SBFT Tool Competition 2026 - CPS-SDC Regression Testing Track

Prakash Aryan
University of Bern
Bern, Switzerland

Christian Birchler
University of Bern
Bern, Switzerland

Tommaso Fulcini
Politecnico di Torino
Turin, Italy

Luigi Libero Lucio Starace
Università di Napoli Federico II
Naples, Italy

Sebastiano Panichella
University of Bern &
AI4I - The Italian Institute of Artificial
Intelligence for Industry
Bern, Switzerland

Abstract

This edition of the tool competition on regression testing of self-driving cars (SDCs) at the International Workshop on Search-Based and Fuzz Testing aims to provide a platform for software testers to submit their tools addressing the test prioritization problem for simulation-based testing of SDCs, which is considered an emerging and vital domain.

The competition provides an advanced software platform and representative case studies to ease participants' entry into SDC regression testing, enabling them to develop their initial test generation, selection, and prioritization tools for SDCS. In this second edition, the competition includes one tool. The tool was evaluated using (regression) metrics for test prioritization, as well as compared with a baseline random approach. This paper provides an overview of the competition, detailing its context, framework, participating tools, evaluation methodology, and key findings.

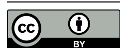
CCS Concepts

• **Software and its engineering → Software verification and validation**; **Search-based software engineering**; • **Computer systems organization → Embedded and cyber-physical systems**.

Keywords

Software engineering, Software testing, Regression testing, Autonomous systems, Simulation-based testing, Cyber-physical systems

ACM Reference Format:
Prakash Aryan, Christian Birchler, Tommaso Fulcini, Luigi Libero Lucio Starace, and Sebastiano Panichella. 2026. SBFT Tool Competition 2026 - CPS-SDC Regression Testing Track. In *19th Search-Based and Fuzz Testing Workshop (SBFT '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3786155.3795699



This work is licensed under a Creative Commons Attribution 4.0 International License.
SBFT '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2387-2/2026/04
https://doi.org/10.1145/3786155.3795699

1 Introduction

Software testing for cyber-physical systems (CPSs), and self-driving cars (SDCs) in particular, poses different challenges from conventional unit and integration testing [6, 16]. A significant challenge arises from the dependence on simulation-based testing, which frequently involves lengthy executions, requires substantial hardware resources, and is influenced by non-determinism [6, 8] as well as the widely recognized disparity between simulated outcomes and actual real-world behavior, also known as *reality gap* [13]. Consequently, the expenses associated with running test suites, considering both time and computational resources, emerge as a key constraint in practical applications [13, 14, 12]. This underscores the importance of utilizing simulation resources in the most efficient way, thereby driving research into methodologies that can enhance fault detection while maintaining the same number of tests executed.

In this context, the self-driving car (SDC) Testing Competition offers a platform for software testing researchers to ease their entry into the SDC domain. The goal of the competition is to motivate them to apply automated testing techniques in a relevant domain, such as the autonomous vision-based SDC navigation systems. Given the aforementioned context, the competition focuses on *test prioritization* – ordering test cases to maximize early fault detection in SDCs and reduce the cost of test automation. However, effective test prioritization is challenging, especially when prioritization factors are non-trivial, and remains an open problem in software engineering research. Test prioritization is one aspect of the higher-level context of *regression testing*, which includes, next to test prioritization, also *test selection* and *test minimization* [15].

Test prioritization is the process of ordering test cases to detect faults as early as possible during test execution. In the context of simulation-based testing for SDCs with long-running test cases, we prioritize test cases to reveal as many faults as possible early in the execution sequence, maximizing the value obtained within limited testing budgets.

2 Methodology

We provided on GitHub an interface specification to the public. The GitHub repository included the random prioritization baseline approach for the participants to assess their tool locally before submitting. A sample benchmark was also available for the participants. Interested participants in the competition could submit their implementations of the interface as pull requests. We collected the implementations and performed our experiments; we

