AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems

Guanpeng Li*, Yiran Li*, Saurabh Jha*, Timothy Tsai†, Michael Sullivan†, Siva Kumar Sastry Hari†, Zbigniew Kalbarczyk*, Ravishankar Iyer*

*University of Illinois at Urbana-Champaign, †NVIDIA

***Abstract*—This paper proposes AV-FUZZER, a testing framework, to find the safety violations of an autonomous vehicle (AV) in the presence of an evolving traffic environment. We perturb the driving maneuvers of traffic participants to create situations in which an AV can run into safety violations. To optimally search for the perturbations to be introduced, we leverage domain knowledge of vehicle dynamics and genetic algorithm to minimize the safety potential of an AV over its projected trajectory. The values of the perturbation determined by this process provide parameters that define participants' trajectories. To improve the efficiency of the search, we design a local fuzzer that increases the exploitation of local optima in the areas where highly likely safety-hazardous situations are observed. By repeating the optimization with significantly different starting points in the search space, AV-FUZZER determines several diverse AV safety violations. We demonstrate AV-FUZZER on an industrial-grade AV platform, Baidu Apollo, and find five distinct types of safety violations in a short period of time. In comparison, other existing techniques can find at most two. We analyze the safety violations found in Apollo and discuss their overarching causes.**

***Index Terms*—Autonomous vehicles, safety-critical applications**

I. INTRODUCTION

From reducing traffic congestion to improving access to transportation, autonomous vehicles (AVs) hold significant potential to increase productivity and improve quality of life. Ensuring AV safety is critical to success in the marketplace, particularly since there is a public perception problem with regard to their safety, and this public wariness impacts AV vendors' decisions to produce and deploy these vehicles.

Testing is an essential aspect of AV development that ensures the vehicles driven by self-driving software are safe. Commonly, AVs are tested using stress-testing techniques. A popular practice is leveraging human-specified inputs, where a human operator specifies the maneuvers of surrounding vehicles in traffic and observes the target AV's behavior [1]. However, this approach is expensive and takes a very long time due to extensive human labor involvement. Recently, researchers have proposed automatic (or semiautomatic) techniques to generate test cases for AVs [2]–[6]. These approaches either use random sampling methods or rely on training machine-learning models to navigate test case generation [4], [7]. However, these techniques often overlook edge cases or tend to repeatedly find failures similar to ones already discovered. Moreover, they can be inefficient in testing real-world AVs with industrial-grade simulators, as the state space of vehicle behaviors (including the physical states of all the vehicles and the internal software states of the AV) are huge. AV technology vendors, such as Baidu Apollo, release software updates on a weekly basis [8], [9]. It is challenging to scale existing AV testing techniques to such a rapidly developing AV industry.

This paper proposes AV-FUZZER, an efficient AV testing framework that generates test cases to determine the safety violations of an AV in the presence of an evolving traffic environment. We perturb the driving maneuvers of traffic participants (e.g., other vehicles in the environment) to creating situations in which an AV runs into safety violations. Our approach is based on the following insights and observations: (1) We can formulate the search of the perturbations to be introduced as an optimization problem that can be solved using genetic algorithm and the domain knowledge of vehicle dynamics. We minimize the safety potential of an AV over its projected trajectory, and the values of the perturbation determined by this process provide participants' parameters that define their trajectories. (2) The efficiency of finding safety violations can be improved by designing a local fuzzer that dynamically increases the exploitation of local optima in areas where highly likely safety-hazardous situations (i.e., near-miss accidents, etc.) are observed. (3) We develop a restart mechanism that repeats the optimization with significantly different starting points in the unexplored search space, to determine diverse safety-hazardous situations in which the AV will run into safety violations.

We demonstrate AV-FUZZER on Baidu Apollo, an industrial-grade, level-4 AV software stack widely used to control AVs on public roads [10], [11]. We find several safety-critical deficiencies in Apollo that have not been discovered or reported before, and we are able to find these safety violations in a relatively short period of search time. Specifically, we find 13 critical scenarios in which Apollo runs into hazardous situations that lead to crashes. In contrast, other techniques, such as random fuzzing and adaptive stress testing [4], find only 1 and 5 safety violations, respectively, in the same amount of search time. We then analyze the overarching causes of the safety violations AV-FUZZER reports in Apollo and characterize them into 5 distinct types that map to various categories of software deficiencies in that system. Of the 5 types, 2 mimic real-world AV accidents reported to the California DMV [12] in the past. While AV-FUZZER finds all 5 types within 20 hours of search, existing techniques [4] can find at most 2 distinct types, even given 10x the search time (200 hours).

Because AV-FUZZER is both efficient and effective, it can be integrated into the AV development cycle. *To the best of our*

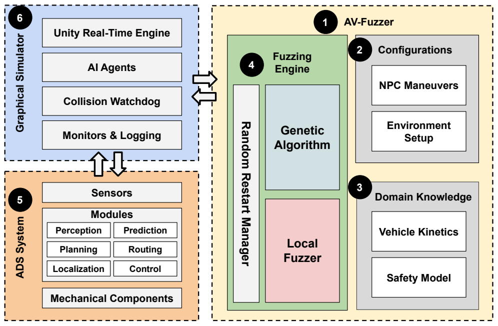


Figure 1: A high-level overview of AV-FUZZER framework, including the graphical simulator, and the AV architecture.

knowledge, this is the first study that (i) proposes an efficient testing framework that assesses how driving maneuvers of traffic participants impact the safety of an end-to-end, real-world, industrial-grade AV system and (ii) investigates the software deficiencies of Apollo that lead to safety violations in autonomous driving.

To summarize:

- We propose AV-FUZZER[1], an AV testing framework that quickly finds diverse safety violations caused by an AV in the presence of a dynamically changing traffic environment.
- We demonstrate the capabilities of AV-FUZZER on an industry-grade, level-4 AV technology stack, Baidu Apollo, and find more safety violations much more efficiently than other existing AV test methods.
- We analyze and classify the safety violations AV-FUZZER reports in Apollo into 5 distinct types, and discuss their overarching causes.

II. BACKGROUND

In this section, we describe the architecture of an autonomous driving system, the high-level concept of genetic algorithms, and the high-fidelity simulations used in this study.

A. Autonomous Driving Systems

AVs use Autonomous Driving System (ADS) technology to replace human drivers in controlling a vehicle's steering, acceleration, and monitoring of the surrounding environment (e.g., other vehicles) [3], [8], [13]. A modern ADS architecture consists of a sensor layer and six basic modules [9], as shown in ❺ in Fig. 1.

Sensor Layer: The sensor layer preprocesses input data and filters sensor noise. An ADS supports a wide range of sensors, such as cameras, inertial measurement units (IMU), Global Positioning Systems (GPS), sonar, RADAR, and LiDAR. In this study, we use the prototype vehicle Baidu Apollo, which is equipped with two camera sensors (one at the top and another in front of the vehicle) and one LiDAR [9].

Perception Module: The perception module reads data from the sensor layer to detect static objects (e.g., lanes, traffic signs, or barriers) and dynamic objects (e.g., passenger vehicles or trucks) in the traffic environment using computer vision and deep-learning techniques. The object detection algorithm performs tasks, such as segmentation, classification, and clustering, based on the sensor data from individual sensors. Then it uses fusion techniques such as extended Kalman filters [14], to merge the data and generate a final track list of objects.

[1]AV-FUZZER can be downloaded at https://github.com/cclinus/AV-Fuzzer

Localization Module: The localization module is responsible for providing the location of the AV. It fuses multiple sets of input data from various sources to locate the AV in the world model. It does so via aggregation of input data from GPS, IMU, and LiDAR sensors.

Prediction Module: The prediction module is responsible for studying and predicting the behavior of all the objects detected by the perception module in the world model. It generates basic information on objects, such as their positions, headings, velocities, and accelerations, and then uses this data to generate predicted trajectories with probabilities for those objects.

Routing Module: The routing module generates high-level navigation information based on the current location and destination of the AV. The output of the module, passage lanes, and roads are computed based on the HD map.

Planning Module: The planning module generates navigation plans based on the origin and destination of the AV and computes a safe (i.e., collision-free) driving trajectory for the AV using the output data from the localization and prediction modules.

