SUPPLEMENTARY MATERIAL

A DETAILS OF THE SCENARIO SEED CORPUS.

A.1 Classification rules for scenario road types.

In the previous sections, we mentioned CLASSIFIED_ROAD_TYPE(), which uses the set of waypoints n included in the scenario seed and judges its road type r by querying the connectivity of these waypoints in the topological map G and the distance between the road points. The classification rules are shown in Table 1. Classifying based on connectivity is easy to understand. As shown in Figure 4, different road types have differences in the connectivity between waypoints. This difference mainly manifests in the highest connection nodes in all the waypoints in n, which can be used to classify road types effectively. There are also cases where the number of connectivity is the same, such as X intersection and cross intersection, T intersection and Y intersection. This can be judged based on the maximum distance among nodes in the highest adjacent node situation. Because the distribution of waypoints in X and Y intersections is more compact compared to cross and T intersections.

Table 1: Classification rules for scenario road types.

Road Type	Max Connection at Node	Max Distance between Nodes
Intersection of 5	5	< 50
Crossroad	4	> 10
X Intersection	4	≤ 10
T Intersection	3	> 10
Y Intersection	3	≤ 10
Straight Road	≤ 2	N/A

A.2 File Structure and Semantic Situation of the Scenario Seed Corpus

We establish a scenario seed corpus for all scenario seeds under each map and store it in a json file named "Town**.json". The file structure is shown as follows, where each seed will have a corresponding ID. "wp" stores the set of waypoints, recording all position coordinates where an object can reasonably be placed. 'wp_mark' records the nearby traffic sign situation. 'wp_lanetype' records whether the road corresponding to each waypoint allows straight-ahead, left turns, and right turns. 'tf_loc' records the position information of traffic lights in the scene. 'center_loc' records the information of the center position of the scene. 'plan_way' records the starting and ending coordinate information of all feasible paths and the direction of the paths.

```
Town**.json
{"0":
   {"type": "t_intersection",
       "wp": [[101.389, 55.498, 0.0],...],
       "wp_mark": [["Signal_3Light_Post01"],...],
       "wp_lanetype": [0, 0, 0, 0, 0],
       "tf_loc": [[94.989, 70.409, 0.103], ...],
       "center_loc": [93.011, 30.333, 0.103],
       "plan_way": [["Left", 16.022, [[101.389, 55.498, 0.0], [88.377,
             46.150]]], ...]
   },
   ...}
```

The semantic information contained in our scenario seeds is shown in Figure 1. The red dots represent the possible positions for placement and the connectivity between various positions. The right figure represents the corresponding feasible paths. The seeds obtained based on map crawling technology can basically cover all paths that conform to traffic and real situations.

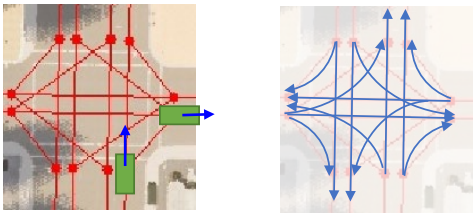
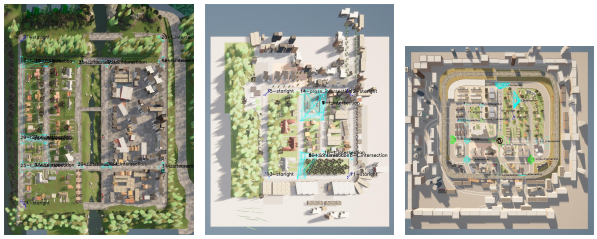


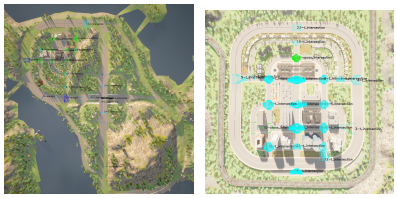
Figure 1: Semantic diagram of scenario seed.

A.3 Distribution and Quantity of Scenario Seeds

The position distribution of scenario seeds on each map is shown in Figure 2, and the statistics of seed numbers by road type are shown in Table 2.



(a) Town01 **(b) Town02** **(c) Town03**



(d) Town04 **(e) Town05**

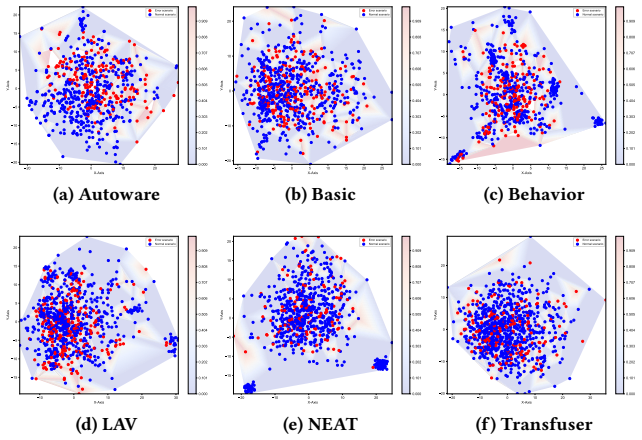
Figure 2: Position distribution of scenario seeds on each map.

Table 2: Distribution of scenario seed road types on each map.