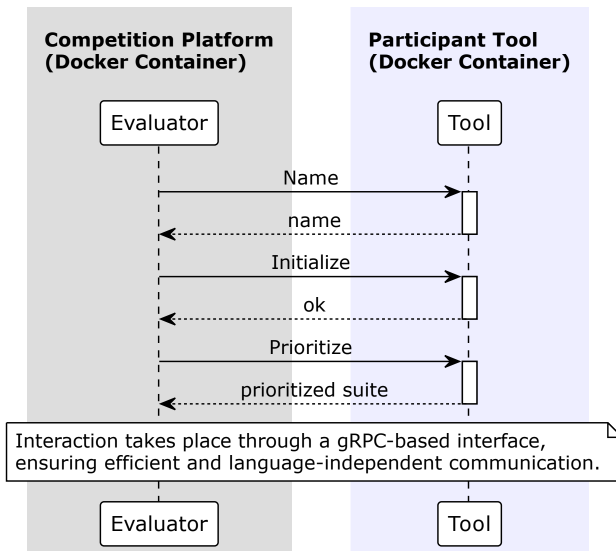


Figure 1: Interaction protocol between the Competition Platform and the participant-provided tool.

executed them with a benchmark that was not shared with the participants. To ensure a fair comparison among competing tools and facilitate their development, we have provided participants with an open-source, extensible infrastructure on GitHub [1].

2.1 Competition Platform

2.1.1 Interface. For this edition of the competition, participants were required to implement a standardized interface that enables interaction with the Competition Platform. The interface specifies the complete lifecycle of a tool, including an optional initialization phase and the generation of a full prioritization of the test suite (see Figure 1).

As done in past editions of the challenge, the interface is formally defined using *Protocol Buffers*[2], which provides a language-neutral and extensible specification of data structures and service endpoints[3]. Communication between the Competition Platform and the participant-provided tool is realized via *gRPC*[4], allowing participants to implement their solutions in a wide range of programming languages and technology stacks.

This design choice ensures a clear separation between the evaluation infrastructure and the submitted tools, while maintaining interoperability, performance, and reproducibility.

2.1.2 Docker. Given the heterogeneity of implementation technologies enabled by the gRPC-based interface (e.g., Python, Java, or other ecosystems), all submitted tools were required to be packaged as Docker containers. Each participant provided a Dockerfile alongside their interface implementation, which was used to build a self-contained image of the tool.

[1] https://github.com/christianbirchler-org/sdc-testing-competition
[2] https://protobuf.dev/
[3] https://github.com/christianbirchler-org/sdc-testing-competition/blob/main/competition.proto
[4] https://grpc.io/

2.1.3 Sample Tool & Data. To facilitate participation and lower the entry barrier, we provided a reference implementation of the interface together with an initial dataset. The sample tool implements a simple random test prioritization strategy and serves both as a usage example and as a baseline for comparison in the evaluation.

2.2 Competition Tools

We received one competing tool: ITEP4SDC. For each tool (including the random baseline), we built a Docker image (x86_64 architecture) and made them publicly available in the GitHub container registry[5].

2.3 Benchmark Test Suites

We use pre-executed test cases based on the SensoDat dataset [8] (developed as results of previous research [5, 4, 2, 3]). SensoDat consists of 36 collections of test cases. Within a single collection, all test cases are generated and executed under the same conditions, i.e., test generator, driving AI setting, and Oracle definition. Each collection represents a test suite; hence, we have a sample size of 36. In Table 1, the benchmarks are listed, which are generated by three different test generators.

2.3.1 Ambiegen. Humeniuk et al. [11] proposed the Ambiegen test generator tool for simulation-based testing of SDCs. The generator uses a two-objective NSGA-II algorithm to generate the simulation-based test cases.

2.3.2 Frenetic. Castellano et al. [9] developed the Frenetic test generator. Frenetic is also an approach that uses a genetic algorithm with a curvature representation of the road.

2.3.3 FreneticV. Castellano et al. [10] proposed the FreneticV test generation approach. It is an extension to the previous Frenetic tool that aims to produce less invalid, i.e., self-intersecting roads.

2.4 Evaluation Metrics

In this edition of the competition, the task has shifted from test *selection* to test *prioritization*. Accordingly, the evaluation focuses on metrics that capture how effectively and efficiently a tool orders the entire test suite with respect to fault revelation. The following metrics were used to assess each submission.

2.4.1 Initialization Time. Before the evaluation begins, each tool receives auxiliary training data. Tools may use this phase to construct internal models or preprocess information. The initialization time captures the duration required for this phase.

2.4.2 Prioritization Time. After initialization, each tool must generate a complete prioritization of the provided test suite. The prioritization time measures only the computation time needed to produce the final ordering, excluding initialization.

