Scenario-based Test Reduction and Prioritization for Multi-Module Autonomous Driving Systems

Yao Deng
yao.deng@hdr.mq.edu.au
Macquarie University
Sydney, NSW, Australia

Xi Zheng*
xi.zheng@mq.edu.au
Macquarie University
Sydney, NSW, Australia

Mengshi Zhang*
mengshizhang@fb.com
Meta
Menlo Park, CA, USA

Guannan Lou
guannan.lou@mq.edu.au
Macquarie University
Sydney, NSW, Australia

Tianyi Zhang
tianyi@purdue.edu
Purdue University
West Lafayette, IN, USA

ABSTRACT

When developing autonomous driving systems (ADS), developers often need to replay previously collected driving recordings to check the correctness of newly introduced changes to the system. However, simply replaying the entire recording is not necessary given the high redundancy of driving scenes in a recording (e.g., keeping the same lane for 10 minutes on a highway). In this paper, we propose a novel test reduction and prioritization approach for multi-module ADS. First, our approach automatically encodes frames in a driving recording to feature vectors based on a driving scene schema. Then, the given recording is sliced into segments based on the similarity of consecutive vectors. Lengthy segments are truncated to reduce the length of a recording and redundant segments with the same vector are removed. The remaining segments are prioritized based on both the coverage and the rarity of driving scenes. We implemented this approach on an industry-level, multi-module ADS called Apollo and evaluated it on three road maps in various regression settings. The results show that our approach significantly reduced the original recordings by over 34% while keeping comparable test effectiveness, identifying almost all injected faults. Furthermore, our test prioritization method achieves about 22% to 39% and 41% to 53% improvements over three baselines in terms of both the average percentage of faults detected (APFD) and TOP-K.

KEYWORDS

Autonomous Driving, Testing Reduction, Test Prioritization, Regression Testing

ACM Reference Format:
Yao Deng, Xi Zheng, Mengshi Zhang, Guannan Lou, and Tianyi Zhang. 2022. Scenario-based Test Reduction and Prioritization for Multi-Module Autonomous Driving Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549152

*Corresponding authors: Xi Zheng, Mengshi Zhang.

1 INTRODUCTION

Autonomous driving has been through quick development in recent years and found several new business models such as driverless taxis [8], driverless buses [27], and last-mile robotic delivery [43]. Nowadays, modern autonomous driving systems (ADSs) such as Apollo [6] make use of machine learning models and logic-based controllers in tandem to process rich sensor data (e.g., images, GPS, point clouds) and plan driving trajectories. To ensure the robustness and reliability of ADSs, autonomous driving companies often conduct extensive on-road testing as well as simulation-based testing to cover various driving scenarios. For example, Google Waymo has a fleet of 25000 virtual vehicles traveling eight million miles per day in simulation [41]. Waymo has also conducted on-road test for more than 20 million miles in 2021 [55]. Despite the significant investment on testing, software bugs still slip through and cause vehicle recalls [45] and traffic accidents [11, 44, 56]. Therefore, new ADS testing methods are in urgent need to ensure the safety of autonomous vehicles [28, 38].

During ADS development and testing, a common practice is to reuse driving recordings collected from on-road testing or simulation to test new updates to ADSs, such as replacing a traffic light detection model with a new model. This is similar to regression testing in traditional software systems [58, 59]. However, it is costly to simply replay the entire recordings to test a change, since some recordings include many redundant driving scenes that are not necessary to repetitively test. In particular, a recent study finds that ADS practitioners wish to have tool support to accelerate the testing process and reduce the cost of testing [38].

Test reduction and prioritization techniques have been widely investigated in traditional software engineering [13–15, 21, 22, 30, 50]. However, these techniques are not directly applicable to ADSs. First, test cases in ADSs are driving recordings with time-series sensor data, rather than discrete program inputs as in traditional software. Therefore, it requires extra care to reduce driving recordings. Second, test coverage metrics [14, 50] used in existing techniques are not applicable to multi-module ADSs, since ADSs contain both logic-based code and machine learning models. While many testing techniques have been proposed for ADSs [1, 2, 9, 12, 20, 20, 23–25, 36, 37, 37, 52, 54, 60, 62], most of them focus on test generation rather than test reduction or prioritization. Furthermore, many of them can only handle a single driving model, rather than a multi-module system. To the best of our knowledge, only two recent techniques have been proposed to address test reduction or prioritization for ADSs [10, 39]. However, they can only handle limited driving scenarios, e.g., road shapes only. Furthermore, both of them require access to test configuration files (e.g., the road map) to extract road features, rather than directly from driving recordings. This limits their utility and generalizability in practice.

To fill the research gap, we propose a novel framework called STRaP (Scenario-based Test Reduction and Prioritization) that automatically extracts and analyzes rich driving scenarios from driving recordings. In particular, we formally define a driving scene schema with rich features, e.g., pedestrians, traffic lights, stop signs, and interactions. We further leverage the publish-subscribe communication mechanism to dynamically extract semantic information related to corresponding features from the communication channels between different modules in an ADS. This publish-subscribe communication mechanism is adopted by most multi-module autonomous driving systems [6, 34] and robot operating system in general [46]. Given a driving recording, STRaP first plays it to extract the corresponding semantic information in each frame of the recording from the communication channels of the ADS under test. The extracted semantic information is encoded to vectors for the ease of comparison and clustering. Then, STRaP slices the driving recording into continuous segments with the same frame vector. Lengthy segments are truncated to reduce the length and redundant segments with the same vector are removed. Furthermore, to expose potential errors (e.g., collisions) early, STRaP sorts the remaining segments based on the coverage of different driving scene features and the rarity of these features. This heuristic is inspired by coverage-based test prioritization approaches in traditional software engineering [59], as well as observations that corner cases or rare driving scenarios are more likely to detect faults [52, 54].

We implemented STRaP for an industry-level multi-module ADS called Apollo [6] and evaluated it in terms of *test reduction capability (RQ1)*, *test effectiveness of reduced recordings (RQ2)*, and *bug detection speed after prioritization (RQ3)*. We created a benchmark of driving recordings on three different types of road maps in a simulator called SVL [47]. We further developed a mutation testing tool to systematically inject errors to different modules of Apollo and evaluated the effectiveness of reduced and prioritized ADS tests. The experiment results show that (1) STRaP reduces over 34% of given driving recordings and thus significantly reduces the testing time in ADS regressing testing; (2) The reduced driving recordings have comparable test effectiveness—they detected 99% of injected faults that were detected by the original driving recordings; (3) The diversity-based test prioritization method in STRaP achieves 39%, 33%, and 22% improvement compared with a coverage-based method related to code change, chronological prioritization, and random prioritization.

There are three main contributions of this work:

- We propose a general scenario-based test reduction and prioritization framework for multi-module autonomous driving systems that adopt the publish-subscribe communication mechanism.
- We make a regression testing benchmark publicly available. The benchmark includes driving recordings on three different kinds of road maps, as well as a mutation testing tool that systematically injects errors in different ADS modules in Apollo and evaluates test effectiveness.[1]. With this benchmark, future researchers can systematically evaluate their ADS testing methods in various regression settings.
- We conduct experiments on an industry-level multi-module ADS and demonstrated the test reduction capability and test effectiveness of our proposed framework.

The rest of the paper is organized as follows: Section 2 introduces the background of current ADS testing practice and multi-module ADSs. Section 3 formulates testing reduction and prioritization problems. Section 4 describes proposed testing reduction and prioritization methods. Section 5 introduces experiment settings. Section 6 demonstrates experiment results. Section 7 introduces related works. Section 8 describes threats to validity of the work. Section 9 concludes the paper.

2 BACKGROUND

2.1 ADS Testing Practices in the Industry

Two primary methods of testing ADSs in the industry are on-road testing and simulation-based testing [38]. The goal is to cover various driving scenarios, especially corner cases, and check consistency and generalizability of ADSs in these scenarios. Such testing processes will produce massive amounts of log data, including sensor data (e.g., video recordings, point clouds) and outputs of ADS modules (e.g., obstacle detection results, speed control commands). These log data will be further processed for regression testing in a simulation environment. When an ADS is updated, it will be tested by replaying previously logged sensor data and comparing the system outputs of the new ADS with previous outputs (e.g., collision or deviation) [53]. However, replaying an entire recording takes a long time and computation power. Alternative, some ADS developers will manually label and clip recordings to obtain specific driving scenarios (e.g., overtaking, merging) for testing. Yet since this requires a lot of manual effort, it cannot be done in a large scale.