Control Module: The control module takes the planned trajectory as input and generates control commands (e.g., actuation, brake, steer) to pass to the CAN bus, which delivers the information to the AV's mechanical system.

B. Genetic Algorithm

Genetic algorithm (GA) [15], [16] is a meta-heuristic search algorithm inspired by natural evolution. The algorithm starts with an initial set of candidate solutions, which are collectively called the *population*. The algorithm is driven by a *fitness function* that computes the fitness score of a candidate. The fitness score reflects how good the candidate is at solving the problem.

At each stage, some candidate solutions are chosen from the population for *recombination operations*. There are two types of recombination operations: (a) *crossover* and (b) *mutation*. In crossover, two candidates are randomly chosen and exchanged in the hope of generating a better solution from a good one. This operation tends to narrow the search and move toward an optimal solution. In mutation, one candidate is randomly selected. The operation flips a bit or an entity in a solution, which expands the search exploration of the algorithm. In general, recombination operations give rise to new, better-performing members, which are then added to the population. In contrast, members that have poor fitness scores are gradually eliminated. Each such processes is called a *generation* and is repeated until either a population member has the desired fitness score (hence a solution is found) or the algorithm terminates upon exceeding the time allocated to it.

C. High-Fidelity Simulation Platform

We use an Unreal Engine (UE) based real-time simulation platform, LGSVL [17], that is capable of simulating complex urban and freeway driving scenarios using a library of urban layouts, buildings, pedestrians, and vehicles. The simulator can generate various sensor data at regular intervals (from cameras, LiDARs, etc.) that can be fed to the ADS platform, providing an end-to-end simulation environment for testing ADSs. The simulator supports a representative model of physics that mimics real-world vehicle dynamics, enabling a high-fidelity simulation of vehicle behaviors and their moving trajectories. The high-fidelity simulation also indicates a large search space of vehicle states (i.e., positions, velocities, AV internal software states, etc.). The simulation platform is a typical environment used by the AV industry to develop and test their ADSs.

III. AV-Fuzzer Overview

This section presents an overview of AV-Fuzzer. Our overarching goal is to efficiently identify traffic maneuvers of surrounding vehicles that lead to AV safety violations. We use GA to minimize a defined fitness function (equivalent to the safety of the AV) to guide the search for problematic maneuvers. In this context, each time we find a highly likely safety-violation case, we trigger further exploitation of the surrounding areas (i.e., perturbing parameters of the surrounding traffic) to pinpoint hazardous conditions that lead to safety violations. We do so by using GA to first explore towards high-potential areas in the search space where the safety of the AV is low. Then we leverage a local fuzzer to extensively exploit each of these high-potential cases and evolve them into safety-violation scenarios, if possible. When we observe the process getting stuck, we restart the search from a significantly different starting point in an unexplored area in the search space, based on the historical trajectories we have already explored. By moving across the optimization space in this way, we are able to determine a number of diverse safety-violation scenarios of the AV.

Fig. 1 shows the overall structure of the AV-Fuzzer framework, as well as diagrams of the simulator and the AV under test. AV-Fuzzer (❶) consists of a configuration module (❷) that specifies the driving environment, a domain knowledge module (❸) that defines safety model and vehicle kinetics, and a fuzzing engine (❹) that drives test scenario generation. AV-Fuzzer is connected to a graphical simulator (❻) that feeds sensor inputs to the AV under test (❺) and provides a simulation platform. In this paper, we call non-AV traffic participants *non-player characters* (NPCs) or target vehicles (TVs). We refer to the AV under test as the ego vehicle (EV) or the AV. We define a *scenario* as a sequence of NPC maneuvers performed in a given a driving environment in a simulation.

A. Configuration: Driving Environment Setup (❷ in Fig. 1)

The configuration module specifies the driving environment in which the test is conducted. In the setup, the user of AV-Fuzzer specifies a *driving environment*: the road structure where the test happens, NPCs and their initial states, and the allowed maneuvers of NPCs in the test. The driving environment can be chosen from existing road structures supported by the simulator used. For example, in our study, we use the *LGSVL* graphical simulator [17] that supports both freeway and urban road structures. Users can also create other road structures for their tests. AV-Fuzzer is not tied to particular configurations. In this study, the allowed NPC maneuvers consist of acceleration/deceleration, following lane, and making lane change. When accelerating/decelerating, a target speed (ranging from 0 - 41m/s) is set for the NPC. The vehicle will try to reach the target speed based on the kinematics of the vehicle. These maneuvers are implemented and supported by LGSVL simulator APIs [17].

B. Domain Knowledge Module (❸ in Fig. 1)

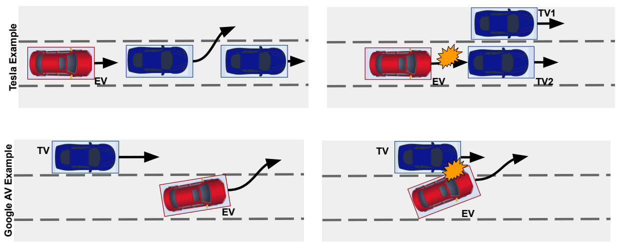


Figure 2: Real-world examples of violating safety constraints which lead to crashes (EV = ego vehicle or the AV under test; TV = target vehicle or NPC)

AV-Fuzzer leverages domain knowledge to perform an informed exploration to search for possible scenarios that are likely to lead to AV safety violations. It does so by combining vehicle kinetics and a safety model with a defined fitness function to guide the scenario-generating process.

1) Safety Model: AV-Fuzzer allows users to define a set of *safety constraints* in their tests based on the design requirements of the AV, the operational design domain (ODD), and traffic laws (which depend on geographical location). Violating the condition(s) of the safety constraints indicates a *safety violation*. One of the requirements for AVs is that they follow traffic regulations and do not cause at-fault accidents. Therefore, we define two common safety constraints that an AV must follow in order not to cause any at-fault accidents on public roads. We use one real-world example for each to illustrate the safety constraints we define in our safety model. These are shown in Fig. 2.

SC-1: The first example corresponds to a problem with Tesla Autopilot: the Tesla fails to register the vehicle in front of it and accelerates; the Tesla rear-ends the vehicle, leading to a fatal accident [18]. Based on traffic regulations, Tesla has significant liability in the rear-end collision, since it failed to maintain a safe distance from the leading vehicle. We define *maintaining a safe distance from the leading vehicle* as one of the safety constraints, SC-1. Hence, failure to do so and thus to cause an accident results in safety violations.

SC-2: The second example (shown at the bottom of Fig. 2) illustrates an accident in which a Google AV failed to yield to a vehicle (TV) approaching from behind in the adjacent lane, resulting in a side collision [19]. In this case, the TV had the right of way, and the Google AV should have yielded to

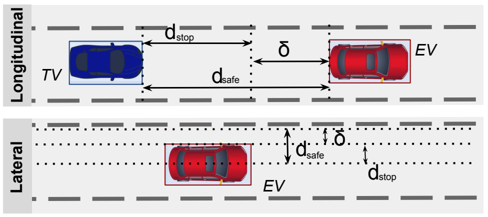


Figure 3: Vehicle kinetic safety model. Definition of d_{stop}, d_{safe}, and δ for lateral and longitudinal movement of the car.

it. Hence, the Google AV has significant liability. We define *yielding to a vehicle that has the right of way* as the other safety constraint, SC-2. Failure to do so and thus causing an accident will result in safety violations.

Recall that our goal is to determine the NPC trajectories in a scenario in which the AV will run into the safety violations. There can be many possible such NPC trajectories that can lead an AV to violate the safety constraints (either SC-1 or SC-2)—these trajectories are what AV-FUZZER aims to find.

2) Vehicle Kinetics: We define a vehicle kinetic model that connects vehicle dynamics with the safety model. This kinetic model is applied to the conditions of violating the safety constraints in a collision to quantify *safety potential*. In the vehicle kinetic model, we define the instantaneous safety criteria of a vehicle in terms of the longitudinal (i.e., in the direction of the vehicle's motion) and lateral (i.e., perpendicular to the direction of the vehicle's motion) Cartesian distance travelled by the AV (see Fig. 3). We define δ, the safety potential, in (1).

$$\delta = d_{\text{safe}} - d_{\text{stop}} \tag{1}$$

In the equation, d_{safe} [20], [21] of a vehicle is defined as the maximum distance the vehicle can travel without colliding with any static or dynamic object. d_{stop} is defined as the distance the vehicle will travel before coming to a complete stop while the maximum comfortable deceleration a_{max} is being applied. Similar models have been proposed in [13].

