University of Sheffield

(AVS–Fast Abstract) Towards Simplification of Failure Scenarios for Machine Learning-enabled Autonomous Systems

Donghwan Shin, Sanjeetha Pennada
University of Sheffield

05 July 2024 | AVS 2024

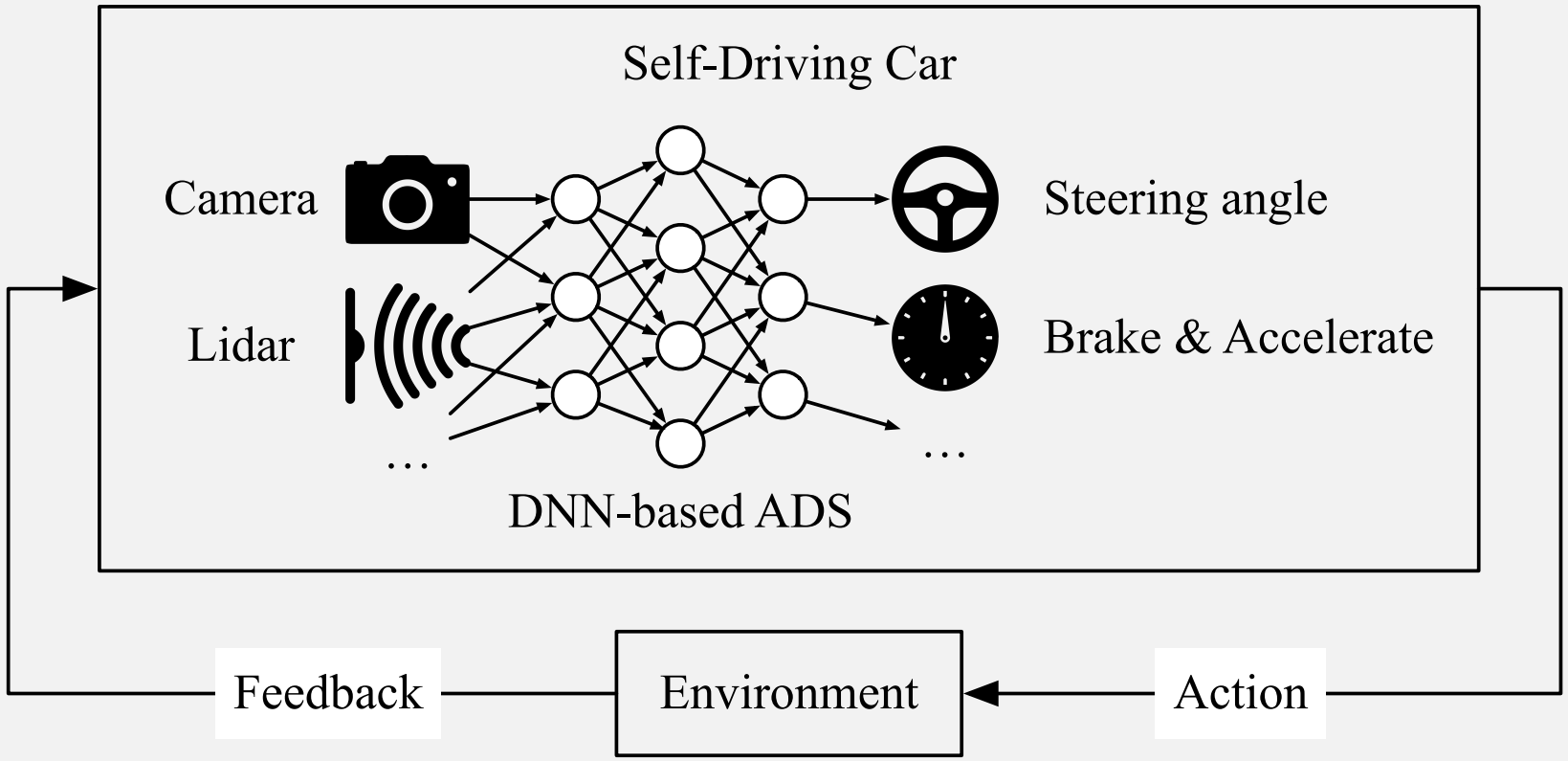
Project Overview

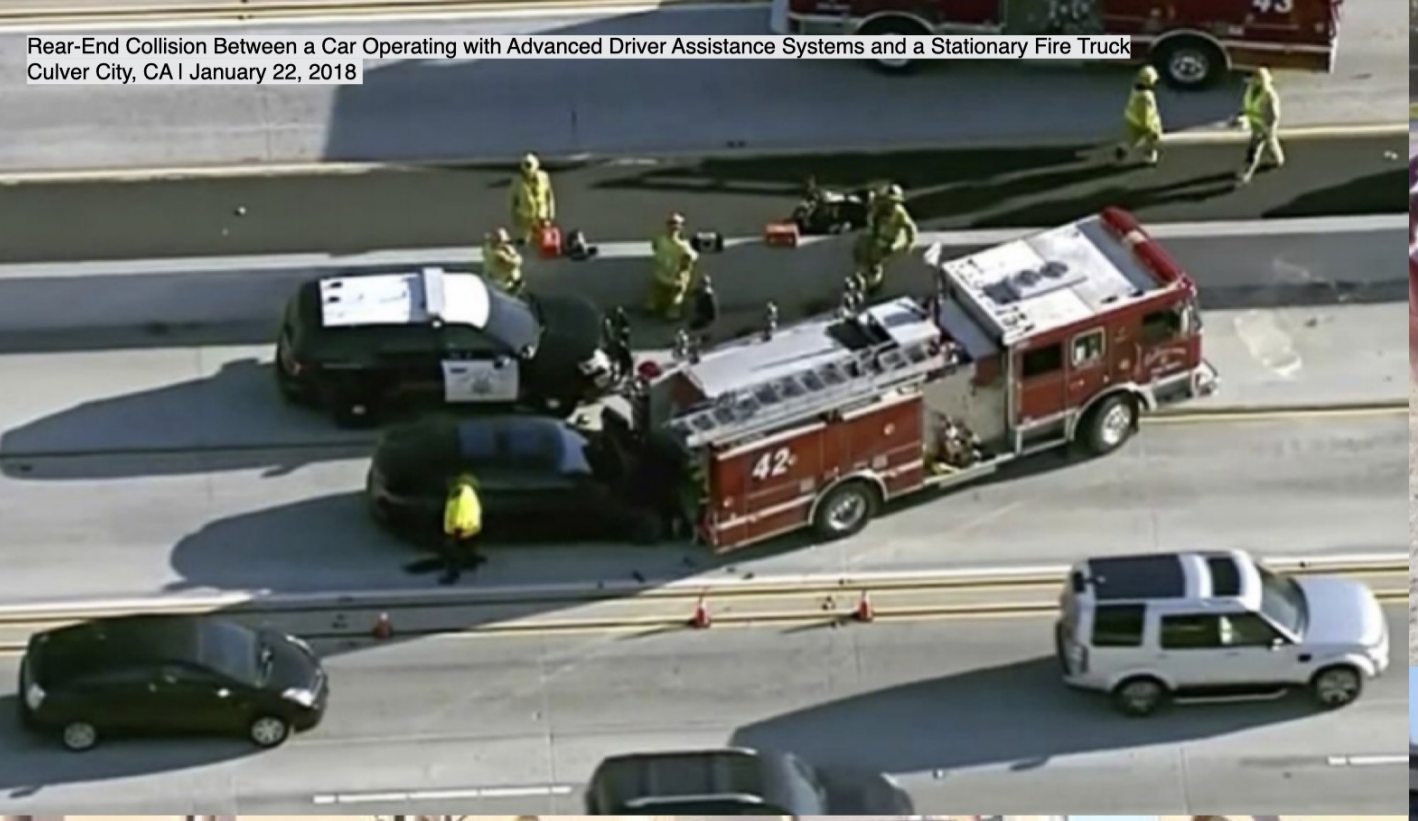
Engineering and Physical Sciences Research Council