2.2 Multi-Module ADSs

Modern ADSs such as Baidu Apollo [6] and Autoware [34] are composed of multiple modules to process rich sensor data and make trajectory plans. A typical multi-module ADS includes perception, localization, prediction, planning, and control modules, as shown in Figure 1. Similar to robot operating systems (ROS) [46], these modules in an ADS communicate and coordinate with each other through *the publish-subscribe communication mechanism.* Each module maintains its channel(s) to publish module outputs. Specifically, when a module generates its output at a timestamp, it packages

[1]Our benchmark has been open-sourced in an anonymous GitHub repository at https://github.com/ITSEG-MQ/STRAP

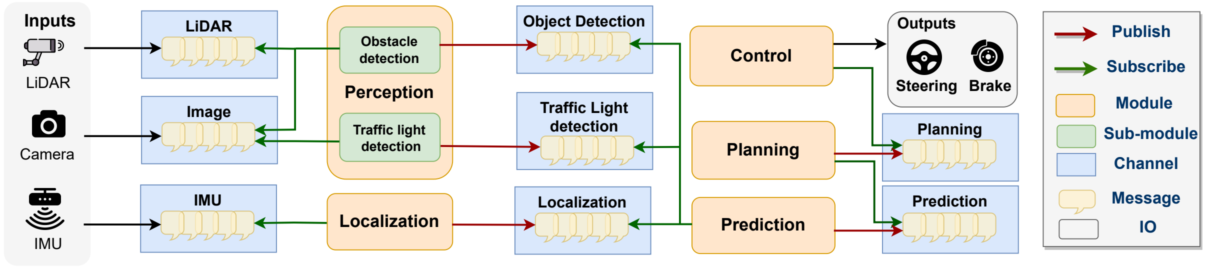


Figure 1: The Architecture and Data Flow of a Multi-Module ADS

the output as a message and publishes the message into its channel. Each module also subscribes to one or more channels of other modules to obtain needed data.

Figure 1 gives an overview of an multi-module ADS based on the architecture of Apollo. As the autonomous vehicle is driving, sensors such as cameras and LiDARs continuously capture environment information, package collected data to messages, and send messages into the corresponding channels. In the perception module, a traffic light detection model subscribes to the message from the image channel and detects traffic lights in the images. The model predictions are then published to the traffic light detection channel. Similarly, the obstacle detection model subscribes to both the image channel and point cloud channel to detect and classify obstacles on the road, such as vehicles and traffic cones. The detection results are published to the obstacle detection channel. The prediction module subscribes data from localization, traffic light detection and obstacle detection channels to predict the trajectories of detected obstacles. The prediction result is published to the prediction channel, which is subscribed by the planning module. Finally, the planning module generates the trajectory of the ego-vehicle, while the control module subscribes data from the planning channel and outputs the control commands such as steering angles and brakes.

2.3 Regression Testing on Driving Recordings

A driving recording includes all sensor data and module outputs that are packaged as messages in different channels during the running of an ADS, as shown in Figure 2. Mathematically, given a start time point t_a and an end time point t_b, a **driving recording** is denoted as $R_{a\to b}$. Suppose it contains n channels $\{C^i_{a\to b}|1 \le i \le n]\}$. Each channel $C^i_{a\to b}$ contains all messages m^i_j created in the time period $[t_a, t_b]$, denoted as $C^i_{a\to b} = \{m^i_j|a \le j \le b\}$ where t_j is the timestamp when a message is created. Given a timestamp t_a, a **frame** f_a is a slice of a recording that contains all channel messages created at timestamp t_a, denoted as $f_a = \{m^i_a|1 \le i \le n\}$. Therefore, a driving recording can be seen as a list of frames, denoted as $R_{a\to b} = \{f_i|a \le i \le b\}$. A **recording segment** $s_{a'\to b'}$ is a clip of the driving recording containing frames created in $[t_{a'}, t_{b'}]$, where $a \le a' \le b' \le b$. However, since different modules are invoked in different time orders and frequency, the number of messages and the creation time of messages in different channels are not quite aligned. This message alignment problem is discussed and solved in Section 4.1.1. Table 1 summaries terminologies and notations in this paper.

Table 1: Terminology Definitions

Terminology	Symbol	Meaning
Recording	$R_{a\to b}$	A log file that stores sensor data and module outputs in different channels from the timestamp t_a to t_b.
Channel	c	A message queue to store the outputs of a module
Message	m_a	The output of a module created at the timestamp t_a
Frame	f_a	A slice of the recording at timestamp t_a
Segment	$s_{a\to b}$ | s_i	A list of frames created from t_a to t_b | The i^{th} segment in a segment list
Frame feature	θ_i	An attribute to represent specific driving scenario-related semantic information in a frame
Frame feature value	v^a_i	The value of the frame feature θ_i for the frame f_a
Frame vector	$\mathbf{v}_a$	A vector $(v^a_1, v^a_2, ..., v^a_n)$ to represent the frame f_a
Segment vector	$\mathbf{sv}_i$	A vector to represent the segment s_i

Messages in a channel can be used to perform regression testing on the module that subscribes to the channel. For example, suppose a driving recording $R_{a\to b}$ with m frames, the image channel $c^{img}_{a\to b}$ in $R_{a\to b}$ with m messages of collected images, and the traffic light detection channel $c^{light}_{a\to b}$ with m messages of traffic light detection results. When the traffic light detection system is updated, by replaying the driving recording, the new version of traffic light detection system can take images from the recording frames and output detection results. Then, by comparing the new detection results with the original detection results in the traffic light detection channel $c^{light}_{a\to b}$, we can check whether inconsistent model predictions or discrepancies are introduced. For example, if a traffic light detected as in red by the old traffic light detection system but it is detected as in green by the new one, it implies a potential fault. Other ADS modules can be tested in the same way.

3 PROBLEM FORMULATION

In this section, we formulate the test reduction problem and the test prioritization problem in ADS testing.

Test reduction: Given a driving recording $R_{a\to b}$, a test reduction method α should output a list of recording segments $S = \{s_1, s_2, ..., s_n\}$, denoted as $S = \alpha(R_{a\to b})$, where n is the number

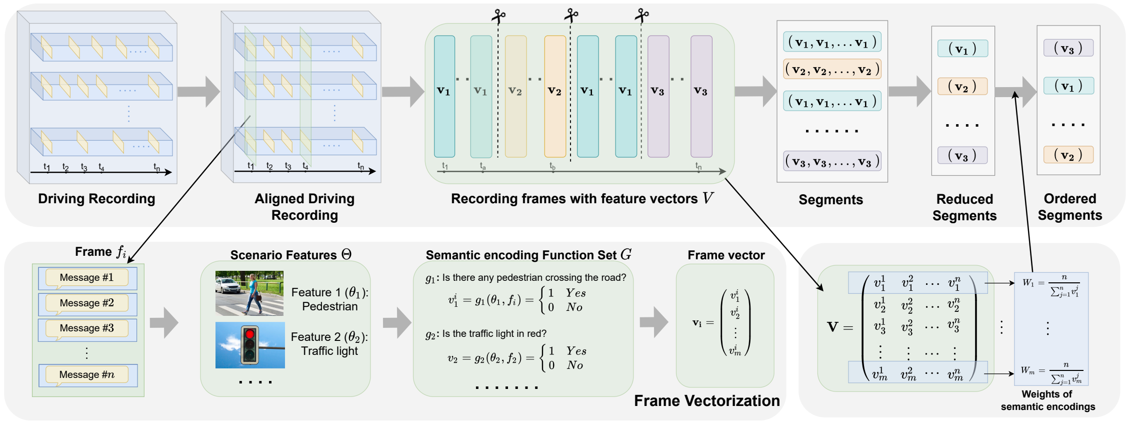

Driving Recording
Aligned Driving Recording
Recording frames with feature vectors V
Segments
Reduced Segments
Ordered Segments
Frame f_i
Message #1
Message #2
Message #3
Message #n
Scenario Features Θ
Feature 1 (θ_1): Pedestrian
Feature 2 (θ_2): Traffic light
Semantic encoding Function Set G
g_1: Is there any pedestrian crossing the road?
g_2: Is the traffic light in red?
Frame vector
Frame Vectorization
Weights of semantic encodings

Figure 2: The architecture of proposed regression testing reduction and prioritization method

of recording segments. The total time cost of replaying recording segments in S should be less than the time cost of replaying $R_{a\to b}$. Meanwhile, S should be able to detect the same number of faults as $R_{a\to b}$. More specifically, test reduction shall comply with two constraints: $\sum_{i=1}^{n}(T(s_i)) \leq T(R_{a\to b})$ and $\sum_{i=1}^{n}(\tau(s_i, SUT)) = \tau(R_{A\to b}, SUT)$, where n is the number of recording segments, T is a function to output replaying time cost of a given segment or recording, SUT is the testing module, and τ represents the number of detected faults in the testing module using the input segment or recording.