Town	Road Type	Count
Town01	T Intersection	27
	Straight Road	2
Town02	Crossroad	1
	T Intersection	11
	Straight Road	4
Town03	Intersection of 5	1
	Crossroad	4
	T Intersection	11
	Straight Road	3
Town04	Crossroad	9
	T Intersection	11
	Straight Road	8
Town05	T Intersection	21
	Crossroad	2

B FEATURE DISTRIBUTION OF HISTORICAL SCENARIOS

In the main text, we proposed a method of transforming scenario data into graph representations (as detailed in §3.5). Therefore, we converted the scenario data generated by various systems throughout history into graph data. This data was then input into the scenario evaluation model to extract representations from the feature layer. We used the PCA method to reduce these features to two dimensions to facilitate graphical visualization, as shown in Figure 3. In the figure, blue points represent normal scenarios, and red points represent critical scenarios. The background heat map represents the confidence level of the scenario evaluation model for the data in that area. Although the distribution of scenario data is not highly separable, it shows a certain regional distribution trend. Meanwhile, the scenario evaluation model also attempts to learn and fit the distribution of critical scenarios during this process. Judging from the distribution of the heat map, the model has achieved good fitting results on the scenario data of various systems.



(a) Autoware (b) Basic (c) Behavior

(d) LAV (e) NEAT (f) Transfuser

Figure 3: Schematic diagram of the feature distribution of historical test scenarios under various systems.

C DETAILED DESCRIPTION OF BUG DISCOVERY

C.1 Autoware

Given that Autoware is based on the ROS operating system, we have identified the following bugs by generating Rosbag data through the monitoring and reviewing of test processes using the Rviz tool upon the discovery of related critical scenarios.

Bug1: GPS signal loss under the bridge. As shown in Figure 4, the vehicle starts to deviate to the left and right after entering the tunnel, ultimately leading to a collision. We noticed that there is a discrepancy and lag between the GNSS signal (indicated by the red arrow) and the actual position of the vehicle (indicated by the yellow arrow), suggesting the GPS signal was affected underneath the bridge.

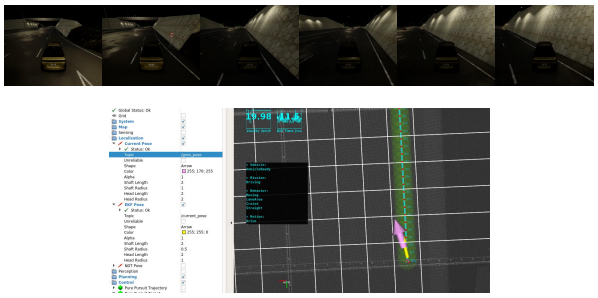


Figure 4: Illustration of Autoware Bug 1.

Bug2: Lidar failed to detect a child close by, resulting in a collision. As depicted in Figure 5, the vehicle encounters multiple pedestrians in this scenario. The lidar is able to detect adults and classify this part of the point cloud data, as shown on the left side. However, when a child appears in front, the lidar, affected by the child's height, fails to scan the child, as shown on the right side where there is an absence of lidar points. This ultimately leads to a collision with the child.

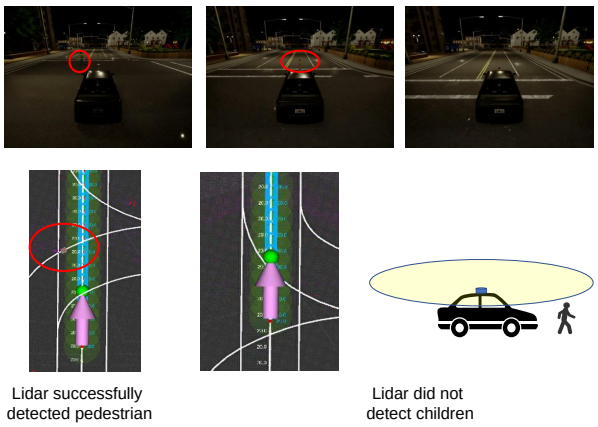


Figure 5: Illustration of Autoware Bug 2.

Bug3: Lidar missed signals from vehicles close by, causing a scrape during a turn. As seen in Figure 6, at a T intersection, the ego car needs to make a left turn, but a parked vehicle at the turning point requires the car to control the corner well. However, during this process, after the ego car approaches the vehicle, the lidar has a blind spot for scanning vehicles that are very close, leading to missed scans of the vehicle. This results in the vehicle continuing to move and causing a scrape.

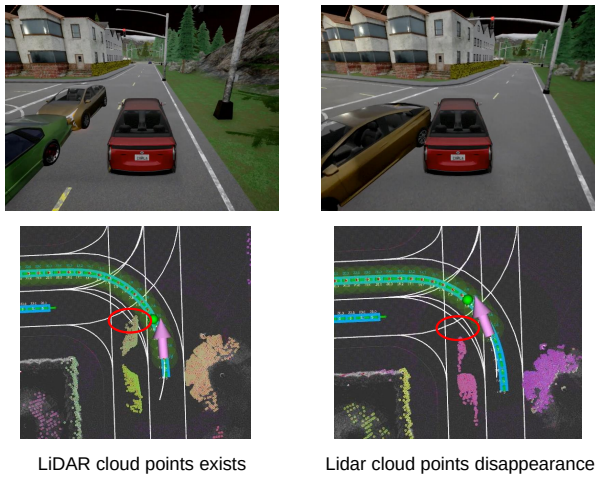


Figure 6: Illustration of Autoware Bug 3.