- **Title**: SimpliFaiS: Simplification of Failure Scenarios for Machine Learning-enabled Autonomous Systems
- **Duration**: 30 months (Jan 2024 – June 2026)
- **Funding**: EPSRC (£370k)
- **Team**
  - **PI**: Donghwan Shin (University of Sheffield, UK)
  - **Post-doc RA**: Sanjeetha Pennada (University of Sheffield, UK)
  - **Academic collaborators**: Gordon Fraser (University of Passau, Germany), Shin Yoo (KAIST, South Korea)
  - **Industry collaborator**: Nam Hee Lee (SolutionLink, South Korea)

Autonomous System?

An intelligent system designed to ***perceive*** and ***act*** on the surrounding physical environment and work *for extended periods of time without human intervention.*





Rear-End Collision Between a Car Operating with Advanced Driver Assistance Systems and a Stationary Fire Truck
Culver City, CA | January 22, 2018



Collision Between Car Operating with Partial Driving Automation and Truck-Tractor Semitrailer
Delray Beach, FA | March 1, 2019



Low-Speed Collision Between Truck-Tractor and Autonomous Shuttle
Las Vegas, NV | November 18, 2017



Collision Between a Sport Utility Vehicle Operating With Partial Driving Automation and a Crash Attenuator
Mountain View, CA | March 23, 2018

How to make it safer and more reliable?

Scenario-based testing

- Let's consider a ***concise sequence of scenes***, including actions and events, instead of a long and redundant natural driving environment.

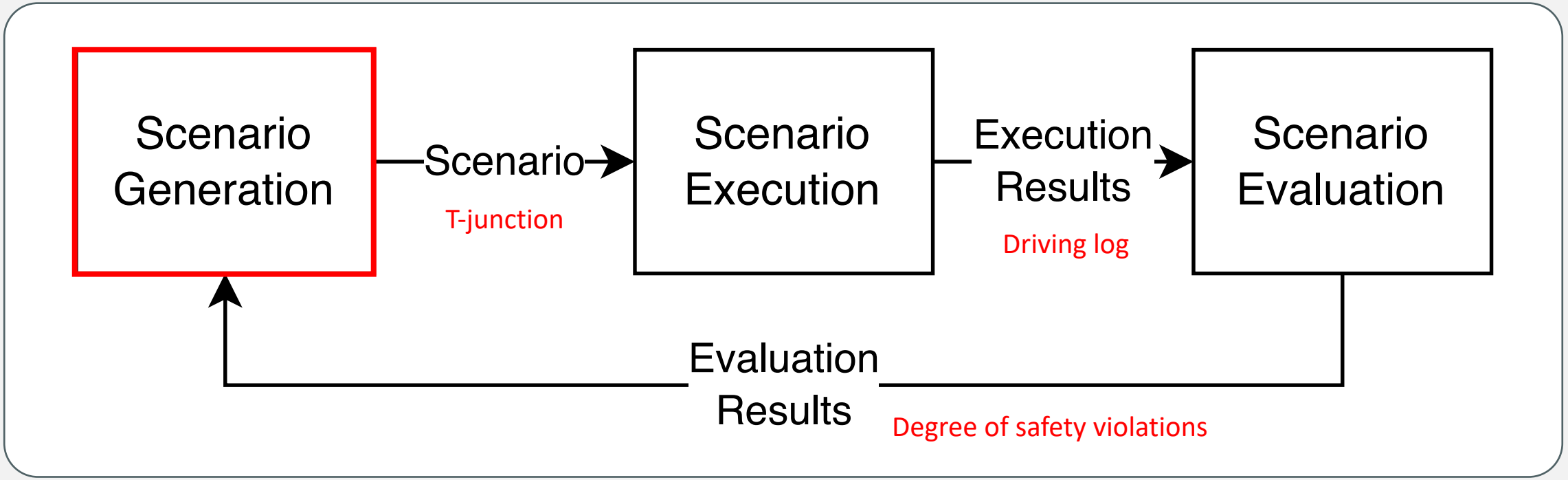
For example, in the **automotive domain**:

