample efficiency and extensibility, in order to support a wide range of research topics such as safe RL and multi-agent autonomy. MetaDrive has realistic vehicle dynamics and kinematics where chassis, wheels and joints are built on constraints and actuated by Bullet physics engine. This feature distinguishes it from other simulators, such as Udacity [53], TORCS [57], DeepDrive [52], Duckietown [6], Highway-env [22] and DriverGym [20], which only simulate vehicle with simple kinematic model, *e.g.*, Bicycle mode [37], and simplify the driving problem to a high-level discrete control task or provide simplistic driving scenarios. The aforementioned simulators are mainly designed for the single-agent scenario. In the Multi-agent Reinforcement Learning(MARL) context, CityFlow [60] and FLOW [56] are two macroscopic traffic simulators that are based on SUMO [27] focusing on traffic flow simulation and hence are not suitable to investigate the detailed behaviors of each learning-based agent. For MARL research, MACAD [34] and MADRaS [42] are two traditional platforms which are built on CARLA [10] and TORCS [57] respectively. The newly developed SMARTS [61] provides an excellent testbed for the interaction of RL agents and social vehicles in atomic traffic scenes. MetaDrive also designs MARL environments that cover complex scenarios such as tollgate and parking lot, with a dense population of 50+ agents. Compared to the existing simulators, the proposed MetaDrive holds the key feature of compositionality and extensiblity for fostering new research opportunity in generalizable reinforcement learning.

3 System Design of MetaDrive

The core feature of MetaDrive is its compositionality. This feature comes in two folds: the abstraction of low-level system implementation and the high-level aggregation of elementary components into traffic scenarios. In low-level implementation, we encapsulate the back-end implementation of the basic components and the intercommunication between Python API and the extremely efficient 3D game engine Panda3D [12]. With the high-level APIs, we implement various methods to generate traffic flows and diverse road networks. The wrapping of back-end simulation makes it

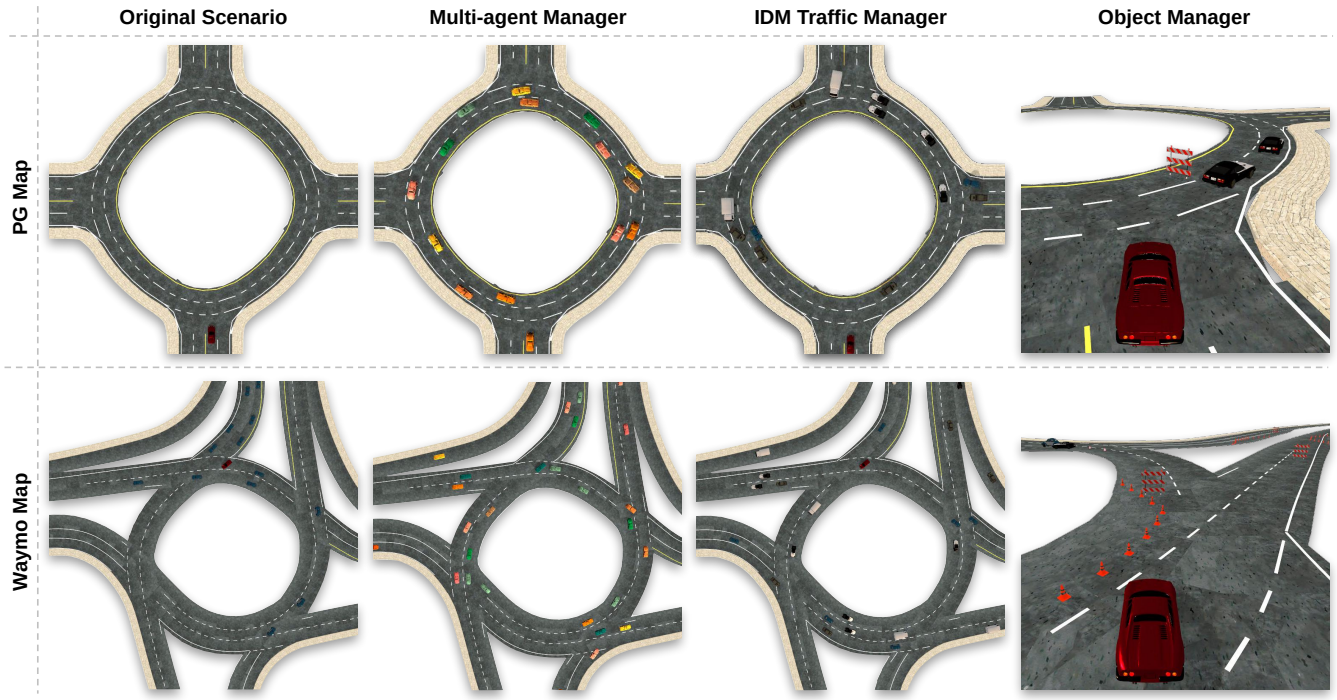

Fig. 2. MetaDrive can compose new scenarios by combining various managers. The first column shows two original cases from PG and Waymo datasets. In the second column, by replacing the Single-agent Manager with the Multi-agent Manager, we compose the multi-agent navigation tasks in two maps. In the third column, IDM Traffic Manager is added to both maps, enabling responsive traffic vehicles following IDM policies. The last column indicates that it also supports composing safety-critical environments on both PG map and real map by adding the Object Manager.

convenient and flexible for RL researchers to develop new scenarios and tasks to prototype their methods in the autonomous driving domain. The diverse driving scenarios in MetaDrive also become the standard environments to benchmark various RL algorithms. In this section, we first introduce the key designs in MetaDrive and the abstraction of object classes. We then introduce the procedure of composing diverse scenarios from the elementary components.

3.1 Core Concepts

Object. In MetaDrive, we abstract the object as the intermediate connector between the simulation engine and the Python environment. Object is the basic entity in driving scenarios, such as vehicle, obstacle, traffic light, and road structure. In the back-end simulation, an object is a proxy to two internal models: the *physical model* and the *rendering model*. Powered by the Bullet engine, the physical model in various shapes has a rigid body that can participate in collision detection and motion calculation. For retrieving realistic camera data, the rendering model provides fine-grained rendering effects such as light reflection, texturing, and shading.

In the Python environment, the object class wraps the aforementioned details, handles the events such as garbage collection and exposes only the high-level APIs for manipulating the object and retrieving its information directly, such as `object.set_state()`, `object.get_state()`, `object.step()`, and `object.reset()`. This design can not only achieve efficient and realistic simulation but enable users to manipulate objects and acquire their information with one line Python code. Each class of objects has a parameter space that bounds the possible configurations of the instantiated objects, enabling randomizing and diversifying objects. For instance, a vehicle has parameters such as wheel friction, suspension stiffness, color, size, and so on. The parameters can be directly determined from user-specified configuration or randomly sampled in the parameter space. The object will automatically assign the determined parameters to the internal models. With such design, a developer who wants to create and manipulate an object, such as setting the location and velocity of a vehicle, increasing the width of a lane, or getting nearby objects, can simply call the APIs without touching the complex back-end simulation system.

Policy. A policy is a function that takes the object and environmental states as input and determines the action for controllable objects or new states for static objects. In each time step, policy of each object will be automatically called and the control signals, namely the actuation for vehicle or new position for obstacle, are passed to the objects by calling `object.step()` or `object.set_state()` functions. The simulation engine then executes those control signals and advances the world for one step.

For vehicles, we currently implement a rule-based policy that mixes the cruising, lane changing, emergency stopping behaviors with various driving models such as IDM and mobile policy [19]. Furthermore, a well-trained PPO policy [45] is encapsulated in the simulator, serving as another way to control traffic vehicles. Additionally, MetaDrive provides policies that accept commands from external controllers such as neural policy and human control. Besides the traffic participants, the road facilities can be also controlled by policies. For instance, in the MARL Tollgate environment Fig. 7 **B.**, the tollgate follows *Tollgate Policy* which releases the vehicle after it halts for a few seconds inside the gate.