Test prioritization: Given n recording segments $S = \{s_1, s_2, ...s_n\}$ that detect m faults in the new version of an ADS module, test prioritization aims to sort the recording segments and obtain a list $P = [p_1, p_2, ..., p_n]$, where p_i is the execution order of the recording segment s_i. For example, if $p_1 = 5$, the recording segment s_1 will be the fifth to replay. The sorted recording segments should detect all m faults as early as possible.

4 APPROACH

Figure 2 shows the overview of our approach, STRaP. First, given a driving recording, STRaP performs message alignment across different channels and converts messages in each time frame into vectors based on a driving scene schema (Section 4.1). Second, STRaP slices the given recording into segments based on the similarity of consecutive vectors. Long segments of the same continuous vectors will be truncated and redundant segments will be removed (Section 4.2). Third, the remaining segments are prioritized based on the coverage of driving scene features and the rarity of these features (Section 4.3).

4.1 Recording Alignment and Vectorization

4.1.1 Recording message alignment. As different modules in ADS run asynchronously and in different frequencies, the timestamp and the number of messages in different channels are not aligned. For example, in Apollo, the localization module logs a message every 0.1 second, while the prediction module logs a message every 0.15 second. Since the messages in different channels are not aligned,

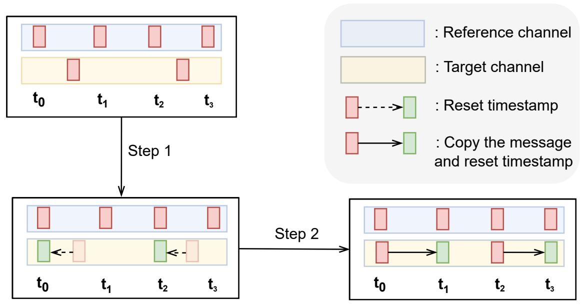


Figure 3: An example of message alignment

raw driving recordings cannot be used as-is for comparison and clustering. Therefore, we need to align them first.

We propose a novel message alignment algorithm to address this issue. Specifically, we choose the channel containing most messages as the reference channel to align messages of other channels (i.e. target channels). Given the reference channel C_{ref}, at each time we retrieve two consecutive messages m^{i}_{ref} and m^{i+1}_{ref} from it and obtain their creation timestamps t_i and t_{i+1}. Then, for the target channel, if the message is generated between (t_i, t_{i+1}), we reset the timestamp of the message to t_i, as shown in Step 1 of Figure 3 . Then, we iterate over the reference channel and the target channel to check whether at t_i there is a corresponding message in the target channel. If no message is found, we copy the prior message at t_{i-1} in the target channel and reset its timestamp to t_i as shown in Step 2 of Figure 3. In this way, we align all channels to ensure that the aligned driving recording can be sliced to a list of frames.

4.1.2 Schema-based Recording Vectorization. Given an aligned driving recording, STRaP converts each frame into a vector $\mathbf{v}$, where each dimension represents specific semantic information in a driving scene. We formally define a schema for a variety of semantic information in a driving scene, as shown in Figure 4. A driving scene is represented as a list of static or dynamic objects. A dynamic object is defined of an actor (e.g., vehicles, pedestrians, etc.) and an action of the actor (e.g., stop, cruise, etc.). A static object

$$
\begin{aligned}
scezzz &:= \epsilon \mid object\ ;\ scene\\
object &:= dynamic\ object \mid static\ object\\
dynamic\ object &:= <actor,\ action>\\
actor &:= vehicle \mid \texttt{pedestrian} \mid cyclist \mid \texttt{unknown}\\
vehicle &:= \texttt{truck} \mid \texttt{car} \mid \texttt{bus} \mid \texttt{van}\\
cyclist &:= \texttt{bicyclist} \mid \texttt{motorcyclist} \mid \texttt{tricyclist}\\
action &:= \texttt{stop} \mid \texttt{cruise} \mid \texttt{change lane} \mid \texttt{overtake} \mid \texttt{cross} \mid \ldots\\
static\ object &:= traffic\ light \mid \texttt{stop sign} \mid \texttt{crosswalk} \mid \texttt{intersection}\\
&\quad\ \texttt{traffic cone} \mid \ldots \mid \texttt{unknown}\\
traffic\ light &:= <color, shape, orientation>\\
color &:= \texttt{red} \mid \texttt{green} \mid \texttt{yellow} \mid \texttt{black}\\
shape &:= \texttt{square} \mid \texttt{round}\\
orientation &:= \texttt{vertical} \mid \texttt{horizontal}
\end{aligned}
$$

Figure 4: Part of the Driving Scene Schema

can be traffic light, stop sign, crosswalk, interaction, traffic cone, etc. An object can have subcategories and also properties. Our driving scene schema is designed as a general schema for all driving systems. Yet one can customize it based on the concrete channel messages logged by an ADS. For example, in Apollo, each detected object is also logged with its coordinates and heading direction (if movable). Such information can also be encoded as new dimensions in the vector. The encoding function in our current implementation only considers the existence of different types of objects and their properties in a driving scene. Given channel messages in a time frame, it parses the channel messages into a list of objects based on the schema and flattens the list of extracted objects and their properties into a vector of integers using label encoding. Specifically, 0 is a reserved code for none (i.e., undetected objects or properties). This encoding function also enforces a specific ordering of objects based on their types for the ease of vector comparison in the next step. For example, consider a driving scene that contains vertically-aligned, round traffic lights with the red light on. The corresponding feature dimensions in the final vector is $<22, 34, 39, 41>$ where 22 is the unique code for traffic light, 34 is the code for red, 39 is the code for the round shape, and 41 is the code for vertical alignment. If no traffic light is detected, the four dimensions are all set to 0.

After vectorization, a driving recording is represented as a list of vectors, where each vector corresponds to each time frame in the recording. In regression testing, if one or more updated ADS modules are given as input, STRaP will further simplify the vectors by only retaining features obtained from the channels that the updated modules subscribe or publish to. For example, suppose only the traffic light detection model is updated. Since the traffic light detection model takes raw images as input and publishes prediction results to the traffic light detection channel, features obtained from the traffic light channel (e.g., traffic light color, shape, and orientation) will be retained in the final vectors. To avoid over-simplifying the vectors, we specify several features that should always be preserved in the final vectors, including stop sign, intersection, and crosswalk.