A vehicle is defined to be in a safe state if $\delta > 0$ in both the lateral and longitudinal directions with respect to another vehicle.[2] The safety potential, δ, is measured and monitored globally in a scenario for every vehicle in the driving environment with respect to the conditions of safety violations. In general, the smaller δ is, the larger the risk of leading to a safety violation at that moment, as the vehicle needs more space than is available to avoid the collision.

C. *Fuzzing Engine (❹ in Fig. 1)*

Fig. 4 illustrates the workflow of the fuzzing engine. At a high level, our *fuzzing engine* starts with GA (① in the figure), which is guided based on feedback from a fitness function. We define the fitness function in terms of the safety potential δ, which is measured using the actual AV self-driving performance and NPC trajectories in a scenario. The GA starts with a set of scenarios with randomly generated NPC maneuvers in the population. It then introduces changes to the NPC maneuvers via recombination operations. The GA keeps the scenarios that have lower safety potential and eliminates the ones with high safety potential, based on the fitness scores of the scenarios. This process is repeated until a scenario with lower safety potential is found. In that way, the GA performs a directed exploration in the NPC maneuver space to narrow the search and move towards safety violation scenarios. Once the GA finds a scenario with a low safety potential, we use the scenario as a *seed scenario* for exploitation (② in Figure 4). Note that this exploration process can also be guided by other optimization algorithms; however, we choose GA since it provides better efficiency in optimizing trajectories in high-fidelity simulations. We will further explain this in Section IV-B and V-A.

We designed a *local fuzzer* (④ in Figure 4) that takes the seed scenario and iteratively exploits its close variants to identify safety violations. The local fuzzer is triggered as a subroutine of the GA when a scenario with low safety potential is observed. Our intuition is that there might exist a safety violation around a near-miss case, hence we exploit the local optima of it before the search is directed to a different target area. In this step, the GA main process is suspended, and the local fuzzer takes over and searches the NPC maneuvers around the seed for safety violations given a period of search time. The GA main process is resumed at the end of the local fuzzer's execution. Any safety violation scenarios found during the process are reported.

We monitor whether the fitness of the generated scenarios is improved over time in the GA. If it is not, we conclude that the GA is stuck at a local optimum. We launch a *Random Restart* process if that happens (shown in ⑤ in Fig. 4), such that a new set of scenarios are initialized and the GA can restart. We do this by randomly generating a large set of NPC maneuvers for the new set of scenarios and choosing the ones that are most different from those in past scenarios that the GA generated. The restart mechanism ensures that (1) the GA gets out of the stuck point when the search direction is likely a dead end, and (2) distinct safety violation scenarios can be found as a result of the restart in an uncharted area in the search space. These components synergistically work together and dynamically balance exploration versus exploitation in the search.

D. *Putting* AV-FUZZER *into Perspective*

The fuzzing engine features informed searching according to the feedback from introducing a new NPC maneuver in a scenario. The feedback based on the fitness function is provided by monitoring the states of vehicles in the environment, and it applies domain knowledge when computing the safety potential. For illustration, we use one of the safety violation scenarios found by AV-FUZZER as the running example in Fig. 5.

Fig. 5 (a) shows a TV that tries to overtake the AV, leaving a little space in front of the AV. If the speed of the TV increases, a rear-end collision is less likely (shown in Fig. 5(b)). We will observe an increasing δ value. However, if the speed of the TV decreases, δ will decrease, leaving the AV with less time and space in which to respond and avoid the collision (shown in Fig. 5(c)). Ideally, if an AV is defect-free, it should be able to handle these changing behaviors of the TV and avoid causing accidents given the safety constraints (Section III-B1).

[2]We use the shorthand $\delta > 0$ to mean both lateral and longitudinal δs.

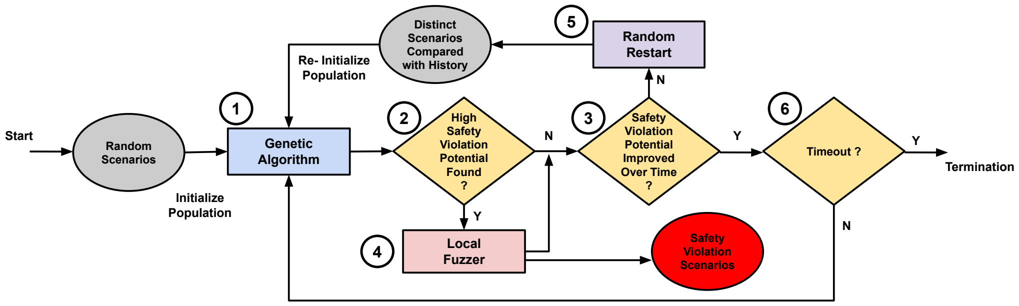

Start
Random Scenarios
Initialize Population
1
Genetic Algorithm
Re- Initialize Population
Distinct Scenarios Compared with History
5
Random Restart
2
High Safety Violation Potential Found ?
N
Y
4
Local Fuzzer
3
Safety Violation Potential Improved Over Time ?
N
Y
Safety Violation Scenarios
6
Timeout ?
Y
N
Termination

Figure 4: Workflow of AI-driven fuzzing engine.

However, if the AV is defective, the latter case (with lower δ) will lead to a scenario where the AV is more likely to reveal a safety violation. The GA randomly mutates an NPC maneuver in every scenario. By monitoring δ in the scenarios, the GA is able to select scenarios that have lower safety potential (i.e., lower δ) as a result of the mutations and discard the ones with higher safety potential. In our example, scenarios similar to that in Fig. 5(b) are likely to be eliminated, whereas scenarios that are similar to that in Fig. 5 (a) are carried forward in the GA evolution. In this way, the surviving scenarios will contain the NPC maneuvers that are more likely to lead to AV safety violations.

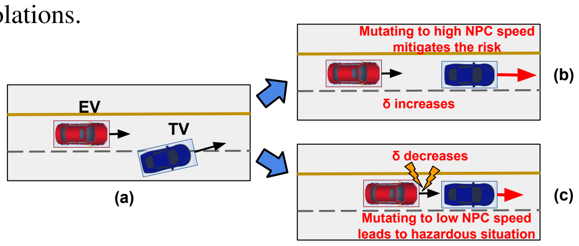

EV
TV
(a)
Mutating to high NPC speed mitigates the risk
δ increases
(b)
δ decreases
Mutating to low NPC speed leads to hazardous situation
(c)

Figure 5: The GA searches for safety violation scenarios in NPC maneuver space based on the actual self-driving performance of the AV.

IV. Algorithm Design

This section presents the formalism for the fuzzing engine.

A. Modeling Vehicle Behaviors

Let m_t be a driving maneuver employed by an agent (either NPC or AV) at time t. We use $m_t^{NPC,i}$ to denote the maneuver of i^{th} NPC and m_t^{AV} for the AV. A sequence of maneuvers performed by the i^{th} NPC until time t is denoted by $\phi_t^{NPC,i}$, and by the AV as ϕ_t^{AV}. We use $\phi_t^{NPC} = \{\phi_t^{NPC,i} : \forall i\}$ to represent the set of all NPC maneuvers until time t.

We define the *state* of an i^{th} agent at time t as $\mathbf{S}_t^i$. It consists of the agent's position ($\vec{x}$), velocity ($\vec{v}$), and acceleration ($\vec{a}$). We use $\mathbf{S}_t^{NPC,i}$ to denote the state of the i^{th} NPC and $\mathbf{S}_t^{AV}$ to denote the state of the AV. The states of all agents at time-step t are denoted by $\mathbf{S}_t$. In our study, the states are observed from the simulator.

We use *sim output* to denote $\mathbf{O}_t$ at time-step t. It is 3D frames generated by the simulator's rendering engine (e.g., Unreal Engine). The AV reads sim output using sensors (e.g., camera images, LiDAR point-cloud, RADAR data, and GPS location) as input $\mathbf{I_t}$.