The relationship between policies and objects is defined by users and can be varied during the episode, making it possible to do a mix-policy simulation. Similar to SMARTS [61], the mix-traffic simulation can be easily implemented by, for example, creating a

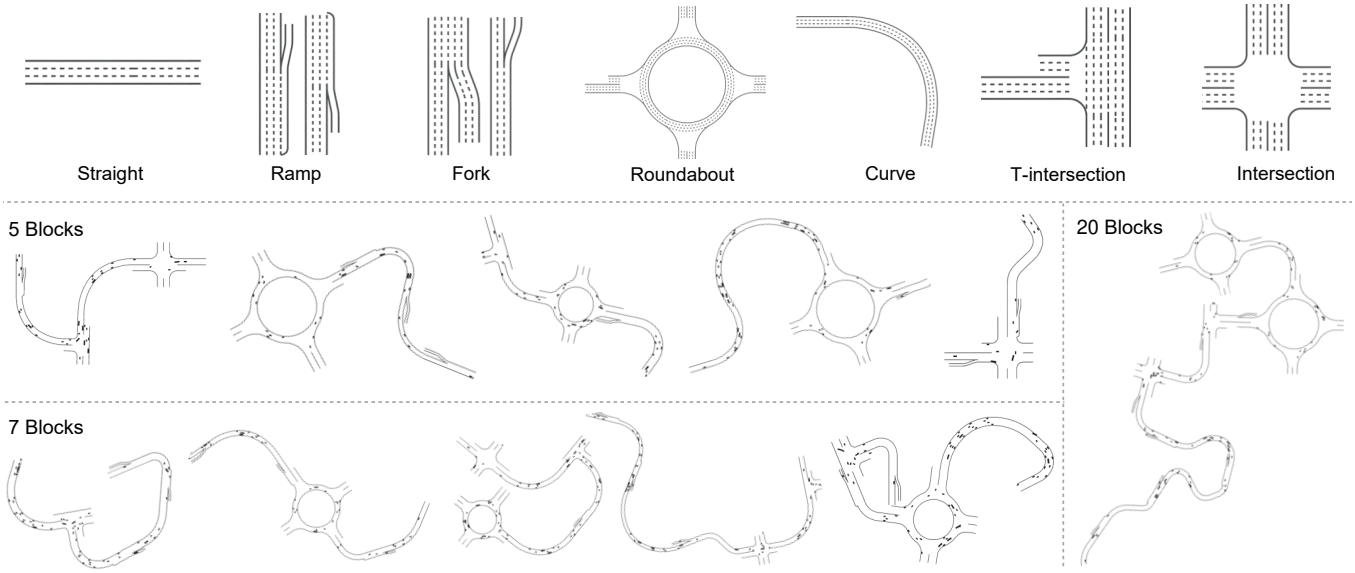

Fig. 3. First row shows the basic road blocks and the other rows plot the generated maps with different block numbers.

scenario where some vehicles are controlled by neural policies, the other vehicles are actuated by IDM policies.

Manager. Different managers are used to manage objects in different roles. In MetaDrive, same class of objects might have different roles. For example, in a single-agent RL environment, though the ego vehicle and the traffic vehicles are all vehicle objects, they have different roles and thus should have different policies. Different roles require various data processing pipelines, policies, and spawning/recycling rules. The ego vehicle requires the environment to provide explicit fine-grained surrounding information and is controlled by the RL agent. In contrast, the traffic vehicles rely on the rule-based policy and can be actuated with less detailed observations and states of other objects. Their spawning rules are also different. The ego vehicle is only created when an episode resets, while traffic vehicles will be recycled if they locate far from the ego vehicle. Therefore, the ego vehicle and the traffic vehicles should be managed by the *Agent Manager* and the *Traffic Manager* respectively. A custom manager should implement `manager.get_state()` and `manager.set_state()` for retrieving and setting the states of managed objects, `manager.step()` for invoking policies and actuating managed objects and `manager.reset()` for resetting objects when meeting the termination conditions.

3.2 Workflow for Scenario Composition

Scenario composition in MetaDrive is conducted hierarchically: MetaDrive environment creates a set of managers according to user specification, then the managers spawn objects and assign policies to those objects if applicable. After the initialization stage, all objects will run automatically in the environment following their policies while the managers monitor the states of the object and kick off new objects or recycle terminated ones. At each step, information and states of the managed objects will be thrown to the outside of the environment for policies, such as RL policies, to determine new control signals.

By flexibly combining existing managers, new environments, as well as the officially released benchmarks, can be easily composed for more intriguing research topics. For example, the *Object Manager* in safe driving environments can be plugged into multi-agent environments with a simply `engine.register_manager()` and we get a new multi-agent environment for safe driving purposes. Furthermore, the *Reply Traffic Manager* in real data replay environments can be replaced by *IDM Traffic Manager* to generate reactive traffic on real maps instead of replaying logged traffic data. By calling `engine.update_manager()`, as shown in Fig. 7 **B.**, the *Agent Manager* in generalization environments can be updated to *Multi-agent Manager* to benchmark the generalizability of existing MARL algorithms. The combination of basic managers and their derived managers is thus called composition. As shown in Fig. 2, we provide some composition results.

4 Composing Diverse Driving Scenarios

In MetaDrive, a driving scenario can be composed by sequentially executing four basic Managers: (1) *Map Manager*, in which PG maps or real maps will be randomly sampled for each new episode, (2) *Traffic Manager*, which contains a set of traffic vehicles cruising in the scene and navigating to their given destinations, (3) *Object Manager* that randomly scatters obstacles in the map to create more challenging and safety-critical scenarios, (4) *Agent Manager* that manages target vehicles which are actuated by external policies, *e.g.* RL agents. Other managers for special purposes, such as *Tollgate Manager* and *Traffic Light Manager*, are also available and can work with the four basic managers to compose more scenes.

4.1 Generating Maps

MetaDrive supports two approaches to generate road networks: the procedural generation and importing from real data set.

Procedural Generation. One of the most important map sources is the Procedural Generation (PG). As shown in Fig. 3, ingredients used to do PG are a set of road blocks sampled from typical types. Each block preserves properties like lanes, spawn points for locating new vehicles, and *sockets* for interconnect other blocks.

We represent a road block using an undirected graph with additional features: $G_\omega = \{V, E, S, P, \omega, \Omega\}$, with nodes V denoting

Algorithm 1: Procedural Generation of Maps

Input: Maximum tries in one block `T`; Number of blocks in each map `n`; Number of required maps `N`

Result: A set of maps $M = \{G_{net}^{(i)}\}_{i=1,\dots,N}$

```
# Define the main function to generate a list of maps
Function main(T, n)
    Initialize an empty list M to store maps
    while M does not contain N maps do
        Initialize an empty road network G_net
        G_net, success=BIG(T, G_net, n)
        if success is True then
            Append G_net to M
    Return M
# Define the Block Incremental Generation (BIG) helper function that appends one block to current map if feasible and return current map with a success flag
Function BIG(T, G_net, n)
    if G_net has n blocks then
        Return G_net, True
    for 1, ..., T do
        Create new block G_ω = GetNewBlock()
        Find the sockets for new block and old blocks: e_1 ~ G_ω(S), e_2 ~ G_net(S)
        Rotate G_ω so that e_1, e_2 have supplementary heading
        if G_ω does not intersect with G_net then
            G_net ← G_net ⋃ G_ω
            G_net, success=BIG(T,G_net, n)
            if success is True then
                Return G_net, True
            else
                Remove the last block from G_net
    Return G_net, False
# Randomly create a block
Function GetNewBlock()
    Randomly choose a road block type T ~ {Staright, ...}
    Instantiate a block and randomize the parameters G_ω, ω ~ Ω_T
    Return G_ω
```

the *joints* in the road network and edges E denoting *lanes* which interconnect nodes. At least one node is assigned as the *socket* $S = \{e_{i_1}, e_{i_2}, ...\}, e \in E$. The socket is usually a short straight road at the end of lanes which serves as the anchor to connect to the socket of other blocks. Block can preserve several sockets. For instance, Roundabout and T-Intersection have 4 sockets and 3 sockets, respectively. Spawn points P can be sampled on lanes for allocation of traffic participants. Apart from the above properties, a block type-specific *parameter space* Ω is defined to bound the possible parameters ω of the block, such as the number of lanes, the lane width, and the road curvature, and so on. The road block is the elementary component that can be assembled into a complete road network $G_{\omega_{net}} = \{V, E, S, P, \omega_{net}\} = G_{\omega_1} \bigcup ... \bigcup G_{\omega_n}$ by the procedural generation algorithm. Shown in Fig. 3, the detail of typical block type T is summarized as follows:

- **Straight**: A straight road is configured by the number of lanes, length, width, and lane line types.
- **Ramp**: A ramp is a road with an entry or exit existing in the rightest lane. The acceleration lane and deceleration lane are attached to the main road to guide the traffic vehicles to the enrty or exit of the main road.
- **Fork**: A structure used to merge or split additional lanes from the main road and change the number of lanes.
- **Roundabout**: A circular junction with four exits (sockets) with a configurable shape. Both roundabout, ramp, and fork aim to provide diverse merge scenarios.
- **Curve**: A curve block consists of a circular shape or clothoid shape lanes with configurable curvature.
- **T-Intersection**: An intersection that can enter and exit in three ways and thus has three sockets. The turning radius is configurable.
- **Intersection**: A four-way intersection allows bi-directional traffic. It is designed to support the research of unprotected intersection.

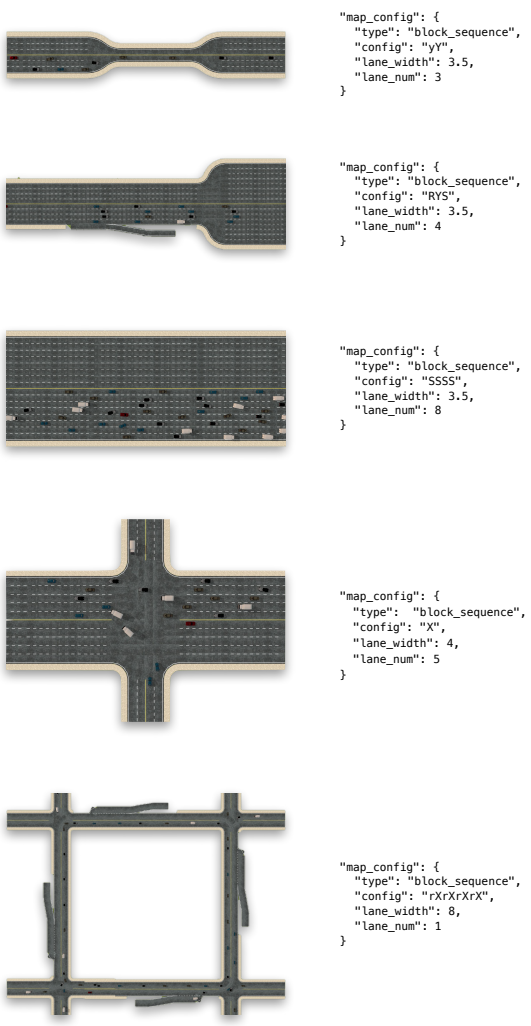

Fig. 4. MetaDrive can derive diverse scenarios with different config in the input config.

As illustrated in Algorithm 1, we propose a search-based PG algorithm *Block Incremental Generation (BIG)*, which recursively appends block to the existing road network if feasible and reverts the last block otherwise. When adding new block, BIG first uniformly chooses a roadblock type T and instantiate a block G_ω with random parameters $\omega \sim \Omega_T$ (the function ***GetNewBlock()***). After rotating the new block so the new block's socket can dock

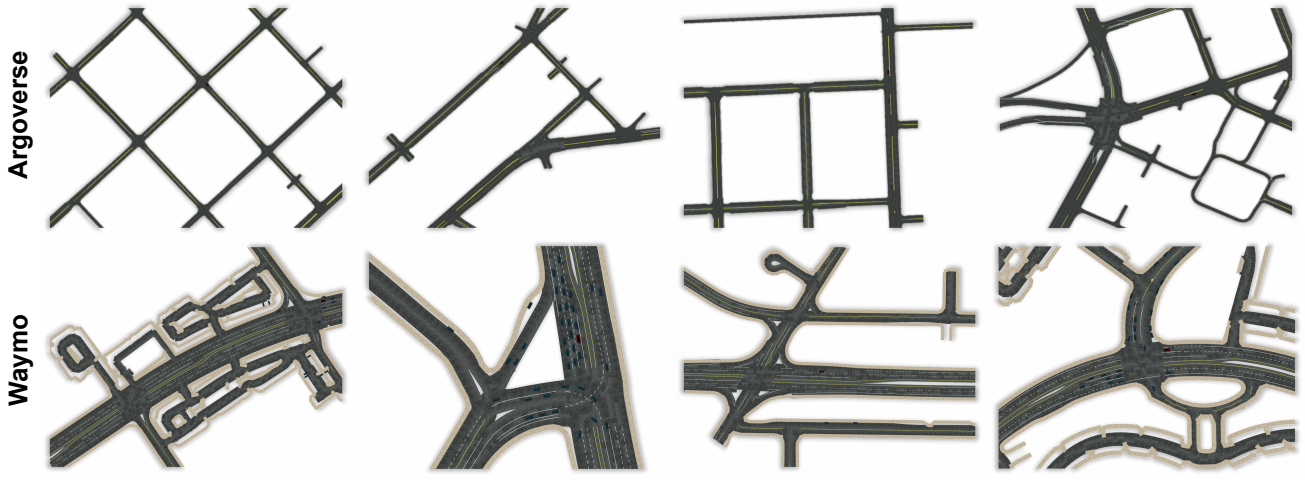

Fig. 5. MetaDrive can load real scenarios from Argoverse dataset [5] and Waymo dataset [51].

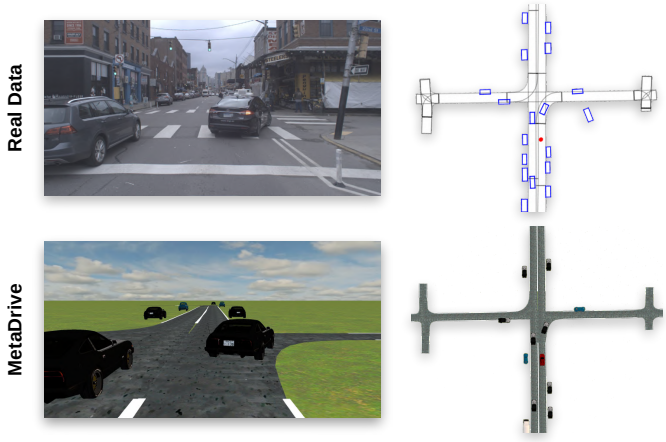

Fig. 6. A simulation scenario is replicated from the real traffic data of Argoverse dataset [5]. The traffic vehicles are actuated by Replay Traffic Manager.

into one socket of the existing network (Line 16, 17), BIG will then verify whether the new block intersects with existing blocks (Line 18). We test the crossover of all edges of G_ω and network G_{net}. If crossovers exist, then we discard G_ω and try a new one. Maximally `T` trials will be conducted. If all of them fail, we remove the latest block and revert to the previous road network (Line 24). The stop criterion of BIG is usually the number of blocks. In addition, users can also specify the block sequences to sample from the parameter spaces to generate maps with same block sequence but in different shapes. Fig. 3 provides demonstration of generated maps assembled by different number of blocks.

The map generation process can be controlled by overwriting the map config, allowing the customization of the road network. When creating the environment, the user can pass a `config`, namely a dictionary, into the environment `MetaDriveEnv(config)` and specify the road network by providing overrides to the default settings of the map. For instance, as shown in Fig. 4, the user can generate a variety of environments such as wide straight roads by modifying the dictionary with few lines of code. In the showcases, we first specify the method to generate maps. `config["map_config"]["type"] = "block_sequence"` indicates generating map containing a given sequence of block types. Alternatively, setting type to `block_num` requests the BIG algorithms to produce a map containing given number of blocks while the type of blocks should be randomized. Detailed annotations of different blocks is given in the documentation of the MetaDrive.