2.4.3 Average Percentage of Faults Detected (APFD). The APFD metric is a standard measure to assess the effectiveness of a test prioritization technique [1]. It quantifies how quickly faults are revealed when following the prioritized order of test cases. More

[5] https://github.com/orgs/christianbirchler-org/packages?repo_name=sdc-testing-competition

Table 1: Benchmark Overview of SensoDat [8]

Collection (Test Suite)	# Test Cases
campaign_2_ambiegen	973
campaign_2_frenetic	928
campaign_2_frenetic_v	944
campaign_3_ambiegen	964
campaign_3_frenetic	954
campaign_4_ambiegen	965
campaign_4_frenetic	964
campaign_4_frenetic_v	525
campaign_5_ambiegen	958
campaign_5_frenetic	945
campaign_5_frenetic_v	940
campaign_6_ambiegen	959
campaign_6_frenetic	944
campaign_6_frenetic_v	764
campaign_7_ambiegen	963
campaign_7_frenetic	967
campaign_7_frenetic_v	47
campaign_8_ambiegen	952
campaign_8_frenetic	952
campaign_9_ambiegen	953
campaign_9_frenetic	964
campaign_10_ambiegen	971
campaign_11_ambiegen	973
campaign_11_frenetic	866
campaign_11_frenetic_v	953
campaign_12_frenetic	956
campaign_12_freneticV	942
campaign_13_ambiegen	954
campaign_13_frenetic	959
campaign_13_frenetic_v	951
campaign_14_ambiegen	959
campaign_14_frenetic	866
campaign_14_frenetic_v	934
campaign_15_ambiegen	952
campaign_15_frenetic	870
campaign_15_freneticV	949
Total	**32,580**

formally, the APFD of a test suite is defined as follows:

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \cdot m} + \frac{1}{2n},$$

where n is the number of tests in the test suite, m is the number of failing tests, and TF_i the position (rank) of the i-th failing test in the test suite (with ranks starting at 1). Higher APFD values indicate that faults are detected earlier in the sequence, thus reflecting a more effective prioritization.

*2.4.4 Cost-Aware APFD (*APFD_*C).* In self-driving car testing, individual test scenarios may differ substantially in simulation time due to differences in route length or vehicle dynamics. As a result, two prioritizations that achieve identical APFD scores may differ substantially in how cost-effective they are: one may detect failures early using short tests, while another may only detect failures after longer, more expensive simulations. APFD_C extends APFD to account for this variability by incorporating the execution cost of each test, represented in the context of our challenge by its simulation time. This cost-aware variant evaluates how quickly faults are revealed relative to the cumulative simulation time spent. Formally, let C_j be the execution cost of the j-th test in the prioritized order, $CFi = \sum_{j=1}^{i} C_j$ be the cumulative cost up to (and including) the i-th failing test, and $C_{\text{total}} = \sum_{j=1}^{n} C_j$ be the total cost of executing the full test suite. Then:

$$\text{APFD}_C = 1 - \frac{\sum_{i=1}^{m} CF_i}{C_{\text{total}} \cdot m} + \frac{1}{2m}.$$

2.4.5 Time to First Fault. The time to first fault measures the cumulative simulation time needed to encounter the first failing test case in the prioritized list. Lower values indicate that the tool ranks fault-revealing tests early.

2.4.6 Time to Last Fault. Complementing the previous metric, the time to last fault measures the cumulative simulation time required to reach the final failing test case in the prioritized sequence. This metric indicates how the tool distributes fault-revealing tests across the entire ordering.

2.5 Experimental Setup

The experiments are conducted on a virtual machine with an NVIDIA RTX A6000 GPU. Furthermore, we also used the Nvidia Runtime extension for the Docker engine to enable the use of GPU for the tools.

2.6 Procedure

Contrary to previous years, we did not use each testing campaign from the SensoDat dataset as a dedicated sample, as this could make this year's competition evaluation predictable. Instead, we randomly sampled among all available test cases within SensoDat, thus constructing test suites from a wider range of available test cases. Specifically, a sample consists of multiple subjects, i.e., test cases, constructing a test suite. The construction of the subjects happened randomly among all SensoDat test cases. Hence, for the construction of the samples, two parameters were required: (i) the size of the samples, i.e., how many subjects (test suites) should the sample have, and (ii) the size of the subjects, i.e., how many test cases a subject should consist of.

Each experiment involves a sample with multiple subjects. Each subject represents a test suite that the tools have to "treat", aka prioritize. For each treatment, we assess the prioritization performance. Overall, we experimented with 20 different sample configurations as well with 20 different subject sizes. Concretely, we use sample sizes N_{sample} and subject sizes $N_{subject}$ with $N_{subject}, N_{sample} \in \{i * 10, \forall i \in \{1, 2, 3, ..., 20\}\}$.