4.2 Test Reduction

4.2.1 Segmentation. STRaP slices the given recording into segments based on the similarity of consecutive vectors. If vectors in a consecutive time range are the same, then they will be sliced into

Algorithm 1: Recording Segmentation and Reduction

```
Input : V: a list of frame feature vectors with size $N$, $N$ is
        the number of frames;
        l: the clip length of a segment;
        w: the sliding window size;
Output : S: a list of reduced driving segments;
         SV: a list of segment vectors;
$S_{temp}$, $SV_{temp}$ ← [];
// Initiate the start and end indices of a
   segment
ss, se ← 0;
// Create an empty list to save smoothed feature
   vectors
$V_{temp}$ ← [];
// Smoothing using sliding window
for $i \leftarrow 1$ to $N$ do
    // Obtain the window of frame feature vectors
    if $i \le \frac{w}{2}$ then
        window ← V $[i : i + w - 1]$;
    else
        if $i \ge N - \frac{w}{2}$ then
            window ← V $[i : i + w - 1]$;
        else
            window ← V $[i - \frac{w}{2} : i + \frac{w}{2}]$;
        end
    end
    // Set $V_{temp}[i]$ as the majority vector in the
       window
    $V_{temp}$.append(Majority(window));
end
// Recording segmentation
for $i \leftarrow 2$ to $N$ do
    if V $[i] \neq$ V $[i-1]$ or $i == N$ then
        se ← i ;
    else
        $S_{temp}$.append($V_{temp}$ [ss : se]);
        $SV_{temp}$.append($V_{temp}$ $[i-1]$);
        // Reset segment indices
        ss, se ← $i$;
    end
end
// Clip each segment to length $l$
foreach $s$ in $S_{temp}$ do
    $s$ ← Clip($s$, $l$);
end
// Drop segments with the same segment vector
S, SV ← [];
for $i \leftarrow 1$ to $S_{temp}.length$ do
    if !SV.contain($SV_{temp}$ $[i]$) then
        SV.append($SV_{temp}$ $[i]$);
        S.append($S_{temp}$ $[i]$);
    end
end
return S, SV;
```

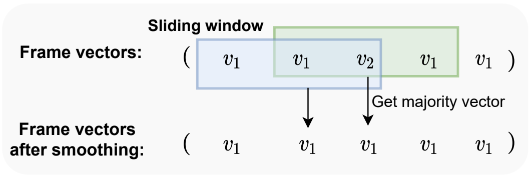

Sliding window
Frame vectors: (v_1 v_1 v_2 v_1 v_1)
Get majority vector
Frame vectors after smoothing: (v_1 v_1 v_1 v_1 v_1)

Figure 5: Smoothing frame vectors with a sliding window

one single segment. For example, in a 1-hour driving recording, if the ego-vehicle drives on a highway with no new objects detected for 10 minutes, the vectors for each time frame in this 10 minutes will be the same and therefore this 10 minutes will be sliced into a single segment.

In practice, we noticed that due to noises in raw sensor data and the uncertainty of DL models, channel messages often contain *glitches*—exceptional predictions that only exist for a very short period of time (e.g., 0.1 second). Such glitches often do not lead to abnormal ADS behavior since they only exist very shortly and ADS make decisions based on a sequence of time frames, not just one. However, it has a significant impact on our segmentation algorithm. Specifically, it will lead to many tiny segments since a glitch will produce an exceptional vector that is different from its preceding and following vectors. To handle this issue, STRAP smooths glitches in the list of vectors using a sliding window, as shown in Lines 4–16 of Algorithm 1. Figure 5 illustrates this process with a sliding window of 3 vectors ($n = 3$). When smoothing is finished, we split frames into segments. Each segment contains a list of frames represented by the same vector, which is denoted as the segment vector.

4.2.2 Segment Reduction. STRAP adopts two strategies to reduce the length and the number of segments. First, for each driving segment, as each frame in the segment is semantically equivalent (i.e., represented by the same vector), STRAP only keeps the first n frames in the segment. STRAP keeps n frames rather than one frame, because certain ADS modules, such as the planning module, rely on a sequence of frames to make a decision, not just one frame. In our current implementation for Apollo, we experimented with different number of frames and finally set n to 45 (roughly 3 seconds). Second, as each segment reflects a unique driving scenario, we only keep one unique segment and remove those segments with identical segment vectors. By combining these test reduction strategies, we obtain a list of reduced segments as test cases to evaluate ADS modules.

4.3 Test Prioritization

Coverage-based prioritization methods have been widely investigated in traditional software systems. These methods sort test cases based on the total or additional coverage of program statements or branches [4, 59]. Inspired by these methods, we propose the notion of *semantic coverage* of different driving scenes by counting the number of non-zero feature dimensions in a vector. Note that 0 is a preserved code that indicates the corresponding object or property does not exist. Intuitively, a driving scene with a traffic light and a pedestrian in a crosswalk has a higher coverage than a

Algorithm 2: Coverage and Rarity Based Prioritization

```
Input : V: a list of frame feature vectors with size $N \times Q$; $N$
        is the number of frames, $Q$ is the number of
        elements in a frame feature vector;
        SV: a list of segment vectors with size $M \times Q$; $M$ is
        the number of segments after test reduction;
Output : p: a list of segments' execution order with size $M$;
1  weights ← [];
2  /* Initialize a list weights to save the weights
      of feature elements and a list rarity to save
      the rarity degree of segments              */
3  for $i \leftarrow 1$ to $Q$ do
4      weights [$i$] ← 0;
5      rarity [$i$] ← 0;
6  end
7  // Calculate weights of feature elements
8  for $i \leftarrow 1$ to $Q$ do
9      weights [$i$] ← $\frac{N}{\sum_{j=1}^{N} \texttt{Nonzero}(V[i][j])}$ ;
10 end
11 // Weight normalization
12 for $i \leftarrow 1$ to $Q$ do
13     weights [$i$] ← $\frac{\text{weights}[i]}{\sum_{j=1}^{Q} \text{weights}[j]}$ ;
14 end
15 // Calculate rarity of each segment
16 for $i \leftarrow 1$ to $M$ do
17     rarity [$i$] ← $\sum_{j=1}^{Q}(\text{weights}[j] \times SV[i][j])$;
18 end
19 // Get indices of rarity after descending sorting
20 p ← DescendingArgsort (rarity);
21 return p;
```

driving scene with only a pedestrian. However, only considering the coverage of driving scenes may not be sufficient. Since ADSs are equipped with DL models for perception and prediction, common driving scenes are less likely to expose faults compared with rare driving scenes or corner cases, since common driving scenes are prevalent in training data. This phenomenon has been observed by several existing works in ADS testing [52, 54]. Therefore, we should also take into account the rarity of driving scenes when prioritizing recording segments.

We define a new prioritization method that accounts for both the coverage and rarity of different driving scenes. First, given a driving recording represented by a list of frame vectors $R = \{v_1, v_2, ..., v_n\}$, we calculate the rarity score of each dimension in the vector by the formula at Line 9 of Algorithm 2, where n is the number of frames, $\texttt{nonzero}(V[i][j]) = 1$ when $V[i][j] \neq 0$. The formula presents that the semantic information occurring less times in a driving recording has a higher rarity score. The process is shown in Lines 8 – 10 of Algorithm 2. Then, we normalize weights to $[0, 1]$ (Lines 12 – 14 of Algorithm 2). Then, for each recording segment represented by its segment vector, it sums the rarity score of non-zero feature elements in the corresponding segment vector (Lines 16 – 18 of Algorithm 2).

Finally, we sort the list of recording segments in descending order based on the rareness of recording segments. The sorted indices of the recording segments list is obtained as the execution orders of recording segments.

5 EXPERIMENTS

5.1 Research Questions

To evaluate the effectiveness of STRaP, we conducted experiments to answer the following research questions:

- RQ1: To what extent can the proposed test reduction method reduce a given driving recording?
- RQ2: Compared with an original recording, how effective is the reduced recording in detecting faults?
- RQ3: Compared with alternative prioritization methods, how effective is the proposed test prioritization method in detecting faults?

We implemented and evaluated STRaP on an industry-level ADS, Apollo 5.0 [6]. We chose Apollo 5.0 since it is a mature version with stable modules. We constructed our experiment as follows. First, we collected driving recordings from a simulator SVL [47]. Then, we injected mutants into the source code of Apollo to simulate the module changes with faults. Third, we tested the traffic light detection module, the obstacle detection module, the planning module and the prediction module to build baselines. Finally, we applied STRaP to split driving recordings, generated reduced and prioritized test segments, and evaluated STRaP's effectiveness.