Bug4: Lidar failed to detect a pedestrian lying on the ground, causing a crush collision. As shown in Figure 7, there is a pedestrian lying on the road (the pedestrian was knocked down by another vehicle in the scenario, which is due to our work of mutating the path of objects in the scenario, and interactions also exist between them, forming the level3 construction of the scenario model as previously mentioned). The scanning area of the lidar has height and angle limitations, and in this case, the pedestrian lying on the ground was not scanned, resulting in a roll-over of the pedestrian.

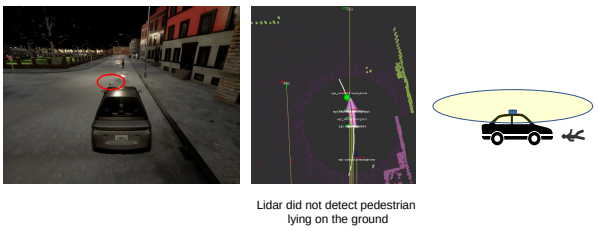


Figure 7: Illustration of Autoware Bug 4.

Bug5-7: Failed to recognize traffic infrastructures. As shown in Figure 8, Autoware's object detection module has a situation where it does not detect or recognize (identified as Unknown) all the traffic facilities (such as traffic lights, speed limit signs, and no entry signs) existing in the scenario. This also results in the most common mistake made by Autoware in operation, which is running a red light. This may be due to the realism difference between simulation and the real world. Autoware's object detection model has not been fine-tuned and retrained specifically for this series of elements in the CARLA simulator.

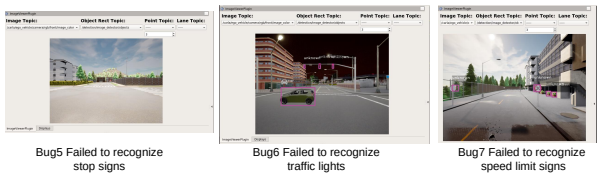


Figure 8: Illustration of Autoware Bug 5-7.

Bug8-9: Failed to recognize unseen elements. As shown in Figure 9, Autoware failed to recognize a child and a Coca-Cola truck, which could also be because these types of elements are not within the distribution of the training data of the object detection model, causing the model to be ineffective in recognizing such elements. Moreover, we also discovered that, for regular pedestrians, the angle of the pedestrian, the distance between the vehicle and the pedestrian, and the weather conditions (nighttime) can all contribute to issues with the model's recognition accuracy.

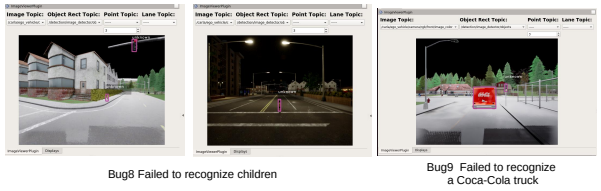


Figure 9: Illustration of Autoware Bug 8-9.

Bug10: Late pedestrian recognition. As shown in Figure 10, object detection only recognizes pedestrians when the vehicle is very close to them. However, the high speed of the vehicle makes it difficult to brake in such situations, resulting in a collision accident.

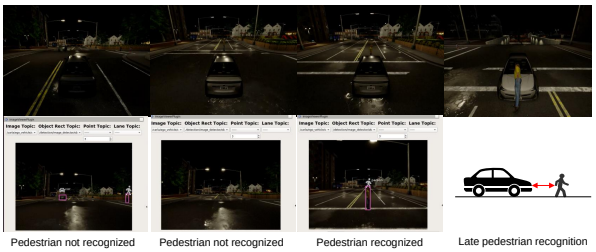


Figure 10: Illustration of Autoware Bug 10.

Bug11: Lidar detected the gradient reflection of the upslope/downslope as an obstacle, causing a stop. As shown in Figure 11, the vehicle often stalls when going up or downhill. This is because the lidar will scan the uphill or downhill road ahead, and the lidar detection model will recognize this part of the data points as obstacles (the area in the green frame in the figure), causing the vehicle to stop moving.

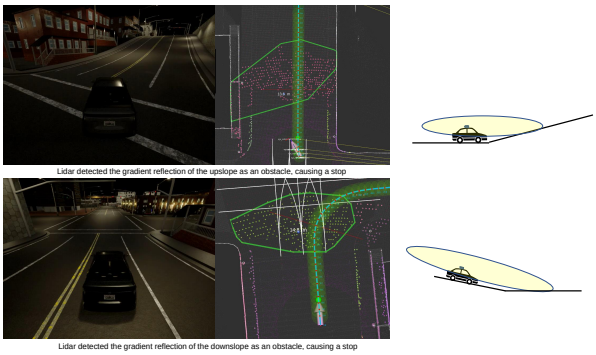


Figure 11: Illustration of Autoware Bug 11.

Bug12: Failed to predict the trajectory of the oncoming vehicle that reversing quickly. As shown in Figure 12, when the ego car faces a vehicle reversing rapidly ahead, the trajectory prediction model fails to predict the reversing trajectory of the vehicle (the situation below is successfully predicted, the red square area in the red circle), indicating that the trajectory prediction model does not have the ability to predict reversing situations, and the Autoware system also lacks the ability to respond in such situations.

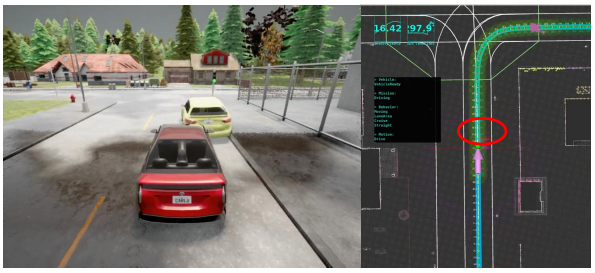


Figure 12: Illustration of Autoware Bug 12.

