Compare and Analyzed safety violation in autonomous driving systems with Random Fuzzer: Baidu apollo 5.0 vs Baidu apollo 6.0

1.0 introduction

Baidu Apollo is one of the few L5-level open-source autonomous driving systems on the market; however, the safety of autonomous driving systems is an enduring topic. Previous papers have undergone a lot of analysis on this, such as ASF, AV-fuzzer. After the publication of these papers, it is undeniable that the developers of Apollo also paid a lot of effort to upgrade and improve the system, whether it is to fix minor bugs or improve the algorithm; I believe this will also be a massive improvement in safety performance.

But what is difficult for us to know is how to measure the difference in safety performance between different versions of Baidu apollo. To express the difference more intuitively, we only use one data to measure it, which is the fault rate (more

detail in the following sections). To solve such a problem, we use the fuzz test method by using the pc platform simulator LGSVL. The cost of this method is very low compared to the real road testing, and it is a very commonly used black-box testing method, for which we propose：

We use random Fuzzer to test and compare the safety performance of the autonomous driving platform Baidu Apollo 5.0 and Baidu apollo 6.0 in LGSVL simulator.

2.0 background

2.1 Baidu apollo 6.0 and 5.0

figure 1 framework of Apollo 5.0

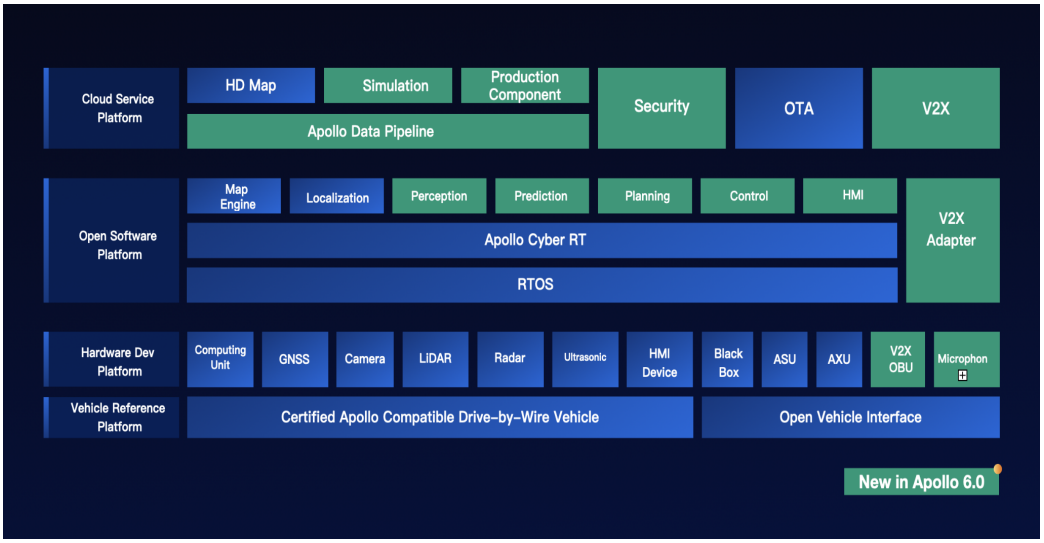


figure 2 framework of Apollo 6.0

Comparing Apollo 5.0 (Figure 1) and 6.0 (Figure 2) (from the official documentation), we can see that developers have improved many modules. We only need to test the open-source platform for our random fuzz, which is located in the second layer of the entire stack. This is the core of the autonomous driving system. It includes control, perception, planning, prediction, and localization modules. During the test, we only need to open these modules to make ads run. Other modules are not involved in the trial, such as the hardware for perception, cameras, and lidars which were provided by lgsvl simulator. During the test, their configurations are the same for both ADS. And communication systems such as V2X (connected cars) do not affect our safety testing.