5.2 Benchmark Creation

We created simulation environments using SVL [47]. SVL is a high fidelity simulator to render driving environments, which provides bridges to connect with ADSs such as Apollo and Autoware. The rendered driving environment is fed to an ADS via the bridge. Then, the ADS outputs control commands and sends them back to SVL to render the next driving scene based on the control commands. We collected driving recordings from three maps including Cubetown, Gomentum, and Shalun in SVL to create three testing suites. Specifically, Cubetown is a simple map containing a circular road. Gomentum (as shown in Figure 6) and Shalun are reconstructed maps based on real-world autonomous vehicle testing (e.g., on-road testing) environments. These three maps cover the various driving environments in urban and rural areas. The weather condition and non-player characters (e.g., vehicles and pedestrians) behaviors are randomly generated by the simulator in default settings. For each test suite, the destination waypoints are randomly generated by the simulator. To collect sufficient data, we manually inspected the waypoints and chose those waypoints with the most complex trajectories. When Apollo 5.0 was connected with SVL, we ran a recorder in Apollo to record all channel messages generated during the simulation. In the end, collected test suites contain driving recordings of different lengths (Cubetown: 84.14 seconds, Gomentum: 96.22 seconds, and Shalun 73.28 seconds). For each driving recording, we applied proposed reduction method to obtain reduced driving segments. Each driving segment is separately replayed to test the ADS. To ensure that Apollo is in a correct state before replaying each segment, STRaP replayed the one second before the segment

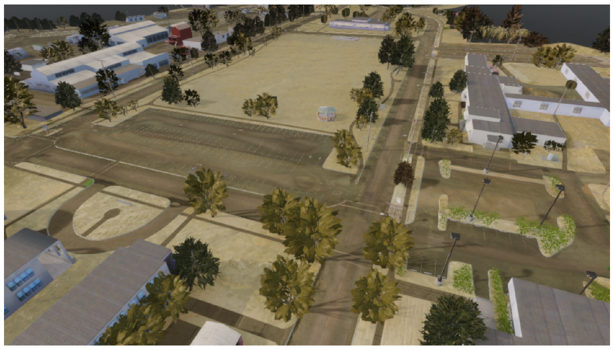
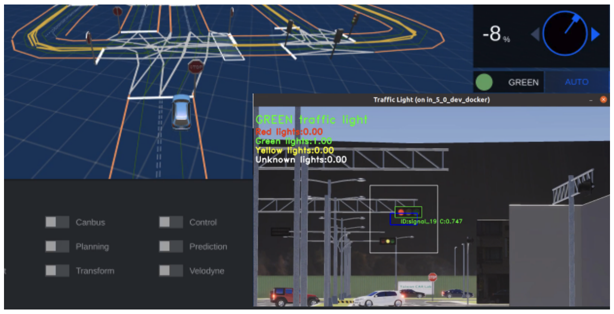


Figure 6: The Gomentum map in SVL



GREEN traffic light
Red lights:0.00
Green lights:1.00
Yellow lights:0.00
Unknown lights:0.00

Figure 7: A bug detected by replaying the recording segment. A new traffic light detection model classifies a red light as a green light, which is inconsistent with the previous run

in the original recording to initialize modules and restore system states.

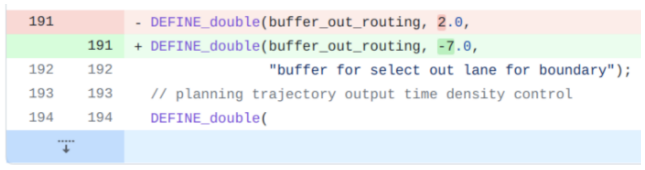
We modified the source code of four modules to simulate the module change with faults. For those modules under test, we implemented a mutation testing [31] tool to randomly inject mutants into the source code. The mutants include commonly used mutation operators such as replacing arithmetic operators, changing constant values, changing variable values, and changing condition operators, where the generated mutants can simulate real faults[2][3][4]. An example is shown in Figure 8[5]. For each module, 9 c++ files were randomly selected for mutation and totally, 36 regression testing benchmarks are created for each test suite.

5.3 Evaluation Metrics

For RQ1, we used the **reduction ratio of testing time** to reflect the effectiveness of the test reduction method, which is defined as the length of the reduced driving recording over the length of the original driving recording.

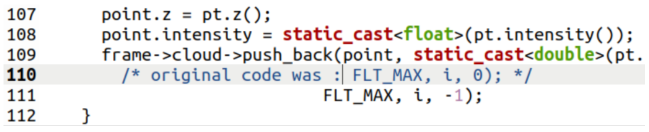
For RQ2, we used the **fault coverage** to reflect the quality of the test reduction method, which is defined as the ratio of the number

[2]Changing constant values:https://github.com/ApolloAuto/apollo/commit/8ddb8f57ccc7e70c85d9288c85c781ec8e5aec76
[3]Changing condition statement: https://github.com/ApolloAuto/apollo/commit/c506adedb968ecde7fee89e015c71ef2a5e95c7c
[4]Replacing arithmetic operators: https://github.com/ApolloAuto/apollo/commit/e3eef077630656aa92b32a12059f31f891d719ff
[5]https://github.com/ApolloAuto/apollo/commit/539839c71452aedeecc30ff3df0d02aec4ec55fa



```
- DEFINE_double(buffer_out_routing, 2.0,
+ DEFINE_double(buffer_out_routing, -7.0,
                "buffer for select out lane for boundary");
  // planning trajectory output time density control
  DEFINE_double(
```

(a) A real fault of correcting buffer size



```
    point.z = pt.z();
    point.intensity = static_cast<float>(pt.intensity());
    frame->cloud->push_back(point, static_cast<double>(pt.
      /* original code was : FLT_MAX, i, 0); */
                         FLT_MAX, i, -1);
  }
```

(b) A mutant of changing constant value

Figure 8: An example of real and artificial fault

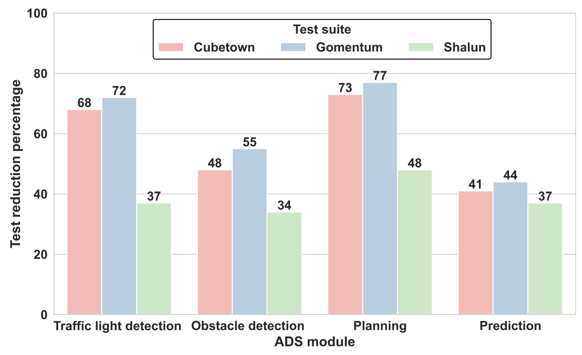


Figure 9: Test reduction percentage when updates are made on different ADS modules

of the detected faults in reduced driving recording over the number of the detected faults in the original driving recording. In our work, we defined a fault as the inconsistency of a module output before and after code changes (e.g., injecting a mutant). Since each output is a driving segment, we applied the frame vectorization approach to such output and retrieved the frame vector list representing the output. For the outputs before and after change, we compared each frame pair. Intuitively, if we find any mismatched frame pairs, we claim an inconsistent case is detected. To reduce the systematic noises introduced by Apollo 5.0, we recognized the inconsistency only when more than 10% of frames were mismatched. This treatment was proposed since our benchmark study showed that the inconsistency rate could be up to 10% when we replicated experiments on the benchmarks.

We calculated the number of faults from the reduced and the original driving segments. For those detected faults, we manually checked the visualized driving segments (before and after a ADS module change) and removed semantically identical faults (e.g., red traffic light is recognized as green, as shown in Figure 7) to improve experiment quality.

For RQ3, we used metrics **Top-K** and **Average Percentage of Faults Detected (APFD)** [51] to evaluate the effectiveness of test prioritization. Given n driving segments containing m bugs, Top-K measures the amount of segments required for finding the first fault. APFD measures the capability of fault detection per percentage of test cases execution. The calculation of APFD is shown in Formula 1,

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{mn} + \frac{1}{2n} \tag{1}$$

where n is the number of test recording segments, m is the number of faults, TF_i is the index of the first segment that detects fault i. For example, if $TF_2 = 3$, it means the second fault is detected at the first time when replaying the third prioritized recording segment. A higher APFD value means the test suite can find all faults faster.