Real Map. Apart from the procedural generation of new scenarios, MetaDrive can also import real traffic data from autonomous driving datasets. The road network data usually consists of a set of lane line center points, such as in Argoverse dataset [5], Waymo dataset [26], [51] and OpenStreetMap [15]. Benefiting from the unified data structure to represent road networks, MetaDrive can seamlessly incorporate real-world data through creating lanes from waypoints in the dataset and then building functionalities at those lanes, such as the transformation between world coordinates and Frenet coordinates. MetaDrive currently supports Waymo motion dataset [26], [51] and Argoverse motion dataset [5], since they both contain the relationship between lanes, such as the entry lanes, existing lanes, left and right neighboring lanes. We provide the Fig. 5 to show the imported real maps from both datasets. It is worth noticing that the Waymo dataset includes more diverse road structures and more detailed information such as boundaries than the Argoverse dataset. On the other hand, the Waymo dataset includes more complete scenarios (approximately 20,000) than the Argoverse dataset (approximately 100). Thus we only conduct real data generalization experiments on Waymo data.

4.2 Generating Traffic Participants

Traffic Generation. MetaDrive maintains the traffic through *Traffic Manager*. The *Traffic Manager* decides when and where to generate or recycle traffic vehicles and also assigns policies to vehicles. To support the compositionality, The attributions of created traffic vehicles, such as vehicle type, powertrain parameters, behaviors (aggressive or conservative), spawn points, and destinations, can be customized or randomized.

For PG maps, *IDM Traffic Manager* is the default traffic manager which actuates traffic vehicles according to rule-based IDM policy, so they are responsive to the target vehicles. There are two modes to initialize the traffic flow in PG scenarios: Respawn mode and Trigger mode. Respawn mode is designed to maintain traffic flow density. In Respawn mode, *Traffic Manager* assigns

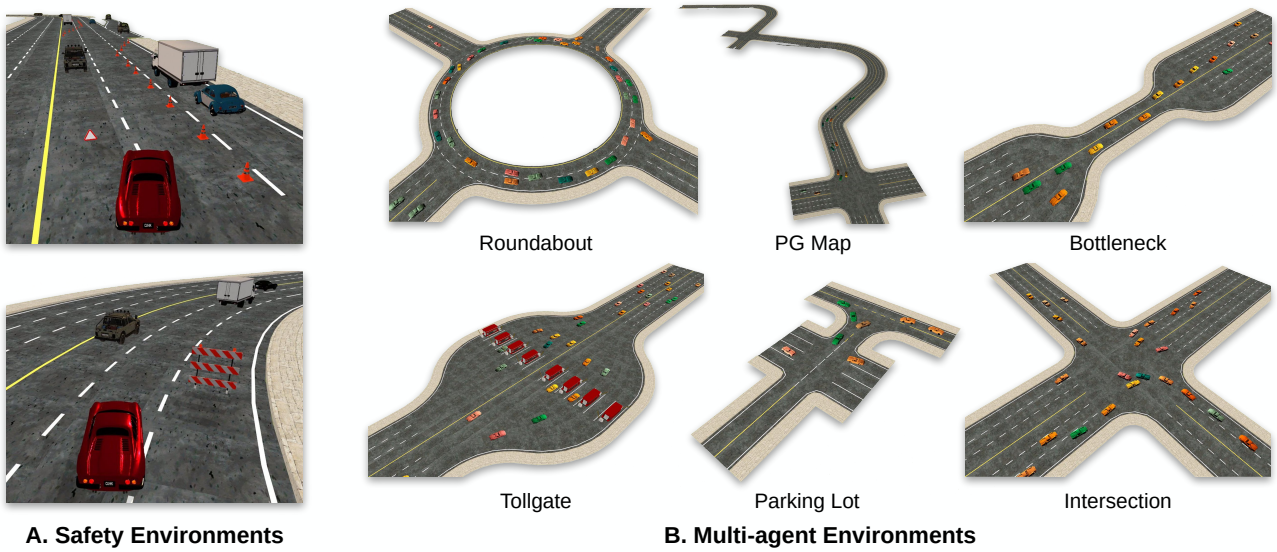

Fig. 7. **A.** Safe driving environments where obstacles such as broken down vehicles and traffic cones are randomly scattered on the map. **B.** Multi-agent environments where all agents need to coordinate their driving behaviors to achieve the population efficiency.

traffic vehicles to random spawn points on the map. The traffic vehicles immediately start driving toward their destinations after spawning. When a traffic vehicle terminates, it will be re-positioned to an available spawn point. On the contrary, the Trigger mode traffic flow is designed to maximize the interaction between target vehicles and traffic vehicles. The traffic vehicles stay still in the spawn points until the target agent enters the trigger zone. Taking the Intersection as an example. Traffic vehicles inside the intersection will be triggered and start moving only when the target vehicle trespasses into the intersection.

For real maps imported from Argoverse [5] and Waymo dataset [26], [51], *IDM Traffic Manager* or *Replay Traffic Manager* can be used. When steering vehicles by Replay Traffic Manager, in the log-reply mode, traffic vehicles strictly reproduce the trajectories collected in the real world without any reaction to the surroundings. As shown in Fig. 6, traffic vehicles with lop-replay policy replicate the behaviors recorded in Argoverse data set. When in the IDM control mode, the IDM controller actuates all the traffic vehicles whose initial states, *i.e.* position are synchronized with the first frame of log data. After that, they are reactive to the ego car and surroundings and make decisions at every step. This, in turn, yields new trajectories different from the recorded data.

Scattering Obstacles. To compose near-accidental scenarios, Object Manager scatters many obstacles such as cones, crash barriers as well as broken down vehicles on the road, as shown in Fig. 7 **A.** The density of the obstacles determines the difficulty of the task. A collision with the obstacle yields a cost for the ego vehicle, which can be used to train safe RL algorithms.

Target Vehicles Management. *Agent Manager* is designed to register and maintain controllable vehicles in both single-agent and multi-agent environments. It processes and packs information collected from the environment, such as the observations, rewards, and user-specified information with corresponding agent id, and then conveys them to external algorithms for training. Our special implementation of *Multi-agent Manager* makes it feasible to benchmark both common MARL setting and the mix-motive RL [17], [35], where the number of active agents is varying and new agents spawn immediately once old ones terminate.

4.3 Sensors and Performance

MetaDrive is implemented based on Panda3D [12] and Bullet Engine. The well-designed rendering system of Panda3D enables MetaDrive to construct realistic monitoring and observational data. Bullet Engine empowers accurate and efficient physics simulation. MetaDrive provides various kinds of sensory input, as illustrated in Fig.1 **C**. For low-level sensors, Lidar, RGB cameras, depth cameras implemented by depth shader can be placed anywhere in the scene with adjustable parameters such as noise distribution, view field, and the laser number. Meanwhile, MetaDrive can also provide high-level scene information as input to the learning policy, such as the road information like bending angle, length and direction, and nearby vehicles' information like velocity, heading, and profile. Note that MetaDrive aims at providing an efficient platform to benchmark RL research, particularly generalizable RL research, therefore we improve the simulation efficiency at the cost of a photorealistic rendering effect. As a result, MetaDrive can run at 300 FPS in the single-agent environment with 10 rule-based traffic vehicles and 60 FPS in the multi-agent environment with 40 RL agents.

5 Benchmarking Reinforcement Learning Tasks

Based on MetaDrive, we construct four driving tasks corresponding to different reinforcement learning problems. The first two are in a single-agent setting where the traffic is actuated by rule-based policies (for PG maps) or trajectory replay policies (for real maps), and a target vehicle is controlled by an external RL agent. The third task is in the safe driving setting where maps are generated by PG and obstacles are scattered randomly. The last task is in the multi-agent setting where a population of agents learns to simulate a traffic flow and each vehicle is actuated by a continuous control policy. We release the implementation of the baseline algorithms for reproducible research at https://github.com/metadriverse/metadrive.

5.1 Experimental Setting

In all tasks, the objective of RL agents is to steer the target vehicles with low-level continuous control actions, namely acceleration,

TABLE 2
Hyper-parameters of benchmarked methods.