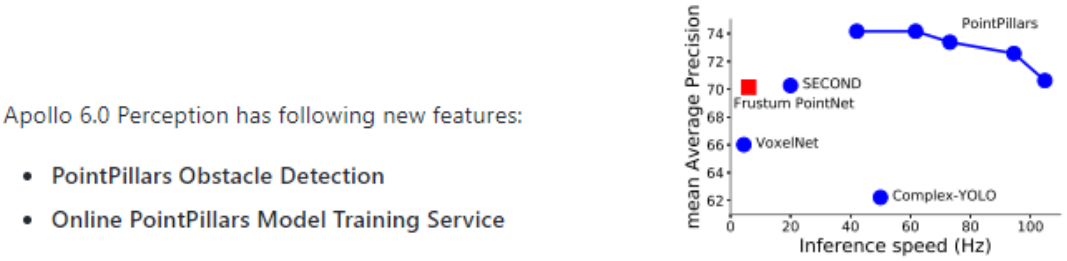


fig3 &4 *official doc of baidu apollo 6.0 explanation of updates, point pillar model's inference speed and accuracy*

By reading the official documents, Baidu Apollo 6.0 mainly provides a new radar perception algorithm called point pillars. Compared to the previous version. PointPillars is a deep network for 3D object detection for lidar point clouds. The output is 3d bounding boxes of obstacles; PointPillars compared with other methods could offering higher detection accuracy at a faster inference speed for only 15ms. It has an outstanding balance of precision and speed; if you need to understand the construction of neural networks, please see their paper.

https://openaccess.thecvf.com/content_CVPR_2019/papers/Lang_PointPillars_Fast_Encoders_for_Object_Detection_From_Point_Clouds_CVPR_2019_paper.pdf

2.2 LGSVL

lgsvl is a real-time 3D simulator of self-driving vehicles, powered by Unity's physics engine and LGSVL driver to provide a complete point cloud map, high-res map, NPC control, and a relatively fine-grained road state and weather system. The simulator

provides a bridge with the ADS, which means that the information in the simulator can be transmitted to the autonomous driving system for processing in real-time, such as images and point clouds. After the ADS receives the information, ads will process the data and send it back to lgsvl with control commands in real-time for the ego car.

3.0 Hypothesis

The point pillar model featured in the Apollo 6.0 will accelerate the inference speed that could let our ego car have more reaction time to avoid hitting the obstacle, lowering the safety violation.

In general, shortening the time of the radar obstacle perception module of the perception module will improve the operation of the overall module. In abstract, if we define the frame generation time in apollo5.0 as 120ms, without considering the parallelism data processing , the radar perception may take 30ms. Still, in apollo 6.0, the point pillar only takes 15ms, so the frame generation will only be 105ms, and the fast frame generation will bring more fine-grained operations to the control module. And improve safety performance and reduce the possibility of safety violations.

4.0 experiment setup

4.1 Experiment environment

The system I use is ubuntu18.04 system with 8 core AMD CPU, 1080ti graphic card, and 32 GB memory, which is a very standard configuration for running an ADS simulation in a PC simulator.

4.2 Fuzz testing

Based on the previous background knowledge and assumptions, we designed a random fuzzer, the random fuzzer is used for randomly generating NPC's behavior in a certain scenario, and the output is how ADS going to react to NPC's behavior and what is the possibility that will lead to EGO car commit safety violation as known as ego at fault accident, and I calculated the fault rate by the number of safety violation case divided the total number of trials.

- fault rate = accidents caused by egocar (ego at fault case) / total number of test case(total of tirals)

To give a simple example, random fuzzer runs 2000 times, and I get 2000 trials, including 200 accidents, these accidents include accidents caused by NPC and accidents caused by ego, of which there are 120 accidents caused by ego, And

there are 80 caused by NPC, I get fault rate = 120/2000 = 0.6% Then compare the fault rates obtained by testing of the two versions of ADS.