- **Scenario**: a sequence of scenes usually including the automated driving system and its interactions in the process of driving.
- **Scene**: a snapshot of all entities (including the ADS, static entities, dynamic entities) and their relationships.
- **Static entities**: entities that do not experience state changes.
    - E.g., roads, trees, buildings, traffic signs, parked vehicles.
- **Dynamic entities**: entities that experience state changes.
    - E.g., moving vehicles, pedestrians, traffic lights.

(Example Scenario) T-junction: Turn right without a traffic light



Overview of Scenario-based Testing



Q. How can we *effectively* and *efficiently* generate (discover) critical scenarios?

Various Scenario Generation Approaches

ICSE 2022

Efficient Online Testing for D
Surrogate-Assisted and Man

Fitash Ul Haq
University of Luxembourg
Luxembourg
fitash.ulhaq@uni.lu

Donghwa
University of L
Luxemb
donghwan.sh

ABSTRACT

With the recent advances of Deep Neural Networks (DNNs) in real-world applications, such as Automated Driving Systems (ADS) for self-driving cars, ensuring the reliability and safety of such DNN-enabled Systems emerges as a fundamental topic in software testing. One of the essential testing phases of such DNN-enabled systems is *online testing*, where the system under test is embedded into a specific and often simulated application environment (e.g., a driving environment) and tested in a closed-loop mode in interaction with the environment. However, despite the importance of online testing for detecting safety violations, automatically generating new and diverse test data that lead to safety violations presents the following challenges: (1) there can be many safety requirements to be considered at the same time, (2) running a high-fidelity simulator is often very computationally-intensive, and (3) the space of all possible test data that may trigger safety violations is too large to be exhaustively explored.

In this paper, we address the challenges by proposing a novel approach, called SAMOTA (Surrogate-Assisted Many-Objective Testing Approach), extending existing *many-objective* search algorithms for test suite generation to efficiently utilize *surrogate models* that mimic the simulator, but are much less expensive to run. Empirical evaluation results on Pylot, an advanced ADS composed of multiple DNNs, using CARLA, a high-fidelity driving simulator, show that SAMOTA is significantly more effective and efficient at detecting unknown safety requirement violations than state-of-the-art many-objective test suite generation algorithms and random search. In other words, SAMOTA appears to be a key enabler technology for online testing in practice.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**; **Search-based software engineering**.

KEYWORDS

DNN testing, online testing, many-objective search, surrogate-

ICSE 2023

Many-Objective Reinforcement Learning for Online Testing of DNN-Enabled Systems

Fitash Ul Haq
University of Luxembourg
Luxembourg
fitash.ulhaq@uni.lu

Donghwan Shin*
University of Sheffield
Sheffield, United Kingdom
d.shin@sheffield.ac.uk

Lionel C. Briand
University of Luxembourg
Luxembourg
University of Ottawa
Ottawa, Canada
lionel.briand@uni.lu

***Abstract*—Deep Neural Networks (DNNs) have been widely used to perform real-world tasks in cyber-physical systems such as Autonomous Driving Systems (ADS). Ensuring the correct behavior of such DNN-Enabled Systems (DES) is a crucial topic. Online testing is one of the promising modes for testing such systems with their application environments (simulated or real) in a closed loop, taking into account the continuous interaction between the systems and their environments. However, the environmental variables (e.g., lighting conditions) that might change during the systems' operation in the real world, causing the DES to violate requirements (safety, functional), are often kept constant during the execution of an online test scenario due to the two major challenges: (1) the space of all possible scenarios to explore would become even larger if they changed and (2) there are typically many requirements to test simultaneously.**