Recall that a *driving environment* is characterized by the initial state of the NPCs and the AV, and the road type (i.e., urban vs. freeway). Examples of driving environments are shown in Fig. 7. A *scenario*, denoted by $\boldsymbol{\mathcal{D}}$, is characterized by the set of all maneuver sequences of NPCs (i.e., $\boldsymbol{\mathcal{D}} = \{\phi_t^{NPC,i} : \forall i\}$).

The maneuvers of the AV are dependent on the state of the AV at time t and on the previous maneuver at time $t-1$ (shown in (2)). They are chosen by the planning module in the ADS of the AV and denoted by $\mathcal{P}_{ev}$. The control module in the ADS takes these maneuvers and provides actuation commands (e.g., steer, throttle, brake), denoted by $\mathbf{A_t^{AV}}$. The actuation commands drive the AV in the simulator.

$$m_t^{AV} = \mathcal{P}_{AV}(\mathbf{S_t}^{NPC,i}, m_{t-1}^{AV}) \tag{2}$$

The AV uses ($\mathbf{I}_{t-k:t}^{AV}$), which senses data from multiple cameras, LiDAR, RADAR, GPS, and IMU, to determine its own state as well as the states of all the NPCs. This is shown in (3). Here, $\mathcal{L}_{av}$ corresponds to the AV's sensor layer, perception module, and prediction module. $\mathbf{S_t}$ is what our fuzzing engine observes and uses to compute safety potential in the fitness function.

$$\mathbf{S_t} = \mathcal{L}_{av}(\mathbf{I}_{t-k:t}^{AV}) \tag{3}$$

The maneuver of the i^{th} NPC at time t is dependent on its own state as well as the previous maneuver at time $t-1$, as that is shown in (4). Here $\mathcal{P}_{sim}$ is the planner of the simulator that issues maneuvers for NPCs. The controller takes those maneuvers and provides actuation commands, denoted by $\mathbf{A_t^{NPC,i}}$, to each NPC. The actuation commands drive the NPCs in the simulator. The NPC maneuvers are what our fuzzing engine perturbs via simulator API.

$$m_t^{NPC,i} = \mathcal{P}_{sim}(\mathbf{S_t}^{NPC,i}, m_{t-1}^{NPC,i}) \tag{4}$$

The state of each agent is readily available to the simulator through its direct access. The states of all the agents and actuation values in the simulator generate 3D frame $\mathbf{O}_t$, forming the observations of the AV that transform to $\mathbf{I}_{t-k:t}^{AV}$.

$$\mathbf{O}_{t+1} = \mathcal{R}_{sim}(\mathbf{S_t}, \mathbf{A}_t) \tag{5}$$

B. Genetic Algorithm

Why GA? There are various optimization algorithms that can be chosen to guide the search. We choose GA for the following reasons: (1) We empirically observed that GA can be more efficient than model-based techniques, such as reinforcement learning (RL), in optimizing NPC trajectories in a high-fidelity simulation. This is because the state space of trajectories and vehicle behaviors is huge in a high-fidelity simulation. Observing a repeated one, necessary for RL to be effective, requires RL to accumulate a huge set of historical data, which can be very time-consuming. In contrast, GA guides the search by trial and error without relying on having a huge set of historical data. (2) There are inherently many local optima in

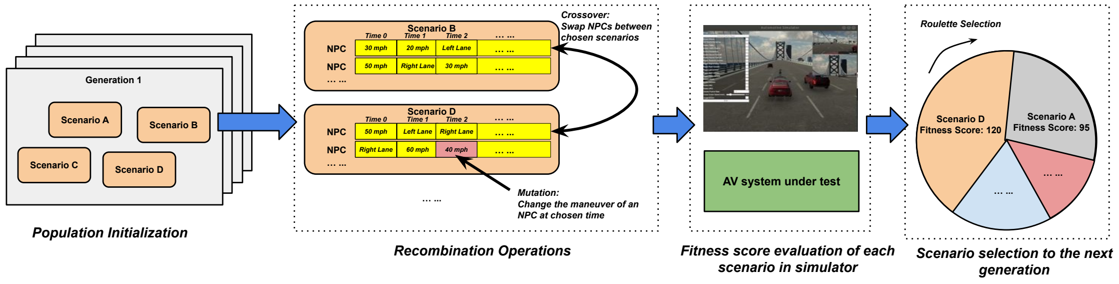

Generation 1
Scenario A
Scenario B
Scenario C
Scenario D
Population Initialization
Scenario B
Time 0
Time 1
Time 2
NPC
30 mph
20 mph
Left Lane
50 mph
Right Lane
30 mph
Scenario D
50 mph
Left Lane
Right Lane
Right Lane
60 mph
40 mph
Crossover: Swap NPCs between chosen scenarios
Mutation: Change the maneuver of an NPC at chosen time
Recombination Operations
AV system under test
Fitness score evaluation of each scenario in simulator
Roulette Selection
Scenario D Fitness Score: 120
Scenario A Fitness Score: 95
Scenario selection to the next generation

Figure 6: A high-level overview of the genetic algorithm design.

the space, each of which may correspond to a unique case of safety violation. We aim at finding as many safety cases as possible, and GA is found to be efficient in promoting solution diversity [22]. (3) Other methods, such as Bayesian linear regression [23], Bayesian optimization [6], and variational inference [24], are promising but can be either data-inefficient or difficult to scale to a high-dimensional state space. We evaluate the performance of AV-FUZZER and compare it with other techniques in Section V.

The high-level workflow of the genetic algorithm is shown in Fig. 6 and will be described next.

1) Initialization: We begin with a set of randomly initialized scenarios as our initial population. Recall that a scenario is characterized by a sequence of all NPC maneuvers (i.e., $\{\phi_t^{NPC,i} : \forall i\}$). In the GA, a scenario corresponds to a chromosome, and a maneuver $m_t^{NPC,i}$ for the i^{th} NPC corresponds to a gene. We ensure that at time-step $t = 0$, all agents are safe, i.e., $\delta_{t=0} > 0$.

2) Fitness Function: In the GA, the fitness score of a candidate is used to determine whether the candidate should be carried forward or eliminated. There are a few proposals for designing fitness functions in the area of testing autonomous systems [25]. For our testing purposes, We choose to design our fitness function based on domain knowledge and vehicle safety. Our fitness function gives a fitness score to each scenario based on the AV's and NPCs' performance. Since we are looking for AV safety violations that lead to crashes, we define our fitness function with the objective of decreasing safety potential (δ). δ can be decreased by either decreasing d_{safe} or increasing d_{stop} (or both) as defined in (1). AV-FUZZER achieves this by choosing NPC maneuvers (ϕ_t^{NPC}) that can lead to such conditions, shown in Section III-D. As shown in (6), our fitness function optimizes decreasing of δ.

$$\mathcal{F}(\mathcal{P}_{AV}, \phi_t^{NPC}) = \min\{\delta_t : \forall t\} \tag{6}$$

In our simulation experiment, we measured the fitness score of a scenario four times per second[3]. A smaller δ_t measured at a time-step t indicates a higher risk of safety violation at time t. The risk can be mitigated or increased in the next time-step $(t+1)$, depending on the dynamics of the environment.

We are interested in solving the following optimization problem in the GA:

$$\phi^{NPC^*} = \underset{\phi^{NPC}}{\operatorname{argmin}}\, \mathcal{F}(\mathcal{P}_{AV}, \phi^{NPC}) \tag{7}$$

where ϕ^{NPC^*} are the NPC maneuvers that may cause the AV to have safety violations. The fitness score is used to rank each candidate scenario, and the GA decides whether the candidate scenario should be considered for the next evaluation round.

[3]This is to balance experiment time and measurement granularity, as each measurement of the fitness score requires the simulation to pause, which slows down the entire simulation process.

3) Recombination Operations: The recombination operations consist of mutation and crossover. The high-level ideas of the two operations are shown in Figure 6.

Mutation: Given a set of $m_t^{NPC,i}$ in a scenario, the mutation operator randomly chooses one of them and randomly changes it with a maneuver from the allowed NPC maneuvers (Section III-A) with a probability of r_m.