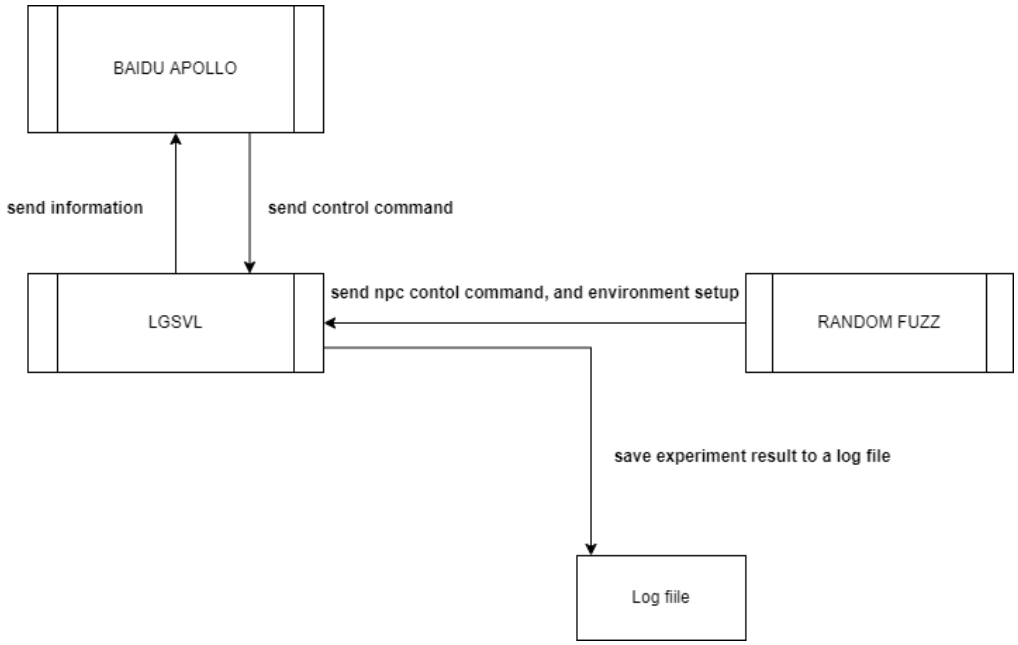


Fig5 workflow of random fuzz

This is the data pipeline of the entire random fuzzer. We start with the lgsvl simulator. In the simulator, hardware such as lidar, IMU and camera will send perception information and localization information to ADS, ADS will process this information and ego. The control command is sent back to the simulator to control the driving of the ego car in the simulator.

The control of the lgsvl simulator is our random fuzzer, which completes the experiment by controlling the configuration of the lgsvl simulator, setting the initial positions of the npc and ego car, the behavior of the npc vehicle, etc., including the running time of the experiment and the frequency of information collection, etc. , where we store the collected information in a log file.

```
= current ego location
Vector(7.3252124786377, -0.00358587503433228, -12.5048742294312)
= current ego speed
6.459923764907668
= current ego angle
Vector(359.976837158203, 359.854125976563, 0.000389291526516899)
= current npc location
Vector(7.78389310836792, -0.0121140480041504, -20.3487739562988)
= current npc speed
7.7009233386171125
= current npc angle
Vector(-0.00060963915893808, 358.602264404297, -1.47549872053787e-06)
= current ego location
Vector(7.31638240814209, -0.00376367568969727, -9.09359931945801)
= current ego speed
7.390537495124599
= current ego angle
Vector(359.873901367188, 359.847045898438, -7.60135008022189e-05)
= current npc location
Vector(7.66752052307129, -0.0121139287948608, -15.6361589431763)
= current npc speed
8.65457342963162
= current npc angle
Vector(-0.000610716524533927, 358.564605712891, -4.02777658337072e-07)
= current ego location
Vector(7.30661869049072, -0.00380682945251465, -5.33417129516602)
```

Fig。6 log file

We used the technique of random fuzz testing to lead this experiment. I used a simple scenario: the map is 2 lane highway, 2 npcs are the traffic participants for this experiment, and our ego car, the initial location of one of the NPC is in the left rear position of our ego car. The behavior of this NPC is generated by a random seed generator. The random seed generator generates a list before the start of each trial. This list contains ten groups of tuples; each tuple contains the turning command (0 is stay, 1 is turn left lane, 2 is turn right lane of the npc and the expected speed. The NPC follows these randomly generated instructions and changes every 2 seconds. So the total time of one trail is 20s, Another NPC is set to travel at a certain speed in front of the ego car.

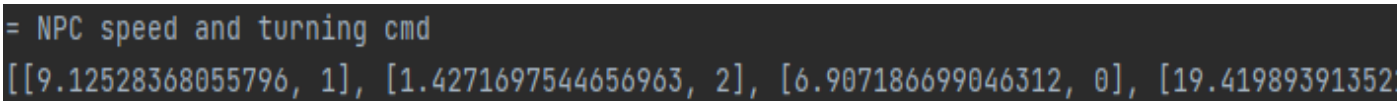


Fig7 log file of randomly generated npc expected speed and turning command

The initial position of our ego car is set between two NPCs, and the final destination is set in the Apollo dream view, near the end of the road. It teleports to the initial target point at the beginning of each experiment, in order to test as much data as possible. I ran random fuzz 2000 times per Apollo 5.0 and 6.0.