SAC/SAC-Lag Parameters	Value
Discount Factor γ	0.99
τ for target network update	0.005
Learning Rate	0.0001
Environmental horizon T	1500
Steps before Learning start	10,000
Buffer Size	1,000,000
Prioritized Buffer	True
Train Batch Size	256
Initial Alpha	1.0
Penalty Learning Rate	0.01
Cost Limit for SAC-Lag	1

PPO/PPO-Lag Parameters	Value
Discount Factor γ	0.99
KL Coefficient	0.2
λ for GAE [44]	0.95
Number of SGD epochs	20
Train Batch Size	8000
SGD mini batch size	100
Learning Rate	0.00005
Clip Parameter ε	0.2
Penalty Learning Rate	0.01
CPO Target KL Divergence	0.01
Cost Limit for PPO-Lag/CPO	1

IPPO/CCPPO Parameters	Value
KL Coefficient	1.0
λ for GAE [44]	0.95
Discount Factor γ	0.99
Environmental steps per training batch	1024
Number of SGD epochs K_p	5
SGD mini batch size	512
Learning Rate	0.0003
Environmental horizon T	1000
Neighborhood radius d_n	40 m
Number of random seeds	8
Maximal environment steps	1M

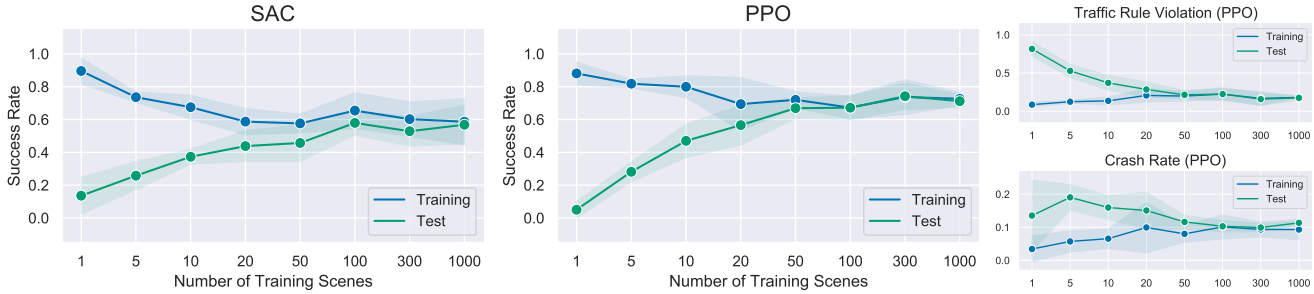

Fig. 8. The generalization result of the agents trained with off-policy RL algorithm Soft Actor-critic (SAC) [14] and on-policy RL algorithm PPO [45]. Increasing the number of training scenarios leads to a higher test success rate and lower traffic rule violation and crash probability, which indicates the agent's generalization is significantly improved. Compared to PPO, the SAC algorithm brings a more stable training performance. The shadow of the curves indicates the standard deviation.

brake, and steering. We attempt to unify all tasks with a general setting of observation, reward function, and evaluation metrics.

Observation. The observation of RL agents is as follows:

- A 240-dimensional (72-dimensional in MARL) vector denoting the 2D-Lidar-like point clouds with $50m$ maximum detecting distance centering at the target vehicle. Each entry is in $[0,1]$ representing the relative distance of the nearest obstacle in the specified direction.
- A vector summarizing the target vehicle's state such as the steering, heading, velocity, and relative distance to the left and right boundaries.
- The navigation information that guides the target vehicle toward the destination. We sparsely spread a set of checkpoints, $50m$ apart on average, in the route and use the relative positions toward future checkpoints as observation to the target vehicle.

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

$$R = c_1 R_{disp} + c_2 R_{speed} + R_{term}. \quad (1)$$

The *displacement reward* $R_{disp} = d_t - d_{t-1}$, wherein the d_t and d_{t-1} denote the longitudinal movement of the target vehicle in Frenet coordinates of current lane between two consecutive time steps, provides dense reward to encourage agent to move forward. The *speed reward* $R_{speed} = v_t / v_{max}$ incentivizes agent to drive fast. v_t and v_{max} denote the current velocity and the maximum velocity ($80\ km/h$), respectively. 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 crashing others or violating traffic rules such as locating at undrivable area. We set $c_1 = 1$ and $c_2 = 0.1$. Sophisticated reward engineering may provide a better supervision signal, which we leave for future work. For benchmarking Safe RL algorithms, collision to vehicles, obstacles, sidewalk and buildings raises a cost $+1$ at each time step.

Evaluation Metrics. We evaluate a given driving agent for multiple episodes and define the ratio of episodes where the agent arrives at the destination as the *success rate*. The definition is the same for *traffic rule violation rate* (namely driving out of the road) and the *crash rate* (crashing other vehicles). Compared to episodic reward, the success rate is a more suitable measurement when evaluating generalization, because we have a large number of scenes with different properties such as the road length and the traffic density, which makes the reward vary drastically across different scenes. We conduct experiments on MetaDrive with algorithms mostly implemented in RLLib [25]. Each trial consumes 2 CPUs with 8 parallel rollout workers. All experiments are repeated 5 times with different random seeds.

5.2 Generalization to Unseen PG Scenes

To benchmark the generalizability of a driving policy, we develop an RL environment that can generate an unlimited number of diverse driving scenarios through the aforementioned procedural generation algorithm. Traffic flow powered by IDM model in varying density is also added to interact with the target vehicle. We split the generated scenes into two sets: the training set and the test set. We train the RL agents only in the training set and evaluate them in the held-out test set. The generalizability of a trained agent is therefore measured by the test performance. The objective of this task is to show how the diversity and quantity of training scenarios

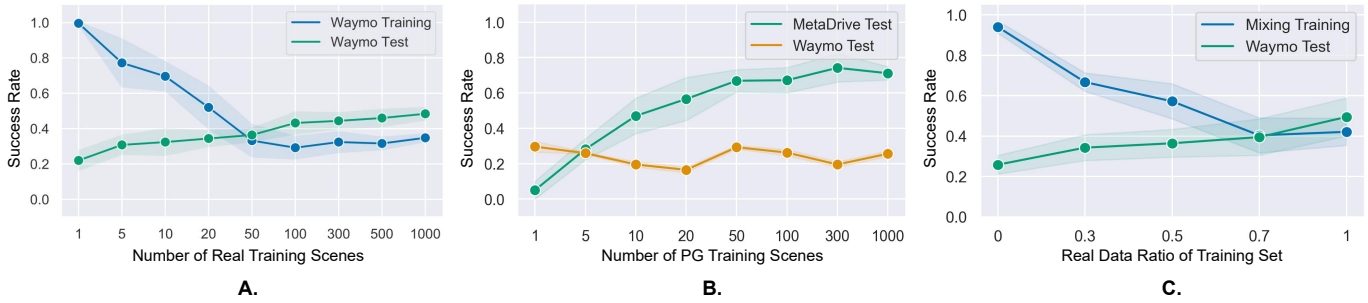

Fig. 9. **A.** Waymo generalization experiment with a changing number of scenarios contained in training set. **B.** The test performance of PG-map-trained policies in real-world scenarios. **C.** Result of the policies trained on 5 training sets consisting of different mixing ratio of real and synthetic data.

affect the generalizability of the learned policy. Table 2 describes detailed hyperparameter settings in this experiments.

We train the agents with two popular RL algorithms PPO [45] and SAC [14]. As shown in Fig. 8, the result of improved generalization capacity is observed in the agents trained from both RL algorithms: First, the overfitting happens if the agent is not trained with a sufficiently large training set. When $N = 1$, namely the agent is trained in a single map, we can see the significant performance gap of the learned policies between the training set and test set. Second, the generalization ability can be greatly strengthened if the agents are trained in more environments. As the number of training scenes N increases, the final test success rate keeps increasing while rule violation and crash decrease drastically. The overfitting is alleviated and the test performance can match the training performance when N is higher. The results clearly show that increasing the diversity of training environments can significantly improve the generalizability of RL agents. It also highlights the strength of the compositional simulator for generalizable reinforcement learning research.