Crossover: We designed the crossover operation as a swap operation, which increases the chances of combining the NPCs in two scenarios to form a new scenario with a higher chance of AV safety violations. In every generation, the swap operation randomly selects two NPCs in two scenarios, one from each, and swaps the two NPCs with a probability of r_c.

We experimented with r_m and r_c values in the range that the GA literature recommends [16] and chose 0.3 and 0.4 for r_m and r_c, respectively; they result in the shortest time to arrive at safety violation scenarios in our case.

4) Roulette Selection: The goal of the selection process is to eliminate unfit candidates from the population. We do this by integrating a roulette selection process that selects candidates in proportion to their fitness scores. If a scenario has a higher fitness score, it will be selected more often. To do this, we "size" the candidates according to their fitness scores. Let T be the sum fitness score of all the scenarios in a generation. A random number from 0 to T falls within the range of some scenario, which is then selected. With fitness-proportionate selection. There is a chance that some scenarios with lower scores may survive the selection process. The reason for this is that the probability that the weaker scenarios will survive is low, but not zero, meaning it is still possible they will be selected. That is an advantage because there is a chance that even weak scenarios may have some features or characteristics that could prove useful following the recombination operations [26].

C. Local Fuzzer

The intention of *local fuzzer* is to dynamically increase the exploitation of the surrounding areas when any scenario with anhigh potential for safety violations is discovered. Such high-potential cases may represent near-miss cases of safety violations. We use them as *seed scenarios*. The local fuzzer process focuses on exploitation that mines local optimum in the neighborhood based on those seed scenarios collected by the GA. Once a scenario (say c_{high}) with a high fitness score is found by the GA after a few initial stages of evolution, the scenario will be used as a seed scenario. A subroutine of a local fuzzer will be called while the evolution process of the

GA is paused. In the local fuzzer, a new population of c_{high} is initialized with the seed scenario, and a mutation-based fuzzing process is performed based on the new population. To maintain a high level of safety violation potential during the local fuzzing, we stick with the fitness function as in the GA, but with a doubled mutation rate, based on the seed scenario as the entire population of the local fuzzer. At the end of local fuzzing, c_{high} in the GA will be replaced by a scenario with a higher fitness score found during the local fuzzing (if there is one). This ensures an opportunity for a good gene of a better scenario to be introduced into the GA population. After the local fuzzing, the subroutine ends, and the main process of the GA is resumed.

D. Random Restart

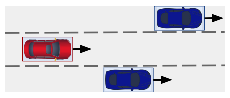
In addition to the local fuzzer, we employ a random restart (RR) mechanism in AV-FUZZER to promote solution diversity. The RR will be activated on demand, dynamically navigating the exploration to an uncharted area in the search space when the GA gets stuck. If we observe that the fitness score does not improve over time, we force an RR. We monitor the fitness improvement by comparing the current fitness score with the average score over the last five generations. We keep the full history of NPC maneuvers explored in every past scenario and use them to generate new initial populations for RR. When forcing an RR, we create 1,000 candidate scenarios initialized with random NPC maneuvers and then select the most-different ones for the new population of RR. The similarity comparison is done by computing the *Euclidean distance* between the NPC trajectories projected by the initialized maneuvers in the candidate scenarios and those in every past scenario, hence we do not require running simulations. The Euclidean distance is calculated based on the locations sampled every second along each of the trajectories.

E. Termination

We repeat the above steps of the GA, local fuzzer, and RR until a given time budget is exhausted. AV-FUZZER then returns all the scenarios that resulted in AV safety violations during the evolution. It is possible that no such solution scenarios have been found. In that case, AV-FUZZER reports empty. Note that there could be many reasons why AV-FUZZER might be unable to find any safety violation scenarios. For example, the ADS under test may be robust against violating the defined safety constraints, or the search time allowed for AV-FUZZER might have been too short.

V. RESULTS

In this section, we demonstrate the efficiency and effectiveness of AV-FUZZER in terms of the time needed to find safety violations and their diversities. We then analyze the safety violations found in Apollo and discuss their underlying causes. Our experiments were conducted on a Ubuntu PC with 256GB memory, an Intel Xeon CPU, and an NVIDIA GTX1080 TI. Apollo 3.5 and LGSVL 2019.05 were used in our experiments.


(a) DS-1: Freeway

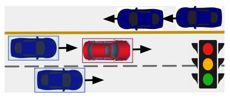
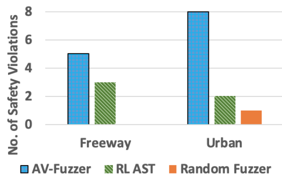

(b) DS-2: Urban

Figure 7: Driving environments supported by simulator.

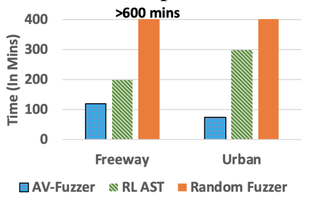
A. Efficiency

We evaluate the efficiency of comparing it against two baselines with respect to (1) the total number of safety violations found and (2) the mean time to the occurrence of a safety violation, given a fixed search time. For our first baseline, we build a random fuzzer that generates random NPC maneuvers in every scenario. For the second baseline, we choose a state-of-the-art adaptive stress testing (AST) technique powered by reinforcement learning (RL) [4]. RL is a popular machine-learning algorithm based on Markov decision process. We establish the baseline by adopting the code of an RL AST technique in [4]. Similar AST techniques are also used in [5], [7]. We connect the two baselines to our simulation infrastructure by replacing the fuzzing engine in the framework (Figure 1) with either a random fuzzer or an RL AST.

Fig. 7 illustrates two driving environments (freeway and urban) that are supported by the simulation engine and are used in our evaluation. In each environment, the AV and NPCs are placed on the road, separated by some distance. The blue vehicles with bounding boxes are the NPCs whose maneuvers are subjected to change. The driving environments are common driving cases encountered by drivers on a daily basis. The vehicles are initialized with zero speed and in a safe state. The destination of the AV is set to the far end of the other side of the road. A simulation in each driving environment takes an average of 3.6 minutes to evaluate in our setup.



(a) No. of safety violations



(b) Meantime to safety violations

Figure 8: Efficiency of AV-FUZZER.

Fig. 8a shows the numbers of safety violations AV-FUZZER, RL AST, and the random fuzzer found in DS-1 and DS-2, given a 10-hour simulation time budget for each technique in each of DS-1 and DS-2. As can be seen, no safety violations are found in DS-1 by the random fuzzer, and 3 are found by RL AST. In contrast, AV-FUZZER finds 5 safety violations. In DS-2, AV-FUZZER finds 8 safety violations, whereas RL AST and random fuzzing find only 2 and 1, respectively.

Fig. 8b illustrates mean time to safety violation found by AV-FUZZER, RL AST, and the random fuzzer in DS-1 and DS-2. During the search, AV-FUZZER generates a total of 5 and 8 seeds in DS-1 and DS-2, respectively. The local fuzzer runs for 5 generations based on each seed. The mean time to safety violations is calculated by dividing total simulation time by the number of the safety violations found by a technique. These are 120 mins and 75 mins for AV-FUZZER in DS-1 and

DS-2, respectively. They are about 300 mins and 200 mins for RL AST, and at least 600 minutes for the random fuzzer. Moreover, it took 177 mins for AV-FUZZER to find the first safety violation in DS-1, and 162 mins in DS-2. In RL AST, it takes 296 mins and 224 mins to find the first one in DS-1 and DS-2, respectively. The random fuzzer finds only 1 safety violation in DS-2 across the entire evaluation.