Our random fuzz is using lgsvl's python API. In each experiment, every 0.5 seconds, I will return vehicle information from lgsvl, including the speed and position of the three vehicles, as well as the steering angle, including when the accident happened; we also have an analyzer to analyze the liability of the accident, the analysis method is also very simple, we only need to check the collision position of the ego car and the NPC car, we can judge the liability, In this case, if the car in behind collides with the car in front, and the liability will be judged as the car behind. In this case, we will judge it as an ego car at-fault accident. Finally, we store all the information in a log file for the next steps of analysis.

5.0 result

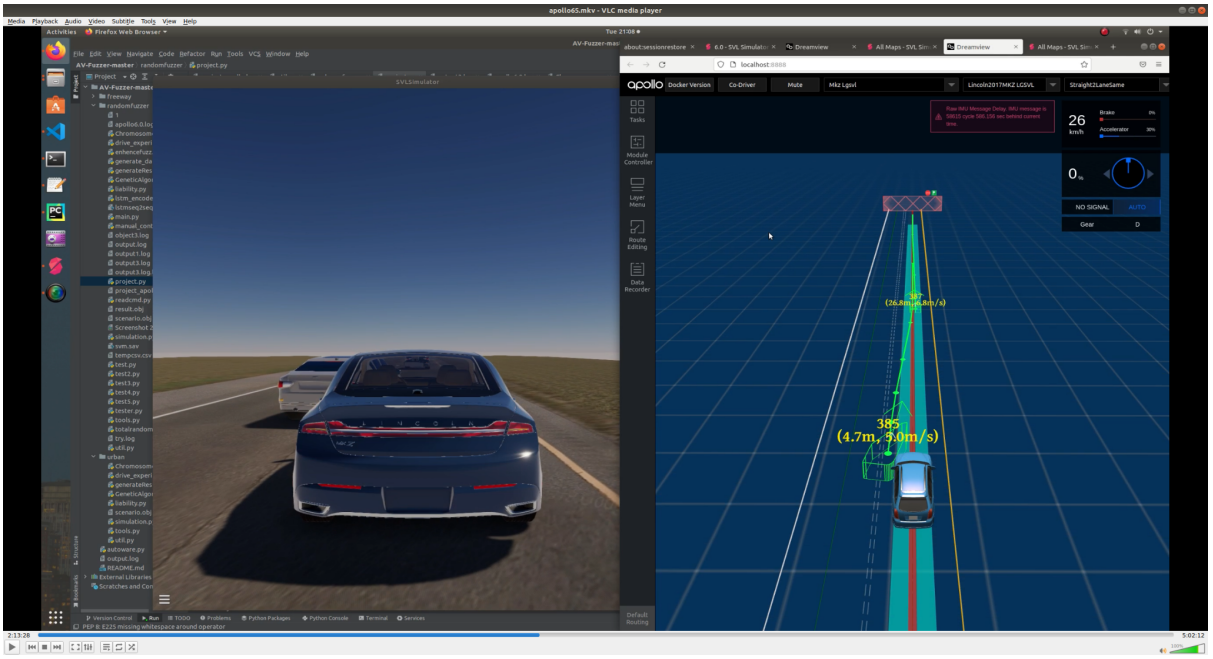


Fig 8 accident happened after an npc car ghost cut into the ego car's lane.