Bug13-15: Global planning module error. As shown in Figure 13, Autoware's global planning module has revealed significant problems during the testing process, often failing to make plans (bug13, the vehicle will stop at the starting point), taking a long detour for a short task path (bug14, the starting and ending points of the task path are marked in the middle figure, the blue line represents the global path planned by the global planning module), or making errors in planning (bug15, planning for unfeasible paths occurs, as shown in the red circle on the right).

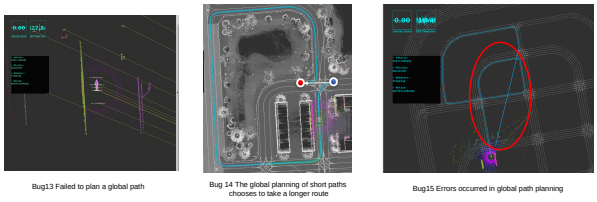


Figure 13: Illustration of Autoware Bug 13-15.

Bug16: Failed to make a reasonable local plan facing the oncoming vehicle. As shown in Figure 14, facing an oncoming vehicle ahead, the perception module perceives the oncoming vehicle, but the local planning module chooses to stop in place for the next step (the green dots in the right figure represent the next planned waypoint). Faced with such adversarial situations, Autoware's local planning module lacks the ability to respond.

Figure 14: Illustration of Autoware Bug 16.

Bug17: Failed to make a reasonable local plan facing the side coming vehicle. As shown in Figure 15, when faced with a vehicle driving from the side, the perception module senses the vehicle coming from the side and also predicts that the next trajectory of the car may collide. However, the local planning module is not able to make the correct local plan, such as setting the next waypoint further away and then accelerating forward; it still plans according to the normal situation. As shown in Figure 32, side collisions occur in a series of critical scenarios, so the local planning module should have more flexible planning capabilities to deal with these dangerous driving conditions.

Figure 15: Illustration of Autoware Bug 17.

Bug18: Rear-ended by a vehicle behind, and the local planning module lacks rear-end response capability. As shown in Figure 16, facing the tailgating vehicle from the rear, the self-driving system equipped with lidar can perceive the arrival of the rear vehicle. This is an advantage compared to human driving. But its local planning module has not responded accordingly.

Figure 16: Illustration of Autoware Bug 18.

Bug19: The vehicle stops running at a close position to the end point on a short turning path, without subsequent local planning. As shown in Figure 17, in some extremely short path settings, especially the kind of short path settings for turning as shown in the figure, when the vehicle is near the end point, it stops local planning. It chooses to stay in place.

Figure 17: Illustration of Autoware Bug 19.

Bug20: The calculation of the steering angle is unreasonable during turns, causing a collision in narrow curves. As shown in Figure 18, in an extremely narrow turn, the control module has a problem with the calculation of the turning angle under this situation (as shown in the red box in the figure), and the vehicle chose to go straight forward, hitting the road pile at the corner.

Figure 18: Illustration of Autoware Bug 20.

Bug21: Buildings, debris, walls, etc. on both sides causing lidar cluster algorithm to identify them as obstacles, affecting sensor positioning module, and thus affecting local planning. As shown in Figure 19, when there are many objects on both sides of the vehicle, such as buildings, walls, trees, etc., the lidar will scan the objects on both sides, causing a lot of noise in the data. The lidar detection model will judge them as obstacles, and it also affects the system's visual positioning model, causing errors in the prediction of the vehicle's posture (the yellow arrow in the figure is the predicted posture, and the red arrow is the actual posture). This in turn affects the calculation of the turning angle by the system control module (the output turning angle in the figure shows a left and right swing), and finally hits the side of the road.

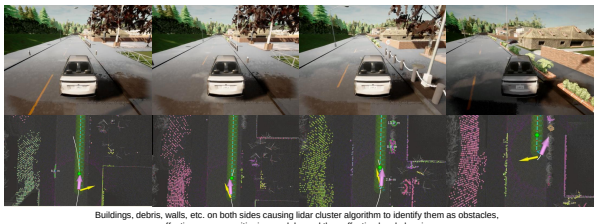


Figure 19: Illustration of Autoware Bug 21.

Bug22: The perception module recognized pedestrians, but the local planning still chose to move forward, and the control module did not generate a braking signal. As shown in Figure 20, the perception module predicts a pedestrian in front of the vehicle. But the local planning module still normally chooses to move towards the front waypoint, and the control signal does not output a braking signal (as shown in the red box in the figure), eventually causing a collision with the pedestrian.

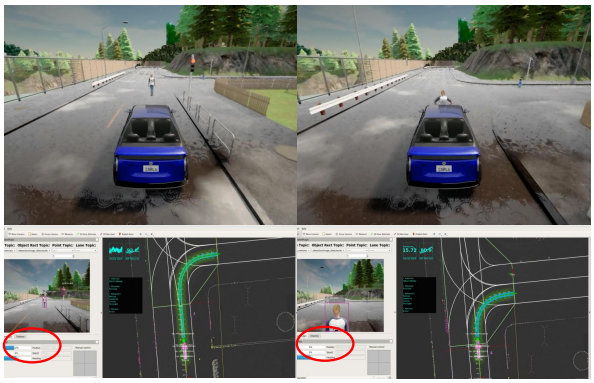


Figure 20: Illustration of Autoware Bug 22.