Overall, our evaluation shows that AV-FUZZER is able to find more safety violations given the same period of search time, hence it is more efficient than either RL AST or random fuzzing. The reasons that AV-FUZZER outperforms both RL AST and random fuzzing are as follows: (1) Our local fuzzer (Section IV-C) in AV-FUZZER reduces the time needed to find different local optima by dynamically increasing exploitation when likely safety violation cases are nearby. As we further measured, the mean time to safety violations is reduced by about 1.8x on average if we remove the local fuzzer in AV-FUZZER. (2) The restart mechanism (Section IV-D) helps AV-FUZZER continue when it gets stuck, which saves time during the search. (3) GA, which is a meta-heuristic search algorithm, provides a quicker solution than model-based RL in optimizing NPC trajectories. This is because an NPC trajectory consists of a dense set of continuous values in a large space. Observing a repeated trajectory, which is necessary for RL to be effective, requires accumulating a huge set of historical data for the training, which can be very time-consuming. One way to mitigate this is to simplify an NPC trajectory by specifying fewer discrete way-points to approximate where a vehicle should move. Similar approaches are used in [6], [7] for simpler environments. However, in our real-time, high-fidelity simulation, such simplification of vehicle dynamics significantly distorts the experiments, making the driving scenarios and physics unrealistic. Hence, the cases found in such setups are no longer representative of the real world and provide little insight into the testing of an industry-grade AV such as Apollo.

B. Safety Violation Scenarios

We review all the safety violation scenarios observed in our experiments and classify similar ones into groups based on their underlying causes. As a result, we categorize all the safety violation scenarios into 5 distinct types based on vehicle trajectories and possible design deficiencies in Apollo. After obtaining the result described above, we continue running the test for another 90 hours for each of DS-1 and DS-2. AV-FUZZER ends up with another 46 safety violation scenarios as the result of the search, thereby reaching a total of 59 during the 200-hour search). *We observe that all the safety violation scenarios, including the additional ones, are variants of the 5 types. Moreover, these 5 types are all revealed in the 13 safety violations found in the first 20-hour search by* AV-FUZZER, *indicating a saturation of the search.* Note that RL AST and the random fuzzer respectively only discover 2 types (Type 1 and 3) and 1 type (Type 1) among the 5 during the entire 200-hour search. This shows that AV-FUZZER can discover more diverse safety violation scenarios in a shorter period of time (20 hours in our simulation experiments), many of which cannot be found by existing techniques even given 10x the search time (200 hours). We show one example of each type in Fig. 10.

Fig. 9 shows the Euclidean distances (Section IV-D) between NPC trajectories across the 5 examples. The larger the distance, the more different the trajectories. As shown, the types have quite different NPC trajectories, with a minimum of 25.03 meters between example 2 and example 4, and a maximum of 85.76 meters between example 2 and example 5. The average distance between scenarios is 63.65 meters. The size of the vehicle in our simulation is about 4.3 meters in length.

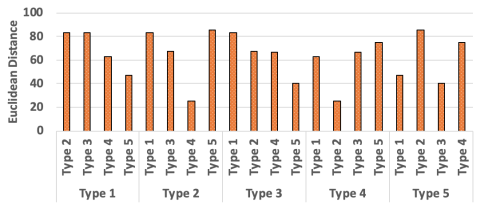


Figure 9: Euclidean distances (in meters) between NPC trajectories in safety violation scenarios.

Example 1 in Fig. 10 is a scenario in which the AV rear-ends an NPC that is trying to overtake the AV. From the AV's perspective, the NPC is detected, moving at high speed, as early as the time the NPC came to be behind the AV. However, the AV is not able to predict the NPC's intention to cut in front of it, and hence does not take preemptive measures to slow down and yield to the NPC. The NPC then starts merging into the AV's lane, but the AV is moving too fast to maintain at a safe distance from the NPC, leading to the rear-end collision. Usually, this type of accident can be avoided by humans drivers, as humans can identify intentions based on early aspects of the overtaking. Those early hints enable human drivers to apply brakes earlier and likely avoid the mishap. One way to mitigate this problem in AVs is to identify such NPC maneuvers based on their trajectories and reduce speed accordingly to cooperate in the merge-in maneuver of the NPC, just as human drivers would commonly do. Unfortunately, Apollo is not equipped with such safety measures.

We observe that a similar accident was recently reported by Pony.AI in Fremont, California, to the California DMV [12]. As stated in the report, a Tesla Model 3 tried to overtake the AV that Pony.Ai is test driving and to merge into the AV's lane. Unfortunately, the AV failed to identify the intention of the Tesla and caused a rear-end collision. This indicates that the kinds of deficiencies AV-FUZZER found do exist in real-world AV systems and traffic environments.

Example 2 in Fig. 10 shows that the AV tries to overtake the slow NPCs (TV1 and TV2) in front of it. The routine is planned as a curvy trajectory that includes both TV1 and TV2. As TV1 accelerates, the AV still tries to include TV1 in its trajectory, so the curvy trajectory becomes very long. Consequently, the AV cruises between two lanes along with the trajectory regardless of the state of TV2. Finally, the AV fails to yield to TV2, which has the right of way, and collides with it. The time-evolving simulation views of this scenario are shown in Fig. 11. As seen, TV1 and TV2 were detected by the

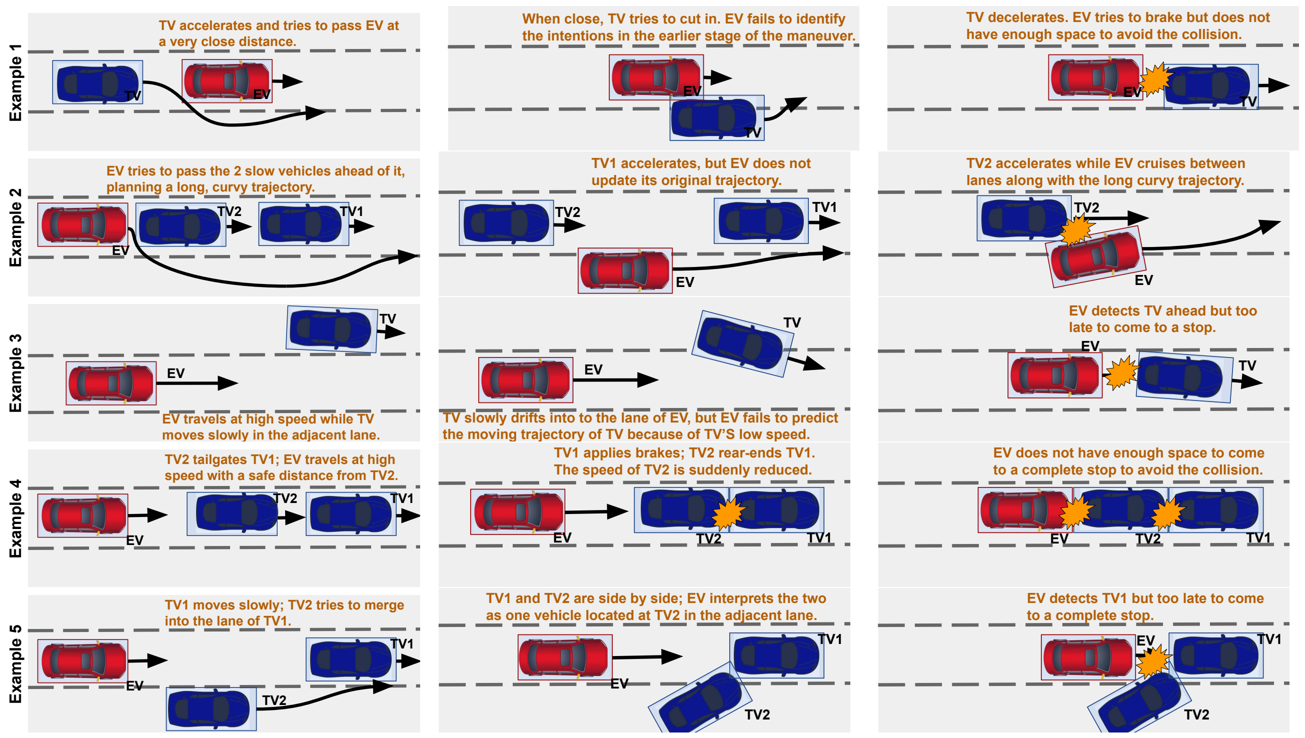
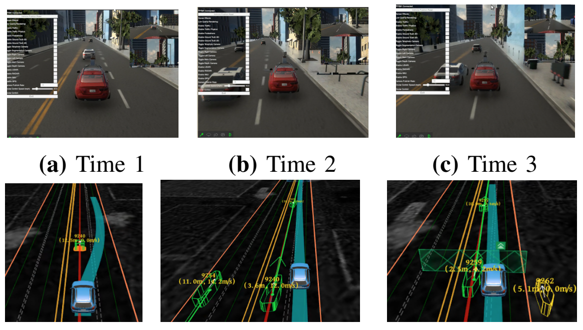