5.3 Generalization to Unseen Real Scenarios

MetaDive has the capacity to import real scenarios from autonomous driving dataset. Currently, we build real-world cases collected by Waymo [26] and split the training set and test set so that they consist of 1000 and 100 non-overlapping cases, respectively.

We conduct three experiments to discover the generalization phenomenon in real cases. Firstly, by changing the number of cases contained in real training sets, we train 9 sets of agents to benchmark the generalizability of SAC [14]. As shown in Fig. 9 **A.**, increasing the training set size can also improve the generalizability of RL agents in real scenarios. With the number of cases exceeding 100, the test performance is slightly higher than the training performance. This might because we build the test set with meticulous selection by human, while keeping the training scenarios draw from raw data. Therefore The noise and invalid cases in the training set undermine the training performance.

In the second experiment, we evaluate the trained PPO agents from PG generalization experiment (Sec. 5.2) on the real-world test set. The orange line in Fig. 9 **B.** shows the test performance of those agents in Waymo dataset. We find that increasing the diversity in PG training set can not improve the test performance on real-world test set. This suggests the sim-to-real gap can not be eliminated by increasing training data from heterogeneous distribution.

The third experiment verifies whether introducing real cases in PG training set can improve the test performance in real test set. We construct 5 training sets consisting of 100 training cases where real scenes and PG scenes are mixed in different ratios (0%, 30%, 50%, 70%, 100%). As shown in Fig. 9 **C.**, increasing the real data ratio improves the test success on real test set. The decreasing training success rate implies that real scenarios is harder than PG scenes so that it is difficult for SAC to find a good solution.

These three experiments show that for sim-to-real RL applications it is critical to ensure that data distributions in the simulator and real-world are similar. Otherwise, the generalizability shown in the simulation can not be well transferred to reality. By building training environments from real datasets, MetaDrive improves the generalization of trained policies in real-world scenarios and thus provides the flexibility to conduct further research.

5.4 Safe Exploration

As driving itself is a safety-critical application, it is important to evaluate the safe RL methods under the domain of autonomous driving. We define a new suite of environments to benchmark the *safe exploration* in RL. As shown in Fig. 7 **A.**, we randomly place obstacles on the road. Different from the generalization task, we do not terminate the agent if a collision with obstacle or other vehicle happens. Instead, we allow the learning agent to continue driving and record the crash with a cost $+1$. Thus as a safe exploration task, the learning agent is required to balance the reward and the cost. We also evaluate the agents on unseen maps to show their generalization ability in the aspect of safety.

We evaluate the reward shaping variants (RS) and Lagrangian variants (Lag) of PPO [45] and SAC [14] as well as the Constrained Policy Optimization (CPO) [1]. RS method considers negative cost as an auxiliary reward while Lagrangian method [39] consider the learning objective as:

$$\max_{\theta} \min_{\lambda \geq 0} = \underset{\tau}{E}[R_\theta(\tau) - \lambda(C_\theta(\tau) - d)], \tag{2}$$

wherein $R_\theta(\tau)$ and $C_\theta(\tau)$ are the episodic reward and the episodic cost respectively, θ is the policy parameters and d is a given cost threshold. Thus the objective is to maximize episodic accumulative reward while restricting the episodic accumulative cost under a predefined threshold.

Different from existing work which applies Lagrangian to SAC directly [13], we additionally equip SAC-Lag with a PID controller to update the multiplier to alleviate the oscillation in the training [49]. All Safe RL algorithms are trained using 500,000 steps. The algorithm configuration is in Table 2. We also provide two offline learning baselines, BC and CQL [21], which use 36k human demonstrated transitions in 97% success

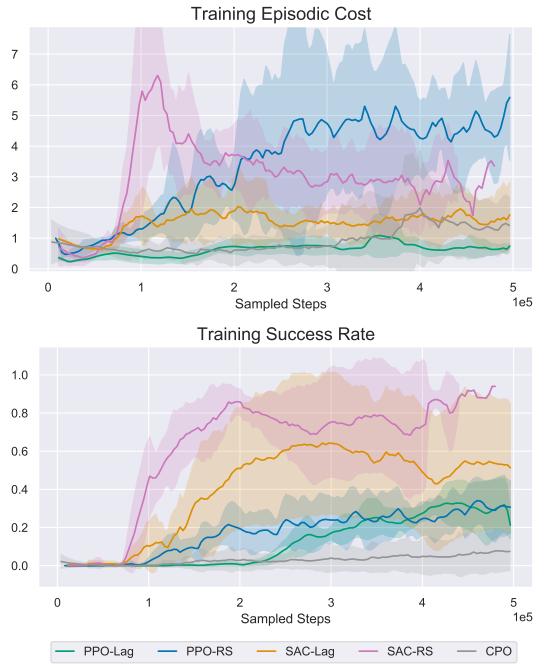

Fig. 10. Since the training time safety is critical to Safe RL, we show the learning progress of different Safe RL methods. Though achieves superior sample efficiency, the reward shaping version of SAC induces a peak in the training cost, while the Lagrangian SAC improves the policy while satisfying the safety constraint.

TABLE 3
The test performance of different approaches in Safe Exploration.

Category	Method	Cumulative Reward	Cumulative Cost	Success Rate
RL	SAC-RS [14]	**327.13** ±7.28	3.38 ±0.60	**0.801** ±0.040
	PPO-RS [45]	197.27 ±16.24	3.33 ±0.68	0.207 ±0.052
Safe RL	SAC-Lag [49]	324.23 ±14.45	1.90 ±0.44	0.714 ±0.103
	PPO-Lag [39]	269.51 ±22.54	1.82 ±0.33	0.477 ±0.114
	CPO [1]	194.06 ±108.86	**1.71** ±1.02	0.210 ±0.290
Offline Methods	BC	101.63 ±16.06	1.00 ±0.45	0.01 ±0.03
	CQL [21]	156.4 ±31.94	6.82 ±5.1	0.11 ±0.07

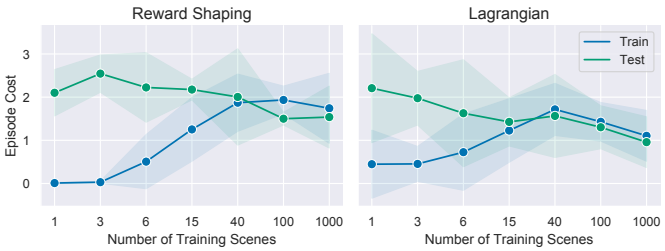

Fig. 11. The episode cost of the trained policies in safety generalization experiment. We observe overfitting and poor safety performance in test time if trained with few training scenes.

rate to do offline learning and are totally safe in training time. To promote imitation learning (IL) and Offline RL research, the driving trajectories collected by humans are also released at https://github.com/metadriverse/metadrive/releases/download/MetaDrive-0.2.3/human_traj_100_new.json.

As shown in Table 3 and Fig. 10, SAC-RS shows superior performance but causes high safety violations. On the contrary, the Lagrangian SAC can achieve lower cumulative costs while giving up little reward. Meanwhile, PPO is inferior compared to SAC, since it is an on-policy algorithm that are less sample-efficient than off-policy methods. Fig. 10 presents the learning dynamics of different Safe RL algorithms. The result suggests that SAC-RS shows the best sample efficiency, but a peak of episodic cost happens when it learns most efficiently. Noticeably, for BC and CQL, though interaction with environments is not required and policy can be learned with zero cost, they all have worse performance than online learning RL methods. CQL still outperforms BC, indicating that Offline RL method can learn a better policy given limited data.

We also conduct the *safety generalization* experiment to verify the impact of training sets to testing-time safety performance. We train the PPO-RS and PPO-Lag on training sets with different size and show the training and test episode cost in Fig. 11. When training with few environments, both methods achieve high test costs even if the training cost is low. The safety generalizability can be improved by increasing the training set diversity. This experimental result implies the overfitting in safe exploration, which is a critical issue if we want to apply the end-to-end driving system to the real world.