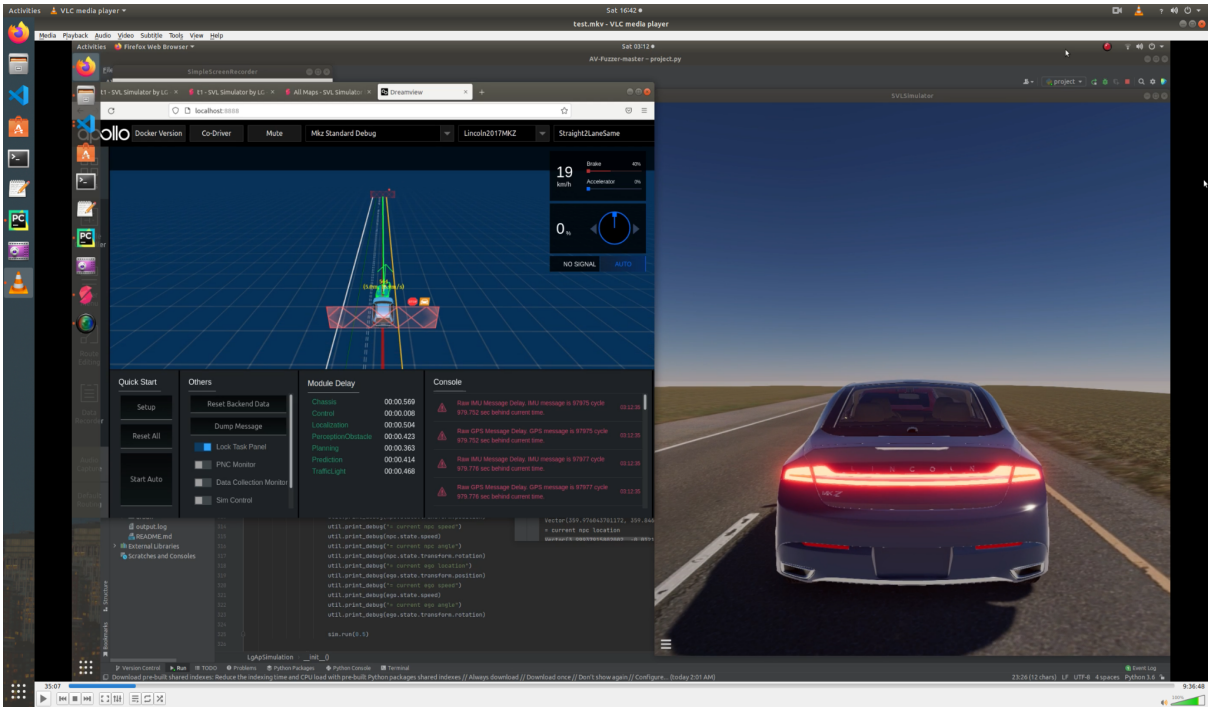


Fig 9 accident happened ego car lost brake control and cause rear-end collision

This is the accident sample I collected when I ran the random fuzzer. The first case is that the random fuzzer generates random npc commands. Sometimes the npc vehicles will randomly change lanes to the ego car's lane, which makes the ego car not have enough time to make any reaction so that the ego car will hit the side or the back of the npc car directly, which is the case of ghost cut-in,

Another situation will be fascinating. Our ego car directly loses its effective control of the brakes, and directly rear-ends the vehicle in front of the NPC driving at a constant speed. This situation is relatively rare. The problem is that the prediction module has incorrect speed estimates for constant speed ahead of the NPC car, resulting in rear-end collisions due to failure to brake in time.

Both of these situations are caused by the ego car not braking in time.