Before a tool treats the sample of subjects (test suites), the tools get a single subject of the sample for initialization purposes. The remaining subjects (test suites) will be prioritized and assessed.

2.7 Data Collection & Analysis

The tool evaluator provides the tools unsorted test suites and retrieves from the tools ordered, i.e., prioritized test suites. Based on the prioritized test suites, the evaluator computes the metrics as described in Section 2.4 for each test suite. For the final evaluation, we consider only the basic statistics of the sample, i.e., the maximum, minimum, mean, and standard deviation, and persist them in a dedicated database for analysis. The database will be made available for further analysis that can be conducted.

3 Experiments and Results

We ran each tool on multiple samples with different configurations, i.e., different sample and subject sizes. In total, each tool was evaluated on 400 samples. In Table 2, we computed the statistics of the sample after we "treated" them with the participants' tools. For the

Table 2: Average Performance Metrics Statistics

Tool	Statistic	Metrics				
		apfd	apfdc	t_prioritize_tests	t_first_fault	t_last_fault
ITEP4SDC	max	0.84	0.92	0.80	87.90	3,706.99
	mean	**0.79**	**0.83**	0.78	**54.77**	**3,016.25**
	std	0.03	0.05	0.02	28.31	553.80
	min	0.74	0.76	0.76	22.69	2,451.42
Random Baseline	max	0.62	0.67	0.52	668.30	6,613.83
	mean	0.50	0.52	**0.46**	145.75	5,781.33
	std	0.05	0.06	0.03	132.02	349.89
	min	0.38	0.39	0.37	6.65	4,898.84

sake of comparison, we also provide the statistics of the random baseline tool (provided in the competition platform implemented with Python). From the results, we observe that the ITEP4SDC outperforms, on average, the random baseline. The only exception is the marginal higher runtime, which seems to be negligible from a practical point of view.

4 Discussion & Conclusion

The ITEP4SDC tool generally outperforms the random baseline prioritizer. However, it is noteworthy that during the experiments, the ITEP4SDC tool was not able to prioritize a wide range of different subjects, whereas the baseline tool did not have any issues in prioritizing any subject. Concretely, overall treatments, ITEP4SDC failed 35,659 times to prioritize but succeeded only on 5,941 test suites. The evaluation results in Table 2 consider only those treatments that actually succeeded; further investigation of that tool is required to identify the root cause of its inability to prioritize a large set of test suites.

We evaluated and compared the ITEP4SDC tool with a random approach as a baseline. According to our results, the ITEP4SDC tool outperforms the baseline when the tool is able to prioritize a given test suite.

This year marked the second edition of the SDC Testing Competition, jointly organized at ICST and SBFT 2026. This edition introduced (regression) metrics for *test prioritization*, as well as compared with a baseline random approach. Future editions will introduce new metrics [7] and expand to diverse obstacle types, such as trees and buildings, as well as environmental factors.

5 Data Availability

The tools, evaluation, and data are publicly available on GitHub: https://github.com/christianbirchler-org/sdc-testing-competition.

Acknowledgments

We thank the participants of the competition for their invaluable contribution. We thank the Horizon 2020 (EU Commission) support for the project InnoGuard, Marie Sklodowska-Curie Actions Doctoral Networks (HORIZON-MSCA-2023-DN), and the Hasler Foundation for the project "Safe2Fly" (No. 2025-02-27-311), and the SNSF for the project entitled "SwarmOps: Human-sensing based MLOps for Collaborative Cyber-physical systems" (Project No. 200021_219732). Furthermore, we also thank CHOOSE, the Swiss Group for Original and Outside-the-box Software Engineering, for their financial support.

References

[1] Francesco Altiero, Anna Corazza, Sergio Di Martino, Adriano Peron, and Luigi Libero Lucio Starace. 2024. Regression test prioritization leveraging source code similarity with tree kernels. *Journal of Software: Evolution and Process*, 36, 8, e2653.

[2] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. 2022. Cost-effective simulation-based test selection in self-driving cars software. *Science of Computer Programming (SCP)*.

[3] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. 2022. Cost-effective simulation-based test selection in self-driving cars software with sdc-scissor. In *the 29th IEEE International Conference on Software Analysis, Evolution, and Reengineering*.

[4] Christian Birchler, Sajad Khatiri, Bill Bosshard, Alessio Gambi, and Sebastiano Panichella. 2022. Machine learning-based test