5.4 Experiment Settings

For test reduction experiments, we clipped each segment by keeping its first 45 frames as explained in Section 4.2. To evaluate the test prioritization method, we created three baselines. The first baseline sorts the segments in chronological order (i.e., *CH*). The second baseline randomly sorts the segments (i.e., *RD*). Specifically, we randomly sorted the reduced segments by 100 times and evaluated the average performances using Top-K and APFD. For the third baseline, we first identified the code change in which function in the source code. Then, we counted the number of calls on the changed function for each recording segment based on the Apollo running log data. Finally, we sorted recording segments in descending order by the number of calls on the changed function (i.e., *CC*). In addition, we marked the proposed semantic coverage method as *SC* and the rarity and semantic coverage-based method as *RSC* in Section 6. All experiments were conducted on a Ubuntu PC with Intel i7-8700 3.2GHz, 32GB of memory, and a NVIDIA GTX 1080Ti GPU.

6 RESULTS

6.1 RQ1: Test Reduction Capability

Figure 9 shows the test reduction percentage for three test suits on four testing modules. The result shows that for the planning module in Gomentum, the reduction ratio of testing time achieves 77%. Similarly, the reduction ratio of testing time for the planning module in Cubetown achieves 73%. A similar pattern can be observed for the traffic light detection and obstacle detection modules in Gomentum and Cubetown where the test reduction ratios are in general close or above 50%.

We noticed the test reduction ratios are relatively lower for modules in Shalun (34% to 37%). We speculated the main reason is that Shalun contains more complex driving scenarios and finer-granularity frame feature encoding functions can be explored besides just using existence of semantic information. We also noticed that test reduction ratios for all test suites on the prediction module are relatively lower than on other modules. We manually inspected the output from the prediction module in Apollo and noticed that the trajectory prediction for vehicles in the ADS tends to fluctuate a lot. For instance, for a vehicle in front of the ego-vehicle, the prediction module in one frame detects the vehicle is driving in a straight line ahead, but in the next frame the prediction result is the vehicle is turning right. Such dynamic prediction changes across frames will result in different semantic feature encoding for continuous frames for the prediction module thus impacting the reduction ratios. Nevertheless, the reduction ratio is still promising with the minimum reduction ratio of 37% in the prediction module.

6.2 RQ2 & RQ3: Test Effectiveness of Test Reduction and Prioritization

In Table 2, we present the effectiveness of the reduced recording in detecting faults on the left-hand side. For the traffic light detection system, besides Shalun, in other two test suites, the fault coverage is 100%. For Shalun, the fault coverage is 92.3% which is also promising. For the 6 missing faults, we manually inspected the traffic light module output and noticed for vertical traffic light in Shalun (in other two suites, the traffic lights are mostly horizontal), the module output sometimes gave wrong bounding box for traffic lights in a very short duration but the glitch is within our threshold for smoothing (10%). This might contribute to the 6 missing faults as such short-duration glitch is embedded in those driving frames discarded. The 10% smoothing threshold is empirically retrieved as the best value for Apollo through our pilot study. Overall, the fault coverage achieves on average 98.8%, which proves the reduction technique is effective across four modules in different test suites.

We also present on the right-hand side of Table 2 the comparison results of our proposed *SC* and *RSC* against the baselines using Top K and APFD metrics. For each module in a specific test suite, the presented results are the averaged values on nine benchmarks. We observed that *RSC* outperforms other baselines for all test suites for prediction module both in Top K and APFD. Similarly, for planning module, *RSC* also outperforms others both in Top K and APFD except in the Shalun test suite for Top K (1.44 VS 1.0 both in *CC* and *SC*). We speculated that even a specific driving segment containing the rarest semantic information are ranked as the top in *RSC*this driving segment does not contain faults. However, empirically the correlation between semantic rareness and faults is very positive as shown in the table as a general trend. For obstacle module, *RSC* has lower value in APFD than the baselines (0.47 VS 0.51 in *CH*) in Gomentum. Similarly, in traffic light detection system, *RSC* outperforms all other baselines in Top K, but for APFD has lower value both in Gomentum (0.76 VS 0.82 in *SC*) and Shalun (0.54 VS 0.56 in *SC*). We believe because the semantic information in both traffic light and obstacle detection modules are largely static. Thus capturing rarest information in such modules has fluctuating performance either scoring the best in Top K (capturing the first error fastest) or APFD (capturing all the errors fastest). In summary, on average, that our method *RSC* achieves the best performance across all modules in different test suite with 1.58 for Top K and 0.61 for APFD.

7 RELATED WORK

7.1 Test Reduction and Prioritization

Test reduction and prioritization are two approaches to reduce the cost of regression testing. Test reduction, also called test minimization [48], seeks to reduce the size of a test suite by removing redundant test cases. Test prioritization [49] aims to maximize some desired properties such as the rate of fault detection by sorting test cases. The comprehensive overview of regression testing techniques can be checked in the survey [59].

The typical test reduction techniques contain heuristic approaches [13–15, 30] and genetic algorithm-based methods [40, 42]. Different heuristics and search algorithms were applied to select a minimal set of test cases that achieve the same test requirements (e.g., branch coverage and code coverage) as the original test suite. For test prioritization, many coverage-based prioritization methods [21, 22, 50] were proposed to maximize structural coverage or code coverage early. The idea behind these methods is that the early maximization of structure coverage will improve the chance to detect faults early [59]. These existing test reduction and prioritization methods target on traditional programs. However, they cannot be directly applied on ADSs with time series inputs. Inspired by traditional coverage metrics, in this paper we propose rarity and semantic coverage-based test prioritization method.

There are a few works related to test reduction and prioritization on ADSs. In [39], Lu et al. conducted experiments to evaluate the performances of five search algorithms for prioritizing driving scenarios and defined objective functions (e.g., the probability of collision) for optimization. However, it does not account for the coverage of different modules in an ADS, which limits its prioritization capability. In our work, the test reduction process is automated based on frame vectorization calculated by the data in driving recordings. Furthermore, we evaluate our method on four ADS modules. In [10], Birchler et al. proposed a prioritization method to sort driving scenarios of lane-keeping systems. In their work, testing suites are maps with different shapes, turns, and other properties. They extracted features of maps and sorted them using genetic algorithms. In our work, testing suites are different driving recordings and the testing modules are perception, prediction, and planning modules. We split driving recordings to recording segments for further test reduction and prioritization.

7.2 Autonomous Driving Testing

The research of testing on ADSs mainly focused on the generation of rare or error-prone driving scenarios. In [54], Tian et al. proposed to apply different affine transformations to generate new testing images to evaluate the steering angle prediction of CNN-based driving models. In [60], Zhang et al. proposed to use Generative adversarial networks (GANs) to generate high-quality testing images such as driving scenes on rainy and snowy days. In [17], Deng et al. proposed to generate driving scenarios based on traffic rules to test driving models for speed prediction. In [62], Zhou et al. proposed to use adversarial attack methods [18, 19] to generate adversarial billboards in driving scenarios. In [23], Gambi et al. generated crash driving scenarios based on police reports.

Several search-based methods were proposed to find and generate driving scenarios leading to crash or deviation of ego-vehicles [1, 2, 9, 12, 20, 24, 25, 37]. While most work targeted on E2E driving models or Advanced driver-assistance systems (ADAS), some recent works started investigating on industry-level ADSs. In [26], Garcia et al. systematically analyzed bugs in Apollo and Autoware based on their Github repository commits. In [37], Li et al. proposed AV-FUZZER to search safety violations of Apollo in the urban driving environment using genetic algorithms [36]. In [20], Ebadi et al. searched testing scenarios for obstacle detection of Apollo. Our

Table 2: Effectiveness of the test reduction and prioritization methods in detecting faults