5.5 Mixed Motive Multi-agent Reinforcement Learning

We design five multi-agent RL environments in MetaDrive to benchmark the MARL algorithms. Depending on the environments, 10 to 40 agents are simultaneously running in the environment. It is difficult to experiment with such dense multi-agent traffic in previous simulators due to poor efficiency. In MetaDrive, we manage to achieve the efficiency of 60 FPS with 40 agents running simultaneously in a shared environment. Fig. 7 **B.** shows six environments for MARL, and their details are introduced below:

- **Roundabout**: A four-way roundabout with two lanes. 40 vehicles spawn during environment reset. This environment includes merge and split junctions.
- **Intersection**: An unprotected four-way intersection allowing bi-directional traffic as well as U-turns. Negotiation and social behaviors are expected to solve this environment. We initialize 30 vehicles.
- **Tollgate**: Tollgate scene includes narrow roads to spawn agents and ample space in the middle with multiple tollgates. The tollgates become static obstacles where crashing is prohibited. We request agents to stop within tollgate for 3s. The agent will fail if they exit the tollgate before being allowed to pass. 40 vehicles are initialized. Complex behaviors such as deceleration and queuing are expected. Additional states such as whether the vehicle is in tollgate and whether the tollgate is blocked are given.
- **Bottleneck**: Complementary to Tollgate, Bottleneck contains a narrow bottleneck lane in the middle that forces the vehicles to yield to others. We initialize 20 vehicles in this scene.
- **Parking Lot**: A compact environment with 8 parking slots. Spawn points are scattered in both parking lots and on external roads. 10 vehicles spawn initially and need to navigate toward external roads or enter parking lots. In this environment, we allow agents to reverse their cars to spare space for others. Good maneuvering and yielding are the keys to solving this task.
- **PG Map**: All vehicles in PG generated three roadblocks maps are controlled by RL agents.

These multi-agent environments pose a new challenge under the

TABLE 4
Success rate (%) of different approaches in Multi-agent RL benchmarks.

Method	Bottle.	Tollgate	Inter.	Round.	Parking	PG Map
IPO [43]	74.18 ± 15.87	74.72 ± 15.82	73.93 ± 9.18	64.55 ± 5.17	20.90 ± 5.70	83.82 ± 4.40
MF-CCPPO [58]	65.27 ± 17.06	48.62 ± 32.84	71.58 ± 7.79	68.95 ± 4.78	14.42 ± 4.92	79.63 ± 4.71
Concat-CCPPO	51.93 ± 17.03	31.92 ± 26.01	62.67 ± 12.46	64.50 ± 8.46	4.73 ± 3.83	75.38 ± 8.69
CL [32]	66.45 ± 10.50	52.99 ± 22.86	67.90 ± 15.00	81.51 ± 4.49	16.17 ± 8.94	77.96 ± 6.95
CoPO [35]	73.39 ± 16.65	79.66 ± 13.92	77.79 ± 2.95	73.65 ± 4.61	20.98 ± 3.55	80.24 ± 4.21

setting of mixed-motive RL, because each constituent agent in this traffic system is self-interested and the relationship between agents is changing.

We benchmark several MARL algorithms: Independent policy optimization (IPPO) method [43] which uses PPO [45] as the individual learners and two centralized critic methods which encode the nearby agents' states into the input of value functions (centralized critics). We test two variants of centralized critic PPO (CCPPO): The first one is the Mean Field (MF) CCPPO, which averages the states of nearby vehicles within 10 meters and feeds the mean states to the value network [58]. The second variant concatenates the state of K nearest vehicles (K=4 in our experiment) as a long vector feeding as extra information to the value network [33]. Recently proposed CoPO [35] is also compared, which is a mixed-motive MARL method addressing the coordination problem in multi-agent systems. The hyperparameters of used methods can be found in Table 2. Table 4 shows the MARL experiment results. Note that the result is based on MetaDrive 0.2.5 and is different to previous paper [35] due to updates in MetaDrive. We find that the concatenated state of K nearest vehicles hurts the performance of CCPPO and leads to worse performance compared to MF-CCPPO. We hypothesize this is because in driving tasks the neighborhood of ego vehicles is varying all the time while concatenating states greatly expands the input dimension thus creating difficulty to the learning. The CoPO method [35] achieves good performance in various environments, due to the local and global coordination.

5.6 Discussion on the Learning-based Driving Policies

When developing MetaDrive, we ran numerous experiments to search proper driving policies, and found that the performance of learning-based methods greatly depends on the observation. RL methods learn faster using the state vector observation in Sec. 5.1 than the first-person view RGB images or top-down semantic map. Also, we find the displacement reward is the most important part of the reward function. We hope these environmental setting can provide insights on designing end-to-end driving policy.

We have implemented and benchmarked many learning algorithms for machine autonomy, including Reinforcement Learning, Imitation Learning (IL), Offline RL methods. The experimental results show that 1) Value-based methods (SAC, TD3) is more suitable for driving tasks than policy-based method (PPO, TRPO) due to their high sample efficiency and superior success rate (Fig. 8, Table 3); 2) For learning safety, it is important to develop new paradigm incorporating safety constraint into the policy instead of manually shaping reward and cost function. Our recent works on human-in-the-loop training [24], [36] provide examples; 3)When learning from limited offline data, Offline RL outperforms IL by a large margin in the safe exploration experiment 3. Also, in our previous work [36], we train CQL [21] and BC with a large dataset containing 300,000 transitions collected by a well-trained PPO expert policy. The results show that the CQL can achieve 72% success rate, while BC only reaches 13%, suggesting that Offline RL is an appealing direction for future works.

6 Ethical Statement and Limitations

Ethical Statement. MetaDrive provides a driving simulation environment for RL research. Though MetaDrive itself has marginal negative societal impact, there might be two possible cases that cause damages. First, if a user applies the trained agent from MetaDrive to a real vehicle, the vehicle may cause accidents due to domain gap or uncertainty in neural network/action distribution. There is still a long way to go to narrow the sim2real gap. Second, since MetaDrive allows human subject to control vehicle via keyboard, joystick, or steering wheel, the player may experience, though the chance is low, discomfort or pain due to exposure to the fast driving in the MetaDrive 3D visualization.

Limitations. MetaDrive has limitations in the following aspects: First, the image rendering is not realistic as other driving simulators with photorealistic rendering effects like CARLA [10]. This is because MetaDrive specifically focuses on the generalizable RL research and thus aims to flexibly compose driving scenarios. Second, the generated driving scenes from MetaDrive don't contain traffic participants such as pedestrians and bicyclists. Third, a systematic driving scenario description protocol is left for future work to enable large-scale generation of corner cases such as near-accidental scenes for prototyping safe autonomous driving systems [11].

7 Conclusion

We develop a compositional and extensible driving simulator MetaDrive to facilitate the research of generalizable reinforcement learning. MetaDrive holds the core feature of compositionality, where an infinite number of diverse driving scenarios can be composed through both the procedural generation and the real traffic data replay. We construct a variety of RL tasks and baselines in both single-agent and multi-agent settings, including benchmarking generalizability across unseen scenes, safe exploration, and simulating multi-agent traffic.

Licenses. MetaDrive is released under Apache License 2.0. Panda3D, used in MetaDrive as a rendering system for visualization, is under the Modified BSD license, which is a free software license with very few restrictions on usage. Bullet engine is under Zlib license. The vehicle models are collected from Sketchfab under CC BY 4.0 or CC BY-NC 4.0. The skybox images are under CC0 1.0 Universal (CC0 1.0). To support real data import, MetaDrive is made using the Waymo Open Dataset [26], [51], provided by Waymo LLC under license terms available at https://waymo.com/open. Also, Argoverse is provided free of charge under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public license. Argoverse code and APIs are provided under the MIT license.

Acknowledgement: This project was partially supported by the Centre for Perceptual and Interactive Intelligence (CPII) Ltd under the Innovation and Technology Fund.

REFERENCES

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.

[2] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[4] Panpan Cai, Yiyuan Lee, Yuanfu Luo, and David Hsu. Summit: A simulator for urban driving in massive mixed traffic. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4023–4029. IEEE, 2020.