C.2 Basic

Bug1: Lack mechanism to detect pedestrians. The system retrieves background data from the CARLA simulator to query and determine the situation near the vehicle. However, when detecting nearby objects, the code only considers nearby vehicles and ignores queries about pedestrians. This causes the vehicle to directly hit pedestrians. The code is shown below: (The category name for pedestrians is "walker")

```
vehicle_list = actor_list.filter("*vehicle*")
vehicle_state, vehicle = self._is_vehicle_hazard(vehicle_list)
```

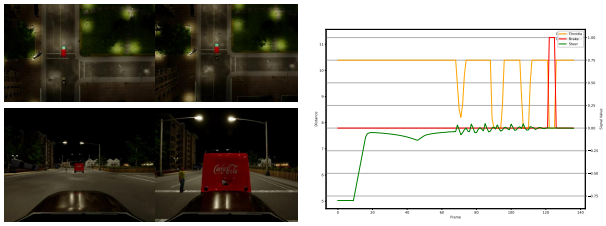
Bug2: Lack mechanism to detect objects behind and at the side-rear of the vehicle, only considers objects within 90 degree angle in front. The system only considers objects within a 90-degree angle range of the forward vector of the vehicle. For situations on the side and behind, it cannot effectively perceive and probe.

Bug3: Failures to detect pedestrians and vehicles due to vehicle speed or moving objects. For the detection and perception of front objects, since this part of the code only considers the position of nearby objects, but does not introduce dynamic situations such as the speed of the object, it is difficult to effectively perceive and probe the dynamic environment.

```
if is_within_distance_ahead(target_vehicle.get_transform(),self._vehicle.
      get_transform(),self._proximity_vehicle_threshold):
    return (True, target_vehicle)
```

Bug4-6. Similar to a series of complex driving conditions encountered by Autoware, for a series of critical scenarios that appear in front, behind, and side of the vehicle, the simple local planning module of this system is even more difficult to make effective plans.

Bug7: The control module did not stop immediately during emergency braking, causing a collision. As shown in Figure 21, the late release of the system's brake signal caused a collision with the front vehicle. (You can see the release of the brake signal in Figure 21b , but it's late.)



(a) Camera (b) Vehicle signal

Figure 21: Illustration of Basic Bug 7.

C.3 Behavior

Compared with Basic, its logical mechanism is more complex, but the bugs that occur are also quite similar, similar content will not be repeated (bug1-5,8).

Bug6: Errors in global planning. As shown in Figure 22, in the upper left corner of the figure, the blue path represents the task path, and the red path represents the path planned by the system. The other figures are the pictures during the process. This path is obviously wrong, and the vehicle finally crashes into the wall.

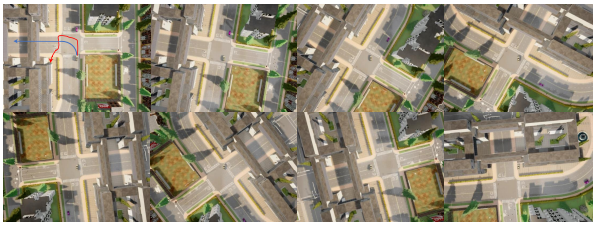
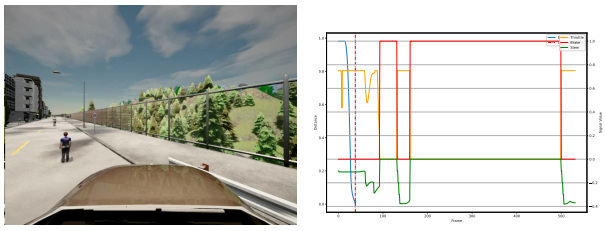


Figure 22: Illustration of Behavior Bug 6.

Bug7: Entered a stagnation state after braking for pedestrians. As shown in Figure 23, a pedestrian is standing in front of the vehicle. At this time, the vehicle will choose to wait in place. Facing a large driving space in front, the vehicle will not choose to bypass the pedestrian and move forward, but stop in place (as seen in the right figure, the red line represents the brake signal condition, and it is always in a state of braking).



(a) Camera (b) Vehicle signal

Figure 23: Illustration of Behavior Bug 7.

C.4 LAV

LAV is a multi-stage deep learning-based system. The types of bugs that occur will not be repeated (bug 2-6).

Bug1: Failed to recognize a pedestrian lying on the ground. Similar to the common situation in Autoware, but due to the low level of visualization of LAV, it is difficult for us to analyze the reasons. As shown in Figure 24, the pedestrian is lying in front of the vehicle, the vehicle does not generate any braking signals, and chooses to run over it.

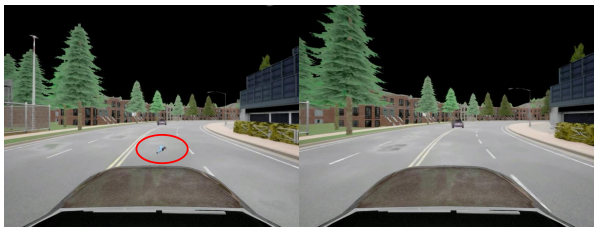


Figure 24: Illustration of LAV Bug 1.

Bug7: Error sheer angle calculated by the control module in turning situations. Compared with Autoware, there is a deviation in the calculation of the turning angle here. As shown in Figure 25, the control module of this system did not calculate the correct turning angle at the crossroads, and hit the side of the road.