Test suite	ADS Module	Total faults	Covered faults	Top K					APFD				
				CH	RD	CC	SC	RSC	CH	RD	CC	SC	RSC
Cubetown	Traffic light	13	13 (100%)	6.0	4.26	5.5	5.0	**2.0**	0.43	0.50	0.39	0.50	**0.67**
Gomentum		19	19 (100%)	2.0	3.57	6.14	**1.0**	**1.0**	0.50	0.50	0.25	**0.82**	0.76
Shalun		78	72 (92.3%)	1.83	1.46	2.17	2.0	**1.33**	**0.56**	0.50	0.51	**0.56**	0.54
Total		**109**	**104 (95.4%)**	3.28	3.10	4.60	2.67	**1.44**	0.50	0.50	0.38	0.63	**0.66**
Cubetown	Obstacle	49	49 (100%)	1.89	2.47	**2.44**	3.0	3.0	0.56	0.50	0.53	**0.59**	**0.59**
Gomentum		28	28 (100%)	3.67	1.74	2.22	**1.0**	**1.0**	**0.51**	0.50	0.34	0.44	0.47
Shalun		27	27 (100%)	2.0	1.17	1.22	**1.0**	**1.0**	0.48	0.50	0.50	**0.53**	**0.53**
Total		**94**	**94 (100%)**	2.52	1.79	1.96	**1.67**	**1.67**	0.52	0.50	0.46	0.52	**0.53**
Cubetown	Planning	42	42 (100%)	3.0	1.38	**1.0**	**1.0**	**1.0**	0.37	0.50	0.58	**0.59**	**0.59**
Gomentum		10	10 (100%)	7.0	4.48	7.14	7.0	**2.0**	0.19	0.50	0.17	0.19	**0.81**
Shalun		64	64 (100%)	3.0	1.60	**1.0**	**1.0**	1.44	0.49	0.50	0.56	0.53	**0.58**
Total		**116**	**116 (100%)**	4.33	2.49	3.05	3.0	**1.48**	0.35	0.50	0.44	0.44	**0.66**
Cubetown	Prediction	34	34 (100%)	4.44	3.27	4.44	**2.22**	**2.22**	0.39	0.50	0.48	0.61	**0.62**
Gomentum		61	61 (100%)	1.22	1.13	1.11	**1.0**	**1.0**	0.50	0.50	0.50	0.50	**0.51**
Shalun		19	19 (100%)	4.63	5.82	6.50	3.75	**2.0**	0.55	0.50	0.45	0.61	**0.64**
Total		**114**	**114 (100.0%)**	3.43	3.41	4.02	2.32	**1.74**	0.48	0.50	0.48	0.57	**0.59**
Total	**Total**	**433**	**428(98.8%)**	3.39	2.70	3.41	2.42	**1.58**	0.46	0.50	0.44	0.54	**0.61**

work focus on test reduction and prioritization based on driving recordings.

7.3 Driving Scenario Identification

Driving scenario identification is the task of identifying driving scenarios and converting driving scenarios to vectors. After this, a few downstream works such as scenario clustering and anomaly detection can be implemented. Currently, only a few works in the software engineering community researched in this direction. In [52], Andrea et al. proposed a method to detect potential misbehaviors of an ADS. The main idea is to train a reconstructor to reconstruct the current driving scenario (single image or a sequence of images). If the reconstruction error exceeds a threshold, the driving scenario is most likely an anomalous scenario, and the autonomous vehicle may misbehave in the scenario. In this work, we identify driving scenarios based on the module outputs in the driving recording and convert each frame to a feature vector. We then use feature vectors for testing reduction and prioritization.

Beyond SE community, some existing work targeted driving scenario vectorization. In [29], Harder et al. proposed *scenario2vector* to characterize a driving image from three aspects *actors, actions*, and *scene*. Then based on the text description of the driving image, key elements belonging to the above three categories are extracted and converted to vectors or matrices. In our work, we also describe key elements of driving scenarios from these three aspects. However, we use them to define a driving scene schema to describe corresponding semantic information occurred in a driving scene. On the other hand, we do not use them together to characterize a driving scenario. For different testing modules, We extract data from its publish channel messages to encode semantic information described in the schema.

In [61], Zhao et al. proposed using Convolutional Neural Network (CNN) and Gated recurrent unit (GRU) network [16] to learn feature representations for driving videos. In [7], Balasubramanian et al. proposed to use self-supervised learning [32] to learn better feature representations for driving recordings. In our work, we treat a driving recording as a combination of multiple driving scenarios. We create and calculate frame-level features for driving recording segmentation.

8 THREATS TO VALIDITY

We discuss threats to validity from three aspects external validity, construct validity, and internal validity proposed in [57].

External Validity: The external validity is generalizability of the proposed method. In this work, we conduct experiments on an industry-level ADS Apollo to evaluate the effectiveness of the proposed test reduction and prioritization methods. These methods are also available for other multi-module ADSs that apply publish-subscribe communication mechanism among modules and save module outputs in driving recordings (e.g., Autoware [34] that adopts ROS [3]).

Internal Validity: The internal validity is that in ADS modules there are many non-deterministic algorithms, which makes module outputs have slight differences when the module runs on the same input multiple times [35]. This problem may introduce uncertain faults when we compare the outputs of the old and new versions of SUTs. To solve the problem, We use a sliding window-based smoothing algorithm to remove noises. Furthermore, we set the threshold to evaluate inconsistent outputs as 10% to ensure the detected inconsistent output is caused by the new version of SUT. In this work, we inject mutants into Apollo source code to create regression benchmarks. The effectiveness of mutation testing are suitable for software testing experimentation [5, 33]. In this way, we explore common faults in different ADS modules and ensure every time we only change one file in an ADS module. We do not use commits in Apollo Github commit history as regression benchmarks because most commits are relevant to the update of multiple modules.

Construct Validity: The construct validity is that we conducted experiments on Apollo 5.0, not Apollo 6.0 that is used in other papers [37]. The reason is that now for Apollo 6.0, the camera-based perception module does not work and LiDAR-based perception is unstable in SVL[6]. In this work, we evaluated our methods on three maps. We tested two other maps (i.e., 'Borregas Avenue' and 'San Francisco') but we found that Apollo is not able to control the ego-vehicle stably. This issue has been reported by Apollo users [7]. Therefore, we excluded these two maps from the experiment. We will keep tracking these issues and try to include more maps for experiments in the future work. For mutant rejection, we did not directly mutate DL models in Apollo because the models are encapsulated in binary files. Alternatively, we mutated C++ files that are related to DL model configuration and model input/output to simulate bugs in DL models. For example, by mutating the output of the obstacle detection model, we can simulate incorrect model predictions.

9 CONCLUSION

This paper proposes STRaP to reduce the cost of regression testing in industry-scale multi-module ADSs. Since such ADSs are largely built upon publish-subscribe mechanism, we define a driving scene schema and for different modules under test, we extract frame data from its publish channels and encode semantic information described in the schema as frame feature vectors. Based on such feature vectorization, we propose a test reduction algorithm to split original driving recordings to recording segments, each of which reflects a unique driving scenario. We also propose semantic coverage based and rarity based test prioritization to order the reduced recording segments as test inputs for ADS modules. In our work, testing suites are different driving recordings and the test modules are perception, prediction, and planning modules in ADSs. In our evaluation using an industry-level multi-module ADS and three road maps in diverse regression testing settings, we prove STRaP is able to significantly reduce the length of original driving recordings (34% to 77%), has promising fault coverage of 98.8%, and achieves 1.58 for Top K and 0.61 for APFD beating the state-of-the-art baselines. This work is a pioneering work to utilize frame-level features for driving recording segmentation. With manually defined driving scene feature schema, we have achieved promising results in both test reduction and prioritization. As future directions for SE community, based on this open-sourced work, automated feature extraction from driving recording can be explored along with more intelligent smoothing algorithms.

10 ACKNOWLEDGEMENTS

This work is in part supported by an Australian Research Council (ARC) Discovery Project (DP210102447), an ARC Linkage Project (LP190100676), and a DATA61 project (Data61 CRP C020996).

REFERENCES

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 1016–1026.

[2] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 143–154.

[3] Afsoon Afzal, Claire Le Goues, Michael Hilton, and Christopher Steven Timperley. 2020. A study on challenges of testing robotic systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 96–107.

[4] KK Aggrawal, Yogesh Singh, and Arvinder Kaur. 2004. Code coverage based technique for prioritizing test cases for regression testing. *ACM SIGSOFT Software Engineering Notes* 29, 5 (2004), 1–4.

[5] James H Andrews, Lionel C Briand, and Yvan Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *Proceedings of the 27th international conference on Software engineering*. 402–411.

[6] ApolloAuto. 2021. Apollo. https://bit.ly/2E3vWyo.