Example 1
TV accelerates and tries to pass EV at a very close distance.
TV
EV
When close, TV tries to cut in. EV fails to identify the intentions in the earlier stage of the maneuver.
EV
TV
TV decelerates. EV tries to brake but does not have enough space to avoid the collision.
EV
TV
Example 2
EV tries to pass the 2 slow vehicles ahead of it, planning a long, curvy trajectory.
EV
TV2
TV1
TV1 accelerates, but EV does not update its original trajectory.
TV2
TV1
EV
TV2 accelerates while EV cruises between lanes along with the long curvy trajectory.
TV2
EV
Example 3
TV
EV
EV travels at high speed while TV moves slowly in the adjacent lane.
TV
EV
TV slowly drifts into to the lane of EV, but EV fails to predict the moving trajectory of TV because of TV'S low speed.
EV detects TV ahead but too late to come to a stop.
EV
TV
Example 4
TV2 tailgates TV1; EV travels at high speed with a safe distance from TV2.
EV
TV2
TV1
TV1 applies brakes; TV2 rear-ends TV1. The speed of TV2 is suddenly reduced.
EV
TV2
TV1
EV does not have enough space to come to a complete stop to avoid the collision.
EV
TV2
TV1
Example 5
TV1 moves slowly; TV2 tries to merge into the lane of TV1.
EV
TV1
TV2
TV1 and TV2 are side by side; EV interprets the two as one vehicle located at TV2 in the adjacent lane.
TV1
EV
TV2
EV detects TV1 but too late to come to a complete stop.
EV
TV1
TV2

Figure 10: Examples of safety violation scenarios.



(a) Time 1 **(b)** Time 2 **(c)** Time 3

(d) Time 1 **(e)** Time 2 **(f)** Time 3

Figure 11: Example 2: Time-evolving simulation view (Top is simulation view, bottom is Apollo view).

AV (seen from Apollo's view), but the planning module of the AV did not plan a new action to accommodate the moving TV1 and TV2 during the overtaking; hence, the AV kept cruising between lanes. This scenario reveals the design deficiency of Apollo in this situation.

Example 3 in Fig. 10 shows a generic case of an accident that Apollo usually fails to avoid. While the AV is cruising at high speed, if an NPC cuts in at very low speed from the adjacent lane, it may lead to a rear-end collision. The reason can be a combination of several possible problems in Apollo. (1) We observe that in such scenarios, Apollo usually mispredicts the future trajectory of the NPC, failing to identify the intention to overtake. The NPC trajectory prediction is done by machine-learning-based models in the *Prediction Module* (Section II-A). Fig. 12 shows the moment before the accident happens. As seen, the AV is not able to predict the future trajectory (which should be shown as a long tail) of the NPC, and hence fails to take preemptive action to decelerate to avoid the collision. We notice that in this case, the NPC is predicted by Apollo to have a speed of 0 m/s by Apollo at that moment, whereas the actual NPC speed was 4.56 m/s, as measured from the simulator. (2) We also observe that Apollo's planner can be wrong even when the NPC trajectory is predicted correctly. Fig. 12 (b) shows this case. As seen, Apollo's planner issues an overtaking action (shown in a blue fence) that crosses the NPC's estimated trajectory. Therefore, the AV starts accelerating, leading to an accident.

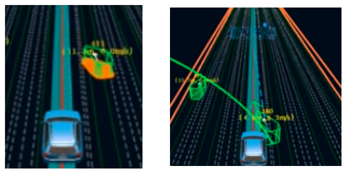
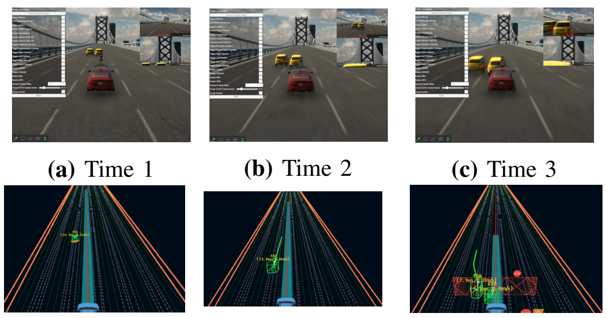


Figure 12: Example 3: (a)Failure to predict NPC trajectory; (b)The planner issues an overtaking action when it should not

Example 4 illustrates a cascade accident. As shown in Fig. 10, Apollo is following TV2. TV2 starts accelerating, and so does Apollo. At some point, TV2 rear-ends TV1, which is stationary. As a result, the speed of TV2 is suddenly reduced to zero, leaving Apollo little time to react. Apollo starts applying the brake when it is only 7.3 m from TV2, and the speed of Apollo is 41 km/h. Based on our profiling, Apollo would need at least 8.75 m to come to a complete stop, so a rear-end collision occurs. Similar problems have been revealed in Tesla Autopilot [27]. One way to mitigate them is to steer the AV to the adjacent lane (if available) to avoid the accident.

Example 5 in Fig. 10 illustrates a case in which an AV interprets two vehicles that are side-by-side as one vehicle. Fig 13 shows time-evolving simulation views of this scenario. As seen, two NPCs are moving ahead at a very short distance. The AV sees the two NPCs as just one, which it thinks is located in the adjacent lane. Consequently, the AV continues to accelerate until it is too late to prevent a collision. This scenario occurs because of problems in the *Perception module*


(a) Time 1 **(b)** Time 2 **(c)** Time 3

(d) Time 1 **(e)** Time 2 **(f)** Time 3

Figure 13: Example 5: Time-evolving simulation view (Top is simulation view, bottom is Apollo view).

of Apollo. Improving the accuracy of DNN models may help prevent similar accidents.

Recall that the safety violations AV-FUZZER reports based on the safety constraints (Section III-B1) are all at-fault accidents where the AV breaks traffic rules and is liable. As can be seen in Figure 10, these accidents are caused by either the wrong actions taken by the AV or a lack of capability in the AV, as a result of ADS deficiencies. We review all these accidents and find that they are indeed avoidable by taking precautionary actions (i.e., brake or switch to emergency lanes) or by improving the perception module.

VI. Related Work

A. AV Software Testing

Many existing studies on assessing AV robustness are geared toward testing and improving certain components (i.e., AI modules, etc.) in ADSs [2], [28]–[35]. Fremont et al. [2] propose a probabilistic programming language for the design and analysis of deep-learning-based perception modules in AVs. Pei et al. [29] propose an automated whitebox testing method for safety-critical, deep-learning components, based on neuron coverage and cross-referencing oracles. Calo et al. [36] propose sequential and combined approaches to search for alternative configurations that avoid accidents for the path-planning module of AVs. While these studies are useful in improving certain AV modules, an end-to-end ADS consists of many safety-critical components that work together synergistically. Therefore, it is critically important to assess ADS software end-to-end, as we do in this work, to understand the deficiencies of the system.

Some recent studies focus on bridging conventional testing techniques to AVs [4], [5], [7], [35], [37]–[39]. Corso et al. [7] leverage adaptive stress-testing techniques to find both vulnerabilities and failures in AVs. They perturb the trajectories of traffic participants and inject noise into the input sensor data of AVs to cause accidents. Hauer et al. [25] design fitness functions for optimization algorithms in AV testing. Klück et al. [39] investigate a search-based optimization algorithm for testing ADAS. Abdessalem et al. [34] use a learnable evolutionary algorithm to stress test vision-based control systems. While these studies provide useful insights into AV testing, they do not focus on improving the efficiency and comprehensiveness of the testing. Further, their evaluations are done on much simpler AV programs than an industrial-grade ADS with high-fidelity simulation; hence they provide limited insights into the deficiencies of real-world ADSs. In contrast, our work proposes a complete, yet practical and efficient testing framework for industry-grade AVs. We further report and analyze the safety-critical deficiencies discovered in Apollo.