[5] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[6] Maxime Chevalier-Boisvert, Florian Golemo, Yanjun Cao, Bhairav Mehta, and Liam Paull. Duckietown environments for openai gym. https://github.com/duckietown/gym-duckietown, 2018.

[7] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

[8] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[9] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[10] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[11] Bernd Gassmann, Fabian Oboril, Cornelius Buerkle, Shuang Liu, Shoumeng Yan, Maria Soledad Elli, Ignacio Alvarez, Naveen Aerrabotu, Suhel Jaber, Peter van Beek, Darshan Iyer, and Jack Weast. Towards standardization of av safety: C++ library for responsibility sensitive safety. In *2019 IEEE Intelligent, Vehicles Symposium (IV)*, 2019.

[12] Mike Goslin and Mark R Mine. The panda3d graphics engine. *Computer*, 37(10):112–114, 2004.

[13] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort, 2020.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[15] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive computing*, 7(4):12–18, 2008.

[16] Junning Huang, Sirui Xie, Jiankai Sun, Qiurui Ma, Chunxiao Liu, Jianping Shi, Dahua Lin, and Bolei Zhou. Learning a decision module by imitating driver's control behaviors, 2019.

[17] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Çaglar Gülçehre, Pedro A Ortega, DJ Strouse, Joel Z Leibo, and Nando de Freitas. Intrinsic social motivation via causal influence in multi-agent rl. corr abs/1810.08647 (2018). *arXiv preprint arXiv:1810.08647*, 2018.

[18] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[19] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[20] Parth Kothari, Christian Perone, Luca Bergamini, Alexandre Alahi, and Peter Ondruska. Drivergym: Democratising reinforcement learning for autonomous driving. *arXiv preprint arXiv:2111.06889*, 2021.

[21] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

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

[23] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, Andrey Kurenkov, Karen Liu, Hyowon Gweon, Jiajun Wu, Li Fei-Fei, and Silvio Savarese. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks, 2021.

[24] Quanyi Li, Zhenghao Peng, and Bolei Zhou. Efficient learning of safe driving policy via human-ai copilot optimization. *International Conference on Representation Learning*, 2022.

[25] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062, 2018.

[26] Waymo LLC. Waymo open dataset: An autonomous driving dataset, 2019.

[27] 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.

[28] Mark Martinez, Chawin Sitawarin, Kevin Finch, Lennart Meincke, Alex Yablonski, and Alain Kornhauser. Beyond grand theft auto v for training, testing and enhancing deep learning in self driving cars. *arXiv preprint arXiv:1712.01397*, 2017.

[29] Karl Mason and Santiago Grijalva. A review of reinforcement learning for autonomous building energy management. *Computers & Electrical Engineering*, 78:300–312, 2019.

[30] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[31] Matthias Müller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. Sim4cv: A photo-realistic simulator for computer vision applications. *International Journal of Computer Vision*, 126(9):902–919, 2018.

[32] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*, 2020.

[33] Avik Pal, Jonah Philion, Yuan-Hong Liao, and Sanja Fidler. Emergent road rules in multi-agent driving environments. *arXiv preprint arXiv:2011.10753*, 2020.

[34] Praveen Palanisamy. Multi-agent connected autonomous driving using deep reinforcement learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[35] Zhenghao Peng, Ka Ming Hui, Chunxiao Liu, Bolei Zhou, et al. Learning to simulate self-driven particles system with coordinated policy optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

[36] Zhenghao Peng, Quanyi Li, Chunxiao Liu, and Bolei Zhou. Safe driving via expert guided policy optimization. In *5th Annual Conference on Robot Learning*, 2021.

[37] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE intelligent vehicles symposium (IV)*, pages 812–818. IEEE, 2017.

[38] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[39] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *openai*, 2019.

[40] Florian Richter, Ryan K Orosco, and Michael C Yip. Open-sourced reinforcement learning environments for surgical robotics. *arXiv preprint arXiv:1903.02090*, 2019.

[41] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[42] Anirban Santara, Sohan Rudra, Sree Aditya Buridi, Meha Kaushik, Abhishek Naik, Bharat Kaul, and Balaraman Ravindran. Madras: Multi agent driving simulator. *Journal of Artificial Intelligence Research*, 70:1517–1555, 2021.

[43] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv e-prints*, pages arXiv–2011, 2020.

[44] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[46] Radu Serban, Michael Taylor, Dan Negrut, and Alessandro Tasora. Chrono:: Vehicle: template-based ground vehicle modelling and simulation. *International Journal of Vehicle Performance*, 5(1):18–39, 2019.

[47] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[48] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[49] Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, pages 9133–9143. PMLR, 2020.

[50] Jiankai Sun, Hao Sun, Tian Han, and Bolei Zhou. Neuro-symbolic program search for autonomous driving decision module design. In *Conference on Robot Learning*, pages 21–30. PMLR, 2021.

[51] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020.

[52] Deepdrive Team. Deepdrive: a simulator that allows anyone with a pc to push the state-of-the-art in self-driving. https://github.com/deepdrive/deepdrive.

[53] Udacity Team. Udacity's self-driving car simulator: A self-driving car simulator built with unity. https://github.com/udacity/self-driving-car-sim.

[54] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[55] Eugene Vinitsky, Aboudy Kreidieh, Luc Le Flem, Nishant Kheterpal, Kathy Jang, Cathy Wu, Fangyu Wu, Richard Liaw, Eric Liang, and Alexandre M Bayen. Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Conference on Robot Learning*, pages 399–409, 2018.

[56] Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M Bayen. Flow: Architecture and benchmarking for reinforcement learning in traffic control. *arXiv preprint arXiv:1710.05465*, page 10, 2017.

[57] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4(6):2, 2000.

[58] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5571–5580. PMLR, 10–15 Jul 2018.

[59] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[60] Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The World Wide Web Conference*, WWW '19, page 3620–3624, New York, NY, USA, 2019. Association for Computing Machinery.

[61] Ming Zhou, Jun Luo, Julian Villella, 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 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 bou Ammar, Hongbo Zhang, Wulong Liu, Jianye Hao, and Jun Wang. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving, 2020.

Quanyi Li is a research assistant at Centre for Perceptual and Interactive Intelligence (CPII), the Chinese University of Hong Kong (CUHK). He received the B.Eng. degree in communication engineering from Beijing University of Posts and Telecommunications. He is interested in embodied AI and its generalizability, safety, interactivity and interpretability.

Zhenghao Peng is a Ph.D. Candidate in Department of Computer Science at University of California, Los Angeles (UCLA). He received M.Phil. degree in Department of Information Engineering at the Chinese University of Hong Kong (CUHK) and B.Eng. at Shanghai Jiao Tong university. His research interests are multi-agent reinforcement learning and human-AI interaction. This work is done while he was at CUHK.

Lan Feng is a master student majoring in robotics, system, and control at Eidgenössische Technische Hochschule Zürich (ETH Zurich). He received B.Eng. degree in Wuhan University (WHU). His research focuses on AI's generalizability and safety.

Qihang Zhang is a Ph.D. Candidate in Department of Information Engineering at the Chinese University of Hong Kong (CUHK). He received the B.Eng. degree in computer science and technology from Zhejiang University (ZJU) in 2021. His research interest lies in embodied intelligence and machine autonomy and learning.

Zhenghai Xue is a PhD student in the School of Computer Science and Engineering, Nanyang Technological University (NTU) Singapore. He received a B.S. in the School of Artificial Intelligence, Nanjing University (NJU) in 2022. He worked as a research assistant in the Department of Information Engineering at the Chinese University of Hong Kong (CUHK) in 2021. His research interest is reinforcement learning.

Bolei Zhou is an Assistant Professor in the Computer Science Department at the University of California, Los Angeles (UCLA). He earned his Ph.D. from MIT in 2018. His research interest lies at the intersection of computer vision and machine autonomy, focusing on enabling interpretable human-AI interaction. He and his colleagues have developed a number of widely used interpretation methods such as CAM and Network Dissection, as well as computer vision benchmarks Places and ADE20K. He is an associate editor for Pattern Recognition and has been area chair for CVPR, ICCV, ECCV, and AAAI. He received MIT Tech Review's Innovators under 35 in Asia-Pacific Award.