[7] Lakshman Balasubramanian, Jonas Wurst, Michael Botsch, and Ke Deng. 2021. Traffic Scenario Clustering by Iterative Optimisation of Self-Supervised Networks Using a Random Forest Activation Pattern Similarity. In *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 682–689.

[8] Rebecca Bellan. 2021. Waymo launches robotaxi service in San Francisco. http://shorturl.at/dltFY.

[9] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2016. 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*. 63–74.

[10] Christian Birchler, Sajad Khatiri, Pouria Derakhshanfar, Sebastiano Panichella, and Annibale Panichella. 2021. Automated test cases prioritization for self-driving cars in virtual environments. *arXiv preprint arXiv:2107.09614* (2021).

[11] Neal E. Boudette. 2017. Tesla's Self-Driving System Cleared in Deadly Crash. https://nyti.ms/2iZ93SL.

[12] Alessandro Calò, Paolo Arcaini, Shaukat Ali, Florian Hauer, and Fuyuki Ishikawa. 2020. Generating avoidable collision scenarios for testing autonomous driving systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 375–386.

[13] Tsong Yueh Chen and Man Fai Lau. 1970. Heuristics towards the optimization of the size of a test suite. *WIT Transactions on Information and Communication Technologies* 14 (1970).

[14] Tsong Yueh Chen and Man Fai Lau. 1998. A new heuristic for test suite reduction. *Information and Software Technology* 40, 5-6 (1998), 347–354.

[15] Tsong Yueh Chen and Man Fai Lau. 1998. A simulation study on some heuristics for test suite reduction. *Information and Software Technology* 40, 13 (1998), 777–787.

[16] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[17] Yao Deng, Guannan Lou, Xi Zheng, Tianyi Zhang, Miryung Kim, Huai Liu, Chen Wang, and Tsong Yueh Chen. 2021. BMT: behavior driven development-based metamorphic testing for autonomous driving models. In *2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET)*. IEEE, 32–36.

[18] Yao Deng, Tiehua Zhang, Guannan Lou, Xi Zheng, Jiong Jin, and Qing-Long Han. 2021. Deep learning-based autonomous driving systems: a survey of attacks and defenses. *IEEE Transactions on Industrial Informatics* 17, 12 (2021), 7897–7912.

[19] Yao Deng, Xi Zheng, Tianyi Zhang, Chen Chen, Guannan Lou, and Miryung Kim. 2020. An analysis of adversarial attacks and defenses on autonomous driving models. In *2020 IEEE international conference on pervasive computing and communications (PerCom)*. IEEE, 1–10.

[20] Hamid Ebadi, Mahshid Helali Moghadam, Markus Borg, Gregory Gay, Afonso Fontes, and Kasper Socha. 2021. Efficient and Effective Generation of Test Cases for Pedestrian Detection-Search-based Software Testing of Baidu Apollo in SVL. In *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 103–110.

[21] Sebastian Elbaum, Alexey Malishevsky, and Gregg Rothermel. 2001. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*. IEEE, 329–338.

[22] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. 2000. Prioritizing test cases for regression testing. In *Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*. 102–112.

[23] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 257–267.

[24] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. 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*. 318–328.

[6]https://www.svlsimulator.com/docs/system-under-test/apollo6-0-instructions/
[7]https://github.com/ApolloAuto/apollo/issues/14142https://github.com/ApolloAuto/apollo/issues/14282

[25] Alessio Gambi, Marc Müller, and Gordon Fraser. 2019. Asfault: Testing self-driving car software using search-based procedural content generation. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 27–30.

[26] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, Chen, and Qi Alfred. 2020. A comprehensive study of autonomous vehicle bugs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 385–396.

[27] The Guardian. 2021. Driverless electric bus hits the road in Spanish city of Málaga. shorturl.at/ovwH3.

[28] Junyao Guo, Unmesh Kurup, and Mohak Shah. 2019. Is it safe to drive? an overview of factors, metrics, and datasets for driveability assessment in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems* 21, 8 (2019), 3135–3151.

[29] Aron Harder, Jaspreet Ranjit, and Madhur Behl. 2021. Scenario2Vector: scenario description language based embeddings for traffic situations. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*. 167–176.

[30] M Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. 1993. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2, 3 (1993), 270–285.

[31] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering* 37, 5 (2010), 649–678.

[32] Longlong Jing and Yingli Tian. 2020. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* 43, 11 (2020), 4037–4058.

[33] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 654–665.

[34] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. 2018. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 287–296.

[35] Philip Koopman and Michael Wagner. 2016. Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety* 4, 1 (2016), 15–24.

[36] Manoj Kumar, Mohammad Husain, Naveen Upreti, and Deepti Gupta. 2010. Genetic algorithm: Review and application. *Available at SSRN 3529843* (2010).

[37] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding safety violations in autonomous driving systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 25–36.

[38] Guannan Lou, Yao Deng, Xi Zheng, Tianyi Zhang, and Mengshi Zhang. 2021. An investigation into the state-of-the-practice autonomous driving testing. *arXiv preprint arXiv:2106.12233* (2021).

[39] Chengjie Lu, Huihui Zhang, Tao Yue, and Shaukat Ali. 2021. Search-Based Selection and Prioritization of Test Scenarios for Autonomous Driving Systems. In *International Symposium on Search Based Software Engineering*. Springer, 41–55.

[40] Xue-ying Ma, Bin-kui Sheng, and Cheng-qing Ye. 2005. Test-suite reduction using genetic algorithm. In *International Workshop on Advanced Parallel Processing Technologies*. Springer, 253–262.

[41] Alexis C. Madrigal. 2017. INSIDE WAYMO'S SECRET WORLD FOR TRAINING SELF-DRIVING CARS. https://shorturl.at/xDEF2.

[42] Nashat Mansour and Khalid El-Fakih. 1999. Simulated annealing and genetic algorithms for optimal regression testing. *Journal of Software Maintenance: Research and Practice* 11, 1 (1999), 19–34.

[43] Alan Ohnsman. 2021. Robotruck Startup Gatik Making Delivery Runs For Walmart Without Humans At The Wheel. shorturl.at/xBGY0.

[44] Đorđe Petrović, Radomir Mijailović, and Dalibor Pešić. 2020. Traffic accidents with autonomous vehicles: type of collisions, manoeuvres and errors of conventional vehicles' drivers. *Transportation research procedia* 45 (2020), 161–168.

[45] New York Post. 2022. Tesla recalls 579K more cars — fourth over the last 2 weeks. shorturl.at/etGMQ.

[46] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.

[47] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE, 1–6.

[48] Gregg Rothermel, Mary Jean Harrold, Jeffery Ostrin, and Christie Hong. 1998. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE, 34–43.

[49] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. 1999. Test case prioritization: An empirical study. In *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*. IEEE, 179–188.

[50] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering* 27, 10 (2001), 929–948.

[51] Praveen Ranjan Srivastava. 2008. Test case prioritization. *Journal of Theoretical & Applied Information Technology* 4, 3 (2008).

[52] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 359–371.

[53] Yun Tang, Yuan Zhou, Tianwei Zhang, Fenghua Wu, Yang Liu, and Gang Wang. 2021. Systematic testing of autonomous driving systems using map topology-based scenario classification. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1342–1346.

[54] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[55] Waymo. 2021. *Waymo Safety Report*. Technical Report. Waymo.

[56] Kale Wiggers. 2020. Waymo's driverless cars were involved in 18 accidents over 20 months. https://venturebeat.com/2020/10/30/waymos-driverless-cars-were-involved-in-18-accidents-over-20-month/.

[57] C. Wohlin, P. Runeson, M. Hst, M. Ohlsson, B. Regnell, and A. Wessln. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[58] W Eric Wong, Joseph R Horgan, Saul London, and Hiralal Agrawal. 1997. A study of effective regression testing in practice. In *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering*. IEEE, 264–274.

[59] S. Yoo and M. Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. *Softw. Test. Verif. Reliab.* 22, 2 (mar 2012), 67–120.

[60] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 132–142.

[61] Jinxin Zhao, Jin Fang, Zhixian Ye, and Liangjun Zhang. 2021. Large Scale Autonomous Driving Scenarios Clustering with Self-supervised Feature Extraction. In *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 473–480.

[62] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 347–358.