B. AV Vulnerability and Fault Injection

Fault injection techniques have been used to assess the vulnerability of computer systems [5], [13], [40], [41]. Recently, Jha et al. [13] have demonstrated a fast fault injection framework for AVs that uses a Bayesian network to find the vulnerabilities of an ADS under transient faults. While the technique offers orders of magnitude speed-up in finding critical faults, the focus is on hardware-fault vulnerabilities of AVs rather than on intrinsic design deficiencies of the system.

There has been a large body of work in the area of AV security [29], [42]–[44]. Most of it focuses on finding adversarial examples to mislead AV perception systems. Boloor et al. [42] demonstrate vulnerabilities by adding scratch marks on roads and tricking AVs to steer out of lanes. Chernikova et al. [43] show that a carefully designed minor modification of camera images can lead to misclassification of DNNs to the classes of the attackers' choice. While these studies highlight the safety concerns of AVs, they focus on the security vulnerabilities of AVs under attack rather than on finding intrinsic ADS software deficiencies or bugs.

C. AV Defensive Driving

Another research direction in AV safety is investigating how AVs can behave defensively in response to potential dangers. Zhan et al. [45] propose a unified planning framework under uncertainty in urban driving environments for AVs. Abeysirigoonawardenal et al. [6] use Bayesian optimization to find adversarial NPC behaviors that crash AVs. They look for collisions, regardless of whether they are avoidable or are infractions of the AV, to formulate defensive AV driving strategies. In our study, we look for design deficiencies of the ADS with respect to safety violations.

VII. Conclusion

We propose AV-FUZZER, an automated fuzzing framework that can generate AV safety violation scenarios. We demonstrate AV-FUZZER on an industrial-grade AV platform, Baidu Apollo, and find safety violation scenarios in a timely manner. These safety violations are traceable to design deficiencies in Apollo. We characterize the deficiencies and discuss their overarching causes.

VIII. Acknowledgement

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship, the National Science Foundation (NSF) under Grant No.1535070, 1547249, CNS 18-1667, Sandia National Lab contract No.1951381, and IBM PhD Fellowship. We thank NVIDIA Corporation for equipment donation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSERC, NSF, Sandia, NVIDIA or IBM.

References

[1] Carla. The carla autonomous driving challenge. [Online]. Available: https://carlachallenge.org/

[2] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2019, pp. 63–78.

[3] A. C. Madrigal, "Inside waymo's secret world for training self-driving cars," *The Atlantic*, vol. 23, 2017.

[4] R. Lee, O. J. Mengshoel, A. Saksena, R. Gardner, D. Genin, J. Silbermann, M. Owen, and M. J. Kochenderfer, "Adaptive stress testing: Finding failure events with reinforcement learning," *arXiv preprint arXiv:1811.02188*, 2018.

[5] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive stress testing for autonomous vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1–7.

[6] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek, "Generating adversarial driving scenarios in high-fidelity simulators," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[7] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validatio," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 163–168.

[8] F. Zhu, L. Ma, X. Xu, D. Guo, X. Cui, and Q. Kong, "Baidu apollo auto-calibration system-an industry-level data-driven and learning based vehicle longitude dynamic calibrating algorithm," *arXiv preprint arXiv:1808.10134*, 2018.

[9] Baidu. Baidu apollo github repository. [Online]. Available: https://github.com/ApolloAuto/apollo

[10] ——. Baidu apollo autonomous vehicle road test report. [Online]. Available: https://www.globenewswire.com/news-release/2019/04/03/1796503/0/en/Baidu-Apollo-Autonomous-Driving-Technological-Leadership-Recognized-by-China-s-First-Autonomous-Vehicle-Road-Test-Report.html

[11] M. F. Gasgoo. Baidu apollo given another 20 licenses by beijing for autonomous car road tests. [Online]. Available: http://autonews.gasgoo.com/china_news/70015513.html

[12] C. DMV. Testing of autonomous vehicles with a driver. [Online]. Available: https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/testing

[13] S. Jha, S. Banerjee, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2019, pp. 112–124.

[14] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal pfrocessing, sensor fusion, and target recognition VI*, vol. 3068. International Society for Optics and Photonics, 1997, pp. 182–194.

[15] N. Narayanamurthy, K. Pattabiraman, and M. Ripeanu, "Finding resilience-friendly compiler optimizations using meta-heuristic search techniques," in *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 2016, pp. 1–12.

[16] R. L. Haupt, "Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors," in *IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium. 2000 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (C*, vol. 2. IEEE, 2000, pp. 1034–1037.

[17] LG. Lgsvl simulator. [Online]. Available: https://www.lgsvlsimulator.com/

[18] S. Alvarez, "Research group demos why tesla autopilot could crash into a stationary vehicle," 2018.

[19] C. Ziegler, "A google self-driving car caused a crash for the first time," *The Verge*, 2016.

[20] S. M. Erlien, "Shared vehicle control using safe driving envelopes for obstacle avoidance and stability," Ph.D. dissertation, Stanford University, 2015.

[21] J. Suh, B. Kim, and K. Yi, "Design and evaluation of a driving mode decision algorithm for automated driving vehicle on a motorway," *IFAC-PapersOnLine*, vol. 49, no. 11, pp. 115–120, 2016.

[22] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[23] K. Azizzadenesheli, E. Brunskill, and A. Anandkumar, "Efficient exploration through bayesian deep q-networks," in *2018 Information Theory and Applications Workshop (ITA)*. IEEE, 2018, pp. 1–9.

[24] Z. C. Lipton, J. Gao, L. Li, X. Li, F. Ahmed, and L. Deng, "Efficient exploration for dialog policy learning with deep bbq networks & replay buffer spiking," *CoRR abs/1608.05081*, 2016.

[25] F. Hauer, A. Pretschner, and B. Holzmüller, "Fitness functions for testing automated and autonomous driving systems," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2019, pp. 69–84.

[26] M. Lones, "Sean luke: essentials of metaheuristics," 2011.

[27] S. Alvarez, "Research group demos why Tesla Autopilot could crash into a stationary vehicle," https://www.teslarati.com/tesla-research-group-autopilot-crash-demo/, June 2018.

[28] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1555–1562.

[29] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18.

[30] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *Proc. International Conf. for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 8:1–8:12.

[31] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "Binfi: an efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2019, p. 69.

[32] G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A configurable fault injector for tensorflow applications," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018, pp. 313–320.

[33] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 63–74.

[34] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 1016–1026.

[35] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 318–328.

[36] A. Calo, P. Arcaini, S. Ali, F. Hauer, and I. Fuyuki, "Generating avoidable collision scenarios for testing autonomous driving systems," in *IEEE Intl. Conf. on SW Testing, Verification and Validation*, 2020.

[37] F. Hauer, T. Schmidt, B. Holzmüller, and A. Pretschner, "Did we test all scenarios for automated and autonomous driving systems?" in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2950–2955.

[38] I. Majzik, O. Semeráth, C. Hajdu, K. Marussy, Z. Szatmári, Z. Micskei, A. Vörös, A. A. Babikian, and D. Varró, "Towards system-level testing with coverage guarantees for autonomous vehicles," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019, pp. 89–94.

[39] F. Klück, M. Zimmermann, F. Wotawa, and M. Nica, "Genetic algorithm-based test parameter optimization for adas system testing," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2019, pp. 418–425.

[40] G. Li and K. Pattabiraman, "Modeling input-dependent error propagation in programs," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2018.

[41] G. Li, K. Pattabiraman, C.-Y. Cher, and P. Bose, "Understanding error propagation in gpgpu applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 240–251.

[42] A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, "Attacking vision-based perception in end-to-end autonomous driving models," *arXiv preprint arXiv:1910.01907*, 2019.

[43] A. Chernikova, A. Oprea, C. Nita-Rotaru, and B. Kim, "Are self-driving cars secure? evasion attacks against deep neural networks for steering angle prediction," *arXiv preprint arXiv:1904.07370*, 2019.

[44] V. L. Thing and J. Wu, “Autonomous vehicle security: A taxonomy of attacks and defences,” in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2016, pp. 164–170.

[45] W. Zhan, C. Liu, C.-Y. Chan, and M. Tomizuka, “A non-conservatively defensive strategy for urban autonomous driving,” in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 459–464.