C.5 NEAT

NEAT is a system based on an end-to-end deep learning model, using omnidirectional camera and BEV perspective camera images as inputs, and directly outputting planning and control signals. This work also carries out related visualization work, converting the output of the feature layer in the model into a BEV perspective prediction map, predicting the area of nearby objects, the road area, and the area where driving is not feasible.

Figure 25: Illustration of LAV Bug 7.

Bug1: More likely to hit pedestrians at night. As can be seen from Figure 26, under night conditions, there are many dark areas in the BEV images captured by the camera, and pedestrians are not obvious in the images, resulting in the system's output prediction map not predicting the presence of pedestrians in front of the vehicle (as shown by the red circle in the figure).

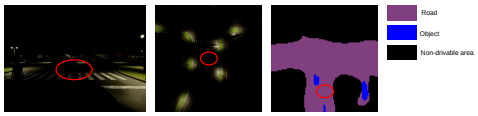


Figure 26: Illustration of NEAT Bug 1.

Bug2: Failed to recognize a Coca-Cola truck, causing a collision. As can be seen from Figure 27, the Coca-Cola truck is in the front side position of the vehicle, but there is a great misjudgment in the system's prediction of the vehicle's position in the prediction map, causing a collision.

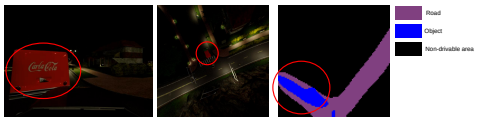


Figure 27: Illustration of NEAT Bug 2.

Bug3: Collides with pedestrians when they are near with multiple pedestrians or vehicles. As can be seen from Figure 28, there are multiple vehicles and pedestrians in front of the vehicle. Pedestrians are not obvious in the BEV image. Although pedestrians were captured in the omnidirectional camera image, the larger body size of nearby vehicles makes the model's attention during the prediction process More biased towards vehicles, did not predict the location of pedestrians (as shown by the red circle in the figure).

Figure 28: Illustration of NEAT Bug 3.

Bug4: Suddenly drove onto the pedestrian walkway during the driving process. As can be seen from Figure 29, the vehicle drove onto the sidewalk and hit a telegraph pole (the red path in the figure). As can be seen from the red circle in the middle figure, the model's ability to distinguish between roads and sidewalks is very weak, which often leads the system to drive onto the sidewalk.

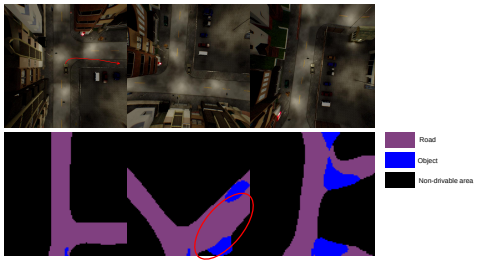


Figure 29: Illustration of NEAT Bug 4.

C.6 Transfuser

Transfuser is also a system based on an end-to-end deep learning model, using omnidirectional camera images and images converted from Lidar signals as inputs, and outputting planning and control signals.

Bug1: Data loss when converting Lidar signals to BEV images. As shown in Figure 30, data loss occurs in the process of converting Lidar signals to images. This will also affect the output of the system, thereby leading to incorrect behavior of the vehicle.

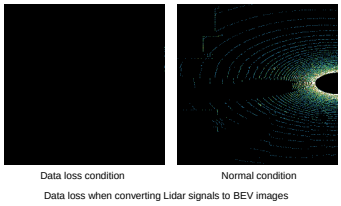


Figure 30: Illustration of Transfuser Bug 1.

Bug2: Collides with pedestrians in red clothes or vehicles with red appearance. As shown in Figure 31, in the analysis of the data, we found that in the collision scenario of this system, there is a bias towards colliding with objects with a red appearance. We conducted statistics on the appearance color of the objects being collided, and found that the distribution is biased towards the red series, as shown in the right figure.

Figure 31: Illustration of Transfuser Bug 2.

D DETAILS OF THE CRITICAL SCENARIOS.

The following are the corner simulation scenario demonstrations shown in Figure 32. These scenarios expose severe safety issues with the autonomous driving system. The vehicle's perception module, influenced by certain weather and time conditions, might fail to recognize some situations. Simultaneously, the planning module lacks understanding and learning of traffic rules, leading to potential collisions during interaction with other vehicles. The control module does not effectively consider the interaction between the vehicle body and the surrounding environment, leading to possible scrapes due to poor management of vehicle distance or road width.

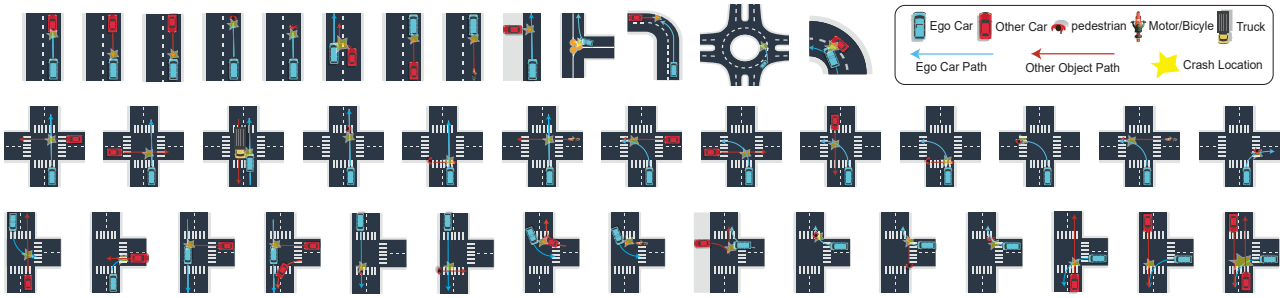


Figure 32: Illustration of the critical scenarios generated where collisions are prone to occur.