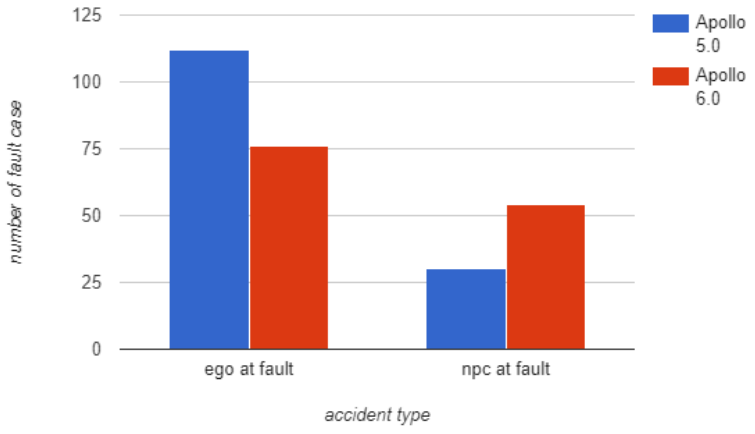


Fig 10 number of fault case of accident type ego at fault and npc at fault

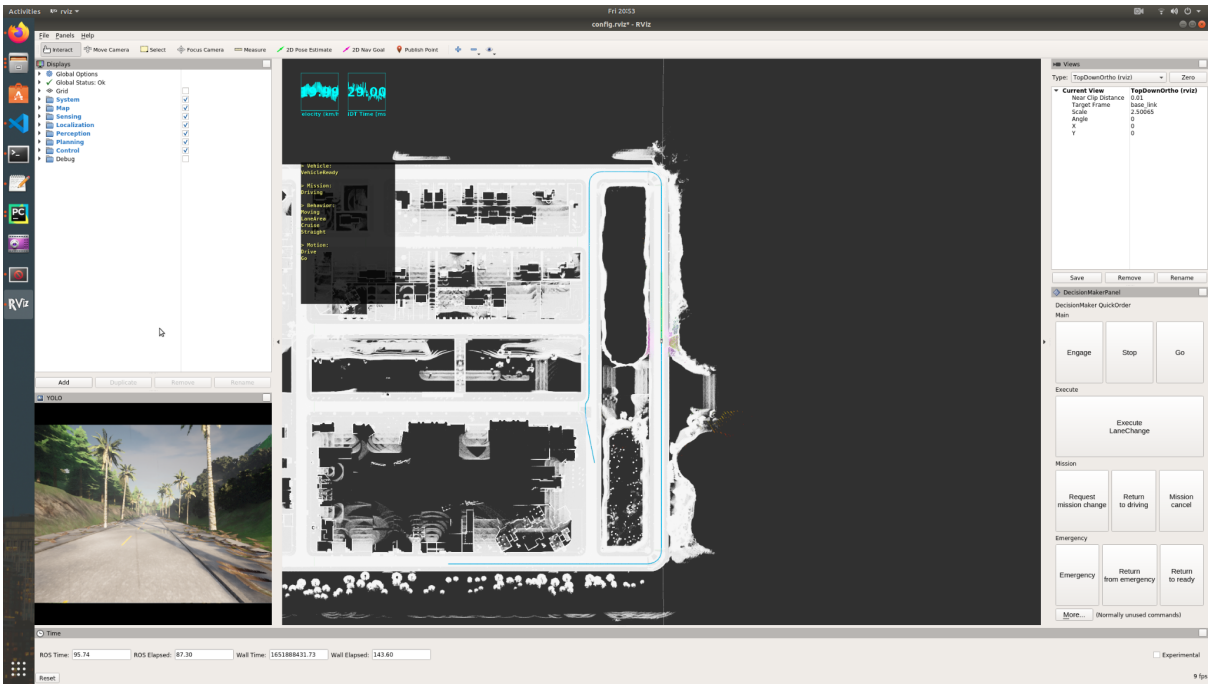
After running 2000 trials for each Apollo 5.0 and Apollo 6.0, by reading the log files, I found that the two autonomous driving platforms have apparent differences in safety performance. From the data I collected, the accident caused by ego happened 112 times in Apollo 5.0, but only 76 times in Apollo 6.0, so we can get the fault rate of Apollo 5.0 is 112 / 2000 = 0.58%, Baidu. The fault rate of Apollo 6.0 is 76/2000 = 0.38%. Comparing the fault rates of the two platforms, Baidu Apollo 6.0 has 36 fewer accidents than 5.0; Apollo 6.0 has 32.2% less chance to cause an ego at-fault accident than Apollo 5.0.

6.0 Limitation

Of course, the running efficiency of random fuzzer is still relatively low, and it will take a lot of time. As a benchmark, it takes a lot of time to run 2000 times. However, it is still not enough for random fuzzer, so We need to find each specific fault case, store it, and run it hundreds to thousands of times to observe and compare whether there are differences between platforms so that the comparison security will be more accurate and rigorous. If we need to observe the improvement of the Apollo 6.0 platform to the previous platform, we also need to modify the source code to test the change of the module, which is very time-consuming.

7.0 future works

In future work, I will use the carla simulator to compare other autonomous driving platform, autoware and Baidu apollo. There are many problems with autoware running in lgsvl temporarily. Baidu apollo cannot run in Carla temporarily. Carla has a smoother and more detailed config than lgsvl, for which I will rewrite the random fuzzer for Carla. Compare the fault rate of two completely different platforms.



8.0 Conclusion

In most cases, the occurrence of ego at-fault accident is due to the lack of braking in time, which leads to the ego car commit accident, but if we want to research a

detailed reason of an accident happened, you need to test all the component's internal state of ads while running, which is very time-consuming, but a simple random fuzzer test can prove that the efforts of developers have improved safety performance.

CITATION

- Lang, Alex H., Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang and Oscar Beijbom. “PointPillars: Fast Encoders for Object Detection From Point Clouds.” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019): 12689-12697.

- Hu, Zhisheng et al. “Coverage-based Scene Fuzzing for Virtual Autonomous Driving Testing.” *ArXiv* abs/2106.00873 (2021): n. pag.

- G. Li *et al*., "AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems," *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 25-36, doi: 10.1109/ISSRE5003.2020.00012.