In this paper, we present MORLOT (Many-Objective Reinforcement Learning for Online Testing), a novel online testing approach to address these challenges by combining Reinforcement Learning (RL) and many-objective search. MORLOT leverages RL to incrementally generate sequences of environmental changes while relying on many-objective search to determine the changes so that they are more likely to achieve any of the uncovered objectives. We empirically evaluate MORLOT using CARLA, a high-fidelity simulator widely used for autonomous driving research, integrated with Transfuser, a DNN-enabled ADS for end-to-end driving. The evaluation results show that MORLOT is significantly more effective and efficient than alternatives with a large effect size. In other words, MORLOT is a good option to test DES with dynamically changing environments while accounting for multiple safety requirements.

***Index Terms*—DNN Testing, Reinforcement learning, Many objective search, Self-driving cars, Online testing**

I. Introduction

Deep Neural Networks (DNNs) have been widely used to perform various tasks, such as object classification [1], with other vehicles). Therefore, ensuring the correct behavior of such DNN-Enabled Systems (DES) has emerged as a fundamental problem of software testing.

Online testing [4] is one of the promising modes of DES testing that accounts for the closed-loop interaction between the DES under test and its environment. In online testing, the DES under test is embedded into the application environment (e.g., a simulated driving environment for ADS) and is monitored to detect the violations of safety and functional requirements during the closed-loop interaction. Such interaction helps account for the accumulation of (possibly small) errors over time, eventually leading to requirements violations that often cannot be detected by offline testing [5]. As a result, online testing has been actively applied in many testing approaches recently [6, 7, 8, 9, 10].

However, existing online testing approaches for DES exhibit at least one of two critical limitations. First, they do not account for the fact that there are often many safety and functional requirements, possibly independent of each other, that must be considered together in practice. Though one could simply repeat an existing test approach for individual requirements, it is inefficient due to its inability to dynamically distribute the test budget (e.g., time) over many requirements according to the feasibility of requirements violations. For example, if one of the requirements cannot be violated, the pre-assigned budget for this requirement would simply be wasted. Furthermore, dividing the limited test budget across many requirements may result in too small a budget for testing individual requirements thoroughly. Second, they do not vary dynamic environmental elements, such as neighboring vehicles and weather conditions, during test case (i.e., test

See below for more details:

- Jian Sun, He Zhang, Huajun Zhou, Rongjie Yu, and Ye Tian. 2022. ***Scenario-Based Test Automation for Highly Automated Vehicles: A Review and Paving the Way for Systematic Safety Assurance***. *IEEE Trans. Intell. Transport. Syst.* 23, 9 (September 2022), 14088–14103. https://doi.org/10.1109/TITS.2021.3136353
- Xinhai Zhang, Jianbo Tao, Kaige Tan, Martin Törngren, José Manuel Gaspar Sánchez, Muhammad Rusyadi Ramli, Xin Tao, Magnus Gyllenhammar, Franz Wotawa, Naveen Mohan, Mihai Nica, and Hermann Felbinger. 2023. ***Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study***. *IIEEE Trans. Software Eng.* 49, 3 (March 2023), 991–1026. https://doi.org/10.1109/TSE.2022.3170122
- Ziyuan Zhong, Yun Tang, Yuan Zhou, Vania de Oliveira Neves, Yang Liu, and Baishakhi Ray. 2021. ***A Survey on Scenario-Based Testing for Automated Driving Systems in High-Fidelity Simulation***. Retrieved April 9, 2023 from http://arxiv.org/abs/2112.00964

and so on…

Failure scenarios found, and then?