Figure 33: Ego car hits the vehicle in front due to perception or control module issues.

Figure 34: Ego car lacks the ability to respond when a vehicle in front is reversing.

Figure 35: Ego car lacks the evasion ability when an oncoming vehicle approaches from the front.

Figure 36: Ego car hits a pedestrian in front due to perception or control module issues.

Figure 37: A vehicle in front knocked a pedestrian to the ground and ego car, failing to recognize the fallen pedestrian, ran over them.

Figure 38: Ego car lacks the ability to respond when a vehicle from the side aggressively merges into its lane.

Figure 39: Ego car lacks evasion ability when rear-ended by a vehicle from behind.

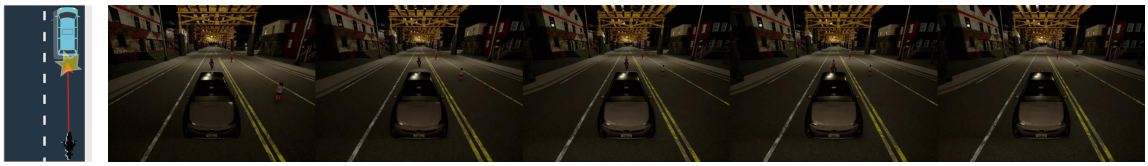


Figure 40: Ego car lacks evasion ability when rear-ended by a bicycle/motorcycle from behind.

Figure 41: Ego car collides when a roadside vehicle reverses without waiting.

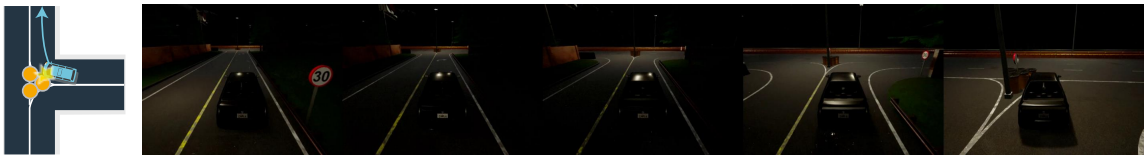


Figure 42: Ego car collides with the fence on a narrow turn.

Figure 43: Ego car lacks the ability to respond to a reversing vehicle while making a turn.

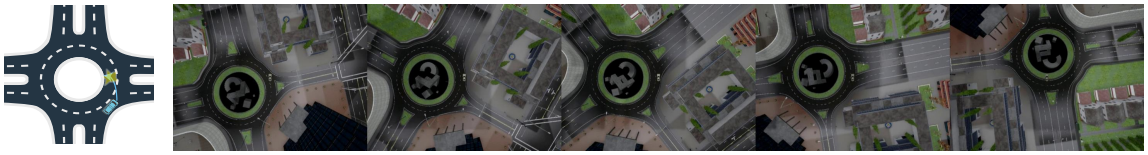


Figure 44: Ego car shows incorrect planning behavior on a circular route due to road structure interference.

Figure 45: Ego car fails to maintain proper distance, causing a scrape with other vehicles.

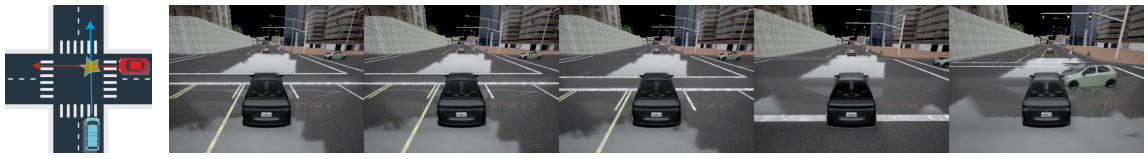


Figure 46: Ego car lacks flexible response ability when hit by a vehicle from the right while going straight at an intersection.

Figure 47: Ego car lacks flexible response ability when hit by a vehicle from the left while going straight at an intersection.

Figure 48: Ego car fails to maintain proper distance, scraping against a large vehicle coming from the front at an intersection.

Figure 49: Ego car hits a pedestrian on the crosswalk in front due to perception or control module issues while going straight at an intersection.

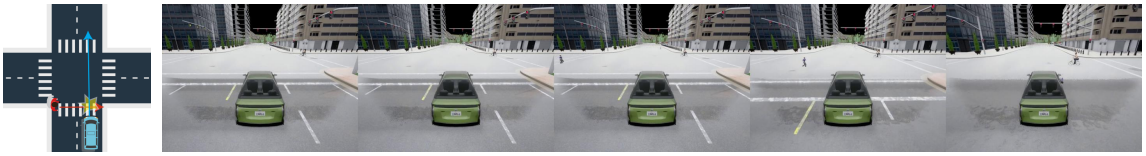


Figure 50: Ego car hits a pedestrian on the crosswalk before it due to perception or control module issues while going straight at an intersection.

Figure 51: Ego car lacks clear understanding of traffic rules, colliding with a bicycle coming from the right while going straight at an intersection.

Figure 52: Ego car collides with a vehicle from the right while making a left turn at an intersection due to lack of clarity on traffic rules regarding yielding to oncoming traffic.

