I. Abstract

In this work, we propose VioHawk, a novel simulation-based fuzzer that hunts for scenarios that imply ADS traffic violations. Our key idea is that, traffic law regulations can be formally modeled as hazardous/non-hazardous driving areas on the map at each timestamp during ADS simulation testing (e.g., when the traffic light is red, the intersection is marked as hazardous areas). Following this idea, VioHawk works by inducing the autonomous vehicle to drive into the law-specified hazardous areas with deterministic mutation operations. We evaluated the effectiveness of VioHawk in testing industry-grade ADS (i.e., Apollo). We constructed a benchmark dataset that contains 42 ADS violation scenarios against real-world traffic laws. Compared to existing tools, VioHawk can reproduce 3.1X-13.3X more violations within the same time budget, and save 1.6X-8.9X the reproduction time for those identified violations. Finally, with the help of VioHawk, we identified 9+8 previously unknown violations of real-world traffic laws on Apollo 7.0/8.0. The paper titled "VioHawk: Detecting Traffic Violations of Autonomous Driving Systems through Criticality-guided Simulation Testing" has been accepted by ISSTA'24.

II. Artifact Overview

The artifact contains the prototype of VioHawk, as well as the essential data for artifact evaluation. As clarified in §III, §IV, and §V, we request our artifact to be evaluated towards three badges, namely the **Available Badge**, the **Functional Badge**, and the **Reusable Badge**.

Prototype of VioHawk. The prototype included in the artifact is designed to conduct comprehensive fuzz testing on autonomous driving systems. The tool primarily encompasses two key techniques. One is **detecting violations** through a defined traffic rules oracle, and the other is **mutating scenarios** based on dangerous and drivable areas. These two key techniques are detailed as follows:

❶ The **Violation Detection** aims to evaluate the ability of VioHawk to detect traffic violations. Using scripts to run the detection code on given scenario data, we compare VioHawk's detection results with manually annotated results to assess the accuracy of the detection.

❷ The **Scenario Mutation** aims to evaluate the ability of VioHawk to mutate scenarios. For the given initial scenarios, we perform scenario mutation to generate new scenarios. The effectiveness of the mutations can be evaluated by observing the differences in the seed before and after the variation.

III. Request Available Badge

The prototype of VioHawk and other essential artifact materials have been released to the public through permanently accessible repositories (detailed as follows):

❶ The required **artifact materials** (i.e., the prototype of VioHawk, experiment dataset and environment setups) is available at https://zenodo.org/records/12666547.

❷ The **source code of VioHawk** is also available at https://github.com/emocat/VioHawk.

❸ The **preprint version of the paper** is available at https://github.com/emocat/VioHawk/blob/main/issta2024-viohawk-preprint.pdf.

IV. Request Functional Badge

To demonstrate the functionality of the artifact, we divided the evaluation into two parts. The first part is to evaluate the key techniques (including Violation Detection and Scenario Mutation), and the second part is to evaluate the whole fuzzing framework. The first part does not require GPU support, while the second part requires strong GPU support and cumbersome environment constructions to run ADS and the simulator.

A. Functionality of the Key Techniques of VioHawk

Environment Setup. We offer a docker-based execution environment for ease of evaluating our prototype.

❶ Enter the artifacts directory.

```
$ cd viohawk_artifacts/
```

❷ Build the docker.

```
$ docker build -t viohawk:latest .
$ docker run -it viohawk:latest
```

Experiment. In our paper accepted by ISSTA'24, We selected 42 basic scenarios and conducted simulation tests for each scenario on advanced ADS simulation platforms (e.g., LGSVL+Apollo). Then, using the generated trajectory files and initial scenarios, we performed violation detection and scenario mutation. However, re-executing all these 42 simulation test tasks on a complete testing system requires significant computing resources and CPU hours. Therefore, we suggest evaluating the functionality of our prototype only on a limited number of data inputs with pre-prepared trajectory files and scenario files (i.e., for each of the 20 scenarios, to detect one violation seed and one compliance seed and mutate the initial scenario file into new one). To be specific, you can execute the provided scripts located in the `viohawk_artifacts/Dataset/` directory.