- Given *failure scenarios* (i.e., scenarios that trigger the failure of the system under test), we need to ***investigate*** and ***understand*** them to identify the ***root causes of the failures***.
- However, **realistic failure scenarios** are becoming more complex, making it more challenging to investigate and understand them.
  - A failure scenario may have ***hundreds*** **of static and dynamic entities**.

*Let's **simplify failure scenarios** automatically!*

Problem Definition

Definition 1. MLAS Failure Scenario Simplification

For a given MLAS M, its failure scenario S that includes a set of scenario entities, and the failure F of M induced by S, the problem of *MLAS failure scenario simplification* is to identify a simplified failure scenario S' that includes the minimum set of scenario entities in S while inducing F.

For example,

Input failure scenario = <**car1**, **car2**, …, building1, building2, …, **traffic-sign1**, …>

Output simplified scenario = <**car1**, **car2**, **traffic-sign1**>, inducing the same failure as the input.

Key Challenges

- **Multiple sets of scenario entities may induce the same failure**, especially due to the nonlinear nature of ML models and their interactions.
- **Too large problem space**, considering *all possible sets* of scenario entities that might induce a given failure.
- **No internal information is available** since MLAS often contains ML components developed by 3rd parties.
- **Long simulation times** due to high-fidelity simulations required for testing MLAS in their application environments (e.g., driving).

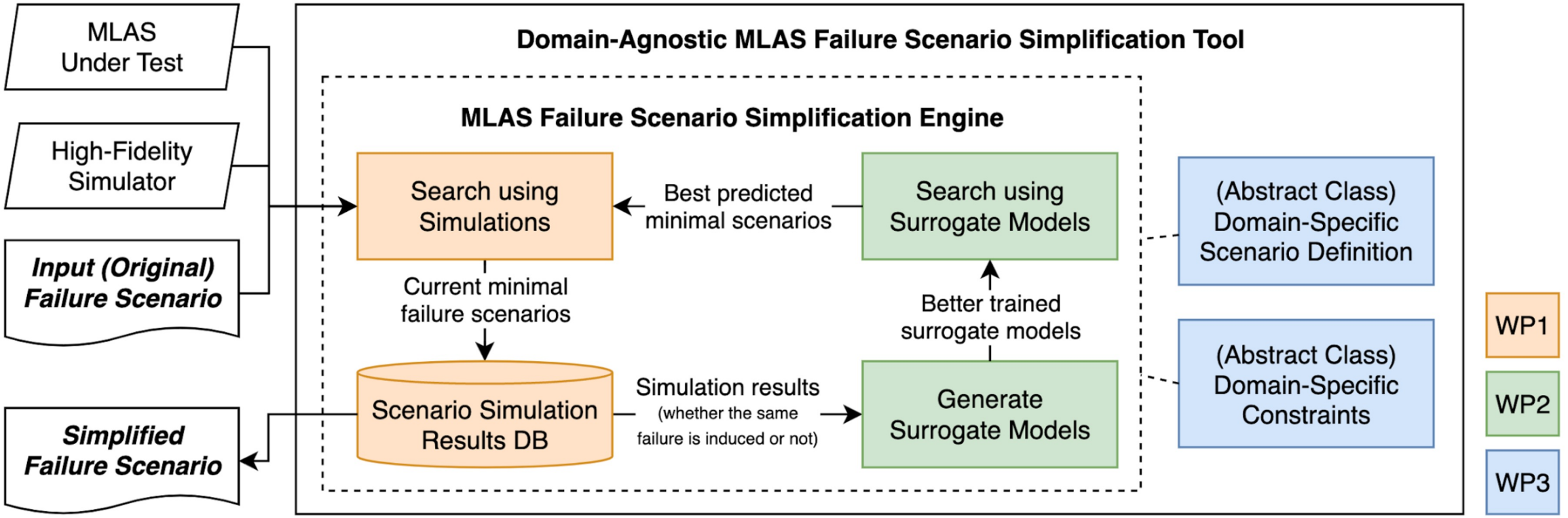
Research Vision

We can address the problem by carefully combining **Search-Based Software Engineering** (SBSE) and **Surrogate-Assisted Optimisation** (SAO).