Figure 53: Ego car collides with a vehicle from the left while making a left turn at an intersection due to lack of clarity on traffic rules regarding yielding to oncoming traffic.

Figure 54: Ego car collides with a vehicle from the front while making a left turn at an intersection due to lack of clarity on traffic rules regarding yielding to oncoming traffic.

Figure 55: Ego car hits a pedestrian on the crosswalk before it due to perception or control module issues while making a left turn at an intersection.

Figure 56: Ego car hits a pedestrian on the crosswalk at the left corner due to perception or control module issues while making a left turn at an intersection.

Figure 57: Ego car lacks clear understanding of traffic rules, colliding with a bicycle coming from the right while making a left turn at an intersection.

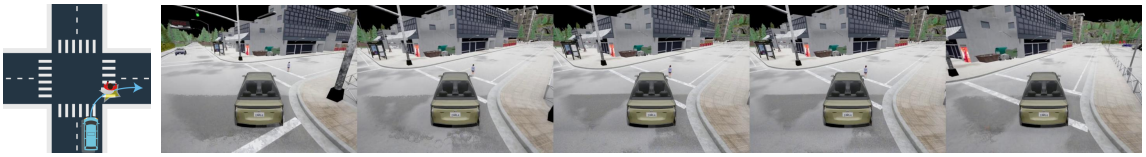


Figure 58: Ego car hits a pedestrian on the crosswalk at the right corner due to perception or control module issues while making a right turn at an intersection.

Figure 59: Ego car lacks flexible response ability when hit by a vehicle coming straight from the front while making a left turn at a T-intersection.

Figure 60: Ego car lacks flexible response ability when hit by a reversing vehicle from the right while making a right turn at a T-intersection.

Figure 61: Ego car lacks flexible response ability when hit by a vehicle from the left while going straight at a T-intersection.

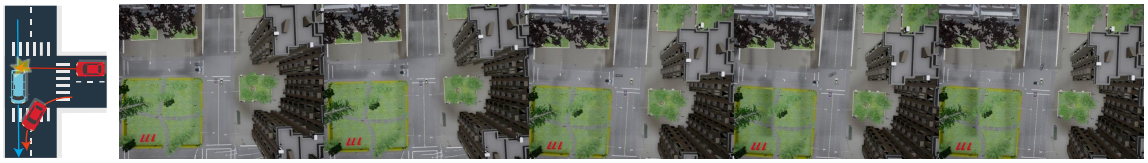


Figure 62: Ego car lacks proper response ability in multi-vehicle situations, getting hit by another vehicle from the left while yielding to a left-turning vehicle in front at a T-intersection.

Figure 63: Ego car hits a pedestrian on the crosswalk in front due to perception or control module issues while going straight at a T-intersection.

Figure 64: Ego car hits a moving pedestrian on the crosswalk in front due to perception or control module issues while going straight at a T-intersection.

Figure 65: Ego car fails to maintain proper distance, scraping against a vehicle at the same junction while making a left turn at a T-intersection.

Figure 66: Ego car lacks clear understanding of traffic rules, colliding with a bicycle on the left while making a left turn at a T-intersection.

Figure 67: Ego car lacks understanding of other vehicles' intent and the rule of yielding to reversing vehicles, colliding with a vehicle preparing to enter the road from the front while making a right turn at a T-intersection.

Figure 68: Ego car hits a pedestrian on the crosswalk at the right front due to perception or control module issues while making a right turn at a T-intersection.

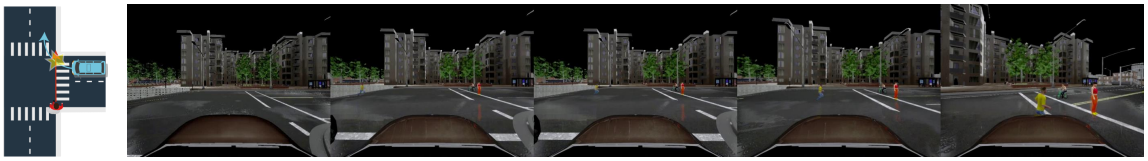


Figure 69: Ego car hits a moving pedestrian on the crosswalk in front due to perception or control module issues while making a right turn at a T-intersection.

Figure 70: A vehicle in front knocked a pedestrian to the ground and ego car, failing to recognize the fallen pedestrian while making a right turn, ran over them.

Figure 71: Ego car collides with a vehicle coming from the left front while making a left turn at a T-intersection due to lack of clarity on traffic rules regarding yielding to oncoming traffic.

Figure 72: Ego car collides with a vehicle coming from the right front while making a left turn at a T-intersection due to lack of clarity on traffic rules regarding yielding to oncoming traffic.

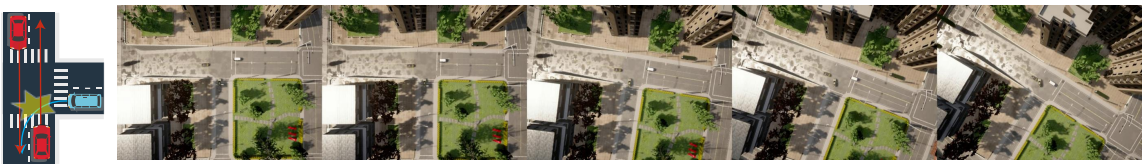


Figure 73: Ego car collides when vehicles are coming straight from both sides at a T-intersection, due to lack of clarity on traffic rules regarding yielding to oncoming traffic while making a left turn.