❶ Violation Detection.

```
$ ./run_viohawk_detection.sh
```

The command above would run the Violation Detection of VioHawk on all 20 scenarios. Then the results of detection conducted under 20 normal scenarios and 20 violation scenarios would be printed.

❷ Scenario Mutation.

```
$ ./run_viohawk_mutation.sh
```

The command above would run the Scenario Mutation of VioHawk on all 20 initial scenarios for one round. After running the script, a mutation directory would be generated in each scenario's folder to save the mutated seeds.

Result Verification. The output files include the violation detection results of 20+20 seeds, the detection accuracy, 20 mutated scenarios, and the scenario mutation logs.

❶ The **violation detection** results of 20+20 seeds would be printed with True or False, and compared with manually labeled results to print the detection accuracy. To verify the validity of the detection, the detection accuracy would be an appropriate metric.

❷ The **scenario mutation** results of 20 scenarios would be saved as mutated seeds. To verify the validity of the mutated seeds, a more convenient method is to directly observe the changes in the seed files (`*.json`). Another more intuitive approach is to use LGSVL in a complete experimental environment (see below for the whole fuzzing framework) to detect changes in the initial scenario visuals.

❸ The **visualization** tool can visually display the hazardous areas and drivable areas of a given scenario. As an example, you can obtain the visualization result by running the following command:

```
$ python3 ~/VioHawk/mutator/mutator.py visualize
-s R1-traffic_light_red/red_light_C_straight/
violation.json -t R1-traffic_light_red/
red_light_C_straight/violation.txt -r
traffic_light_red -m R1-traffic_light_red/
red_light_C_straight/map/+.xodr
```

The above command would generate a GIF, which illustrates the trajectory of the autonomous vehicle, where the blue cars represent the surrounding vehicles, the red polygons indicate hazardous areas and the light blue polygons represent drivable areas. The GIF would be saved at `~/Dataset/output/ZAM-mutation-1.gif`

B. Functionality of the Whole Fuzzing Framework

Environment Setup. To verify the functionality of our fuzzing framework, we suggest dynamically running the fuzzing framework with advanced ADS simulation platforms (e.g., LGSVL+Apollo). However, since the setup process of ADS simulation platforms (e.g., LGSVL+Apollo) is quite complex, we are unable to provide convenient Docker containers or similar methods to quickly replicate the VIOHAWK experimental environment. Therefore, we have provided detailed machine setup steps to help you recreate the experimental environment as closely as possible. If you encounter any issues during the setup, please do not hesitate to contact us.

The hardware/software requirements to run this framework include: ❶ Ubuntu 18.04; ❷ 16+ GB memory; ❸ 8+ CPU cores; ❹ NVIDIA 20 or 30 series GPU (According to Apollo's official recommendations, using 40 series GPU may present some issues). After downloading the artifact, you can accordingly set up the environment (detailed as follows).

❶ **Install Apollo.** Apollo is currently the mainstream open-source autonomous driving system in academia. We recommend reviewers install Apollo 7.0. Apollo's official installation documentation can be referenced here: https://github.com/ApolloAuto/apollo/blob/master/docs/01_Installation%20Instructions/apollo_software_installation_guide.md.

❷ **Install LGSVL.** LGSVL Simulator is a high-fidelity simulation platform designed for autonomous vehicle. However, the official open-source version of LGSVL is not stable. Therefore, we provide our localized stable version of LGSVL. Reviewers can unzip the `lgsvl.zip` from the artifact to the user's home directory. Before running LGSVL, you should edit the `sim_path` in `lgsvl/config.py` and execute the config script.

```
$ python3 config.py
```

❸ **Install VioHawk.** Considering the complex environment setup of the whole fuzzing framework of VIOHAWK, we provide a one-click installation script. Reviewers can run the following script in the provided artifact to install VIOHAWK.

```
$ ./local_install.sh
```

Experiment. After correctly installing the tools and configuring the environment, the commands for running VIOHAWK with Apollo + LGSVL are as follows:

❶ Run the LGSVL simulator (in a GUI terminal) and wait for about 10 seconds, then you will see "API ready!" in the simulator:

```
$ cd ~/lgsvl
$ ./simulator --data ./2021.2-wise --simulationId 0 &
```

❷ Open a new shell, enter the Apollo directory, and enter the built Apollo docker. If docker is not running, then run the second command first:

```
$ cd ~/apollo
$ bash docker/scripts/dev_into.sh
$ bash docker/scripts/dev_start.sh
```

❸ Then enter the docker and restart the Apollo dreamview. You can open the dreamview web: http://localhost:8888/ if succeed.

```
$ ./scripts/bootstrap_lgsvl.sh restart
```

❹ Still in the docker, run the Apollo bridge to connect to the simulator:

```
$ ./scripts/bridge.sh &
```

❺ After that, open another new shell, enter the fuzz example directory, and run a fuzzing test. Sometimes it might take about one minute for Apollo to start.

```
$ cd viohawk_artifacts/Fuzz/red_light_C_straight
$ python3 ~/viohawk_artifacts/VioHawk/fuzzer.py
-i corpus -o out -m map/+.xodr -r
traffic_light_red
```

Result Verification. If VIOHAWK successfully runs, the ADS simulation platform can work properly (as shown in Figure 1) and the shell where running the fuzz example would print violation detection results and the fuzzing process. After each round of mutation is completed, the results will be promptly saved in the `~/out/` directory. We suggest reviewers observe the fuzzed violation scenarios. There are two methods to verify the functionality of fuzzing:

❶ Using the provided script, you can directly run the violation scenario's seed in the `~/out/` directory and observe the violation behavior of the autonomous vehicle in Apollo Dreamview.

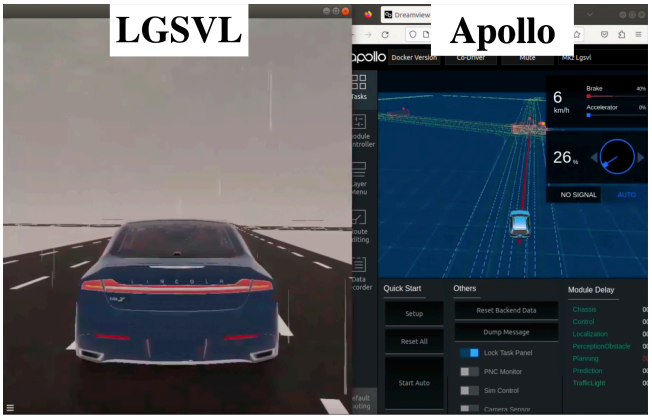

Fig. 1. A running example of LGSVL+Apollo Simulation Platform.

```
$ python3 ~/viohawk_artifacts/VioHawk/scripts/
run_vse_test.py -s $seed
```

❷ Additionally, you can use LGSVL's built-in visualization tool to view the distribution of objects in the scenario. You still need to open LGSVL (in offline mode), then select `VSE Editor` in the simulator window to enter the visualization editing page, and then select `Load` to load the violation seed file.

```
$ cd ~/lgsvl
$ ./simulator --data ./2021.2-wise &
```

V. Request Reusable Badge

Given the premise of functionality, the artifact also provides detailed operation descriptions and all means necessary to verify the reusability(detailed as follows):

❶ **Automation.** The artifact features a high level of automation, as we have designed comprehensive scripts for the entire process. Operators only need to select the initial rules and scenarios, and they can then conduct automated testing and fuzzing through the scripts. (see as in `fuzzer.py`)

❷ **Comprehensive documentation.** We provide a guiding README, which introduces the file structure of the project, lists the environmental requirements, and explains the operation process. Furthermore, we provide more details about how to extend the oracles to apply to more traffic rules. The documentation is provided at https://github.com/emocat/VioHawk/README.md.

❸ **All necessary means.** The artifact contains the source code, all not commonly available requirements, and a working description of compiling it. To enable others to use and expand this artifact successfully, we provide a docker-based execution environment and all necessary tools with their respective configuration instructions. All necessary items can be found at https://zenodo.org/records/12666547.