- **SBSE** aims to identify near-optimal solutions using metaheuristics effectively.
- **SAO** aims to reduce the cost of expensive fitness evaluations significantly.
- We already have successfully used the combination of SBSE and SAO for the problem of MLAS failure scenario generation*.

* Haq, Fitash Ul, Donghwan Shin, and Lionel Briand. "Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization." 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). IEEE, 2022.

Overview of the proposed approach



Ongoing Work

Implemented a framework *for replay-with-modification* based on CARLA (simulator).

Implemented a basic simplification algorithm (delta debugging) as a baseline.

Study candidate scenario representations and search operations (e.g., mutation).

```
Initial Test Input: [6624907388525486529, 5590853173915701067, 11595376130058892747, 11261989655324469416, 10827724883280509631, 18345568482389893230, 1580899482231826983, 1760263306195875
505, 12193930106007007199, 8965582157476607102, 2012950392265273169, 8202849075528332341, 6570133562374327036, 8588501765608435261, 3725429243769496776, 12942124512259043244, 4899373996816
986076, 7783727404168196753, 1601677913125771255, 4983866454221917867, 793883526509223864, 11096794554960802340, 18125725892881901402, 9314362118298296807, 10311091484298245286, 1240570754
4884411272, 14597101749787311194, 1437529571490662170, 160167791312577125, 12317005202958092029, 2630081664371749371, 2795511079265638899, 17619596962267012938, 12253574462247490343, 37095
60905934411224, 97, 98, 99, 100, 73, 72, 71, 70, 69, 68, 61, 50, 40, 24, 26, 25, 27, 130, 456, 440, 305, 301, 347, 122, 165, 166, 211, 212, 215, 217, 218, 242, 243, 244, 312, 323, 334, 344
, 343, 345, 162, 163, 164, 241, 306, 307, 346, 'rain', 'fog', 'cloudy', 'night', 'Bike', 'Pedestrian', 'Four NPCs']
Step | Subsequence                                          | Error Triggered
-----------------------------------------------------------
1    | [68, 61, 50, 40, 24, 26, 25, 27, 130, 456, 440, 305, 301, 347, 122, 165, 166, 211, 212, 215, 217, 218, 242, 243, 244, 312, 323, 334, 344, 343, 345, 162, 163, 164, 241, 306, 307, 34
6, 'rain', 'fog', 'cloudy', 'night', 'Bike', 'Pedestrian', 'Four NPCs']| PASS
2    | [242, 243, 244, 312, 323, 334, 344, 343, 345, 162, 163, 164, 241, 306, 307, 346, 'rain', 'fog', 'cloudy', 'night', 'Bike', 'Pedestrian', 'Four NPCs']| PASS
3    | [164, 241, 306, 307, 346, 'rain', 'fog', 'cloudy', 'night', 'Bike', 'Pedestrian', 'Four NPCs']| PASS
4    | ['fog', 'cloudy', 'night', 'Bike', 'Pedestrian', 'Four NPCs']| PASS
5    | ['Bike', 'Pedestrian', 'Four NPCs']                  | FAIL
6    | ['fog', 'cloudy', 'night']                           | FAIL
7    | ['cloudy', 'night', 'Bike', 'Pedestrian', 'Four NPCs']| PASS
8    | ['night', 'Bike', 'Pedestrian', 'Four NPCs']| PASS
9    | ['Pedestrian', 'Four NPCs']                          | FAIL
10   | ['night', 'Bike']                                    | FAIL
11   | ['Bike', 'Pedestrian', 'Four NPCs']                  | FAIL
12   | ['night', 'Pedestrian', 'Four NPCs']                 | PASS
13   | ['Pedestrian', 'Four NPCs']                          | FAIL
14   | ['night', 'Four NPCs']                               | PASS
15   | ['Four NPCs']                                        | FAIL
16   | ['night']                                            | FAIL
Minimal failure-inducing input: ['night', 'Four NPCs']
```

Example simplification result using delta debugging

Preliminary Results

Using CARLA (simulator) and TransFuser (ADS),

- Removing buildings does not affect the failure.
- In some scenarios, just switching the weather conditions affects the failure.
- In some scenarios, removing shadow affects the failure.

Notice that the results are highly dependent on the specific ADS under test.

- Different ADSs would have different failures and sets of failure-inducing entities.

Future Work

Initial setup

- Different failure scenarios, ADSs to test, and baseline approaches.

Search-based simplification

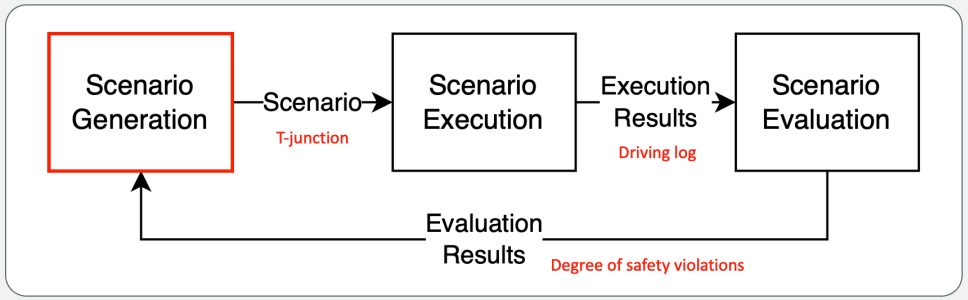
- Single-objective vs. Multi-objective
- Effective search operations (e.g., mutation, crossover)
- Predict the likelihood of inducing the same failure

Surrogate-assisted simplification

- Design surrogate models that mimic the simplification result
- Develop surrogate-assisted search techniques

Summary

Overview of Scenario-based Testing



Q. How can we *effectively* and *efficiently* generate (discover) critical scenarios?

Problem Definition

Definition 1. MLAS Failure Scenario Simplification

For a given MLAS M, its failure scenario S that includes a set of scenario entities, and the failure F of M induced by S, the problem of *MLAS failure scenario simplification* is to identify a simplified failure scenario S' that includes the minimum set of scenario entities in S while inducing F.

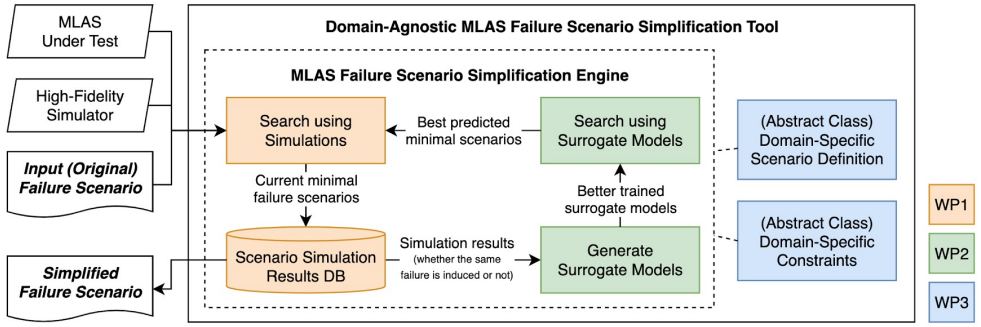
For example,

Input failure scenario = <**car1**, **car2**, …, building1, building2, …, **traffic-sign1**, …>

Output simplified scenario = <**car1**, **car2**, **traffic-sign1**>, inducing the same failure as the input.

10

Overview of the proposed approach



13



Donghwan Shin

Lecturer (Assistant Professor)

d.shin@sheffield.ac.uk

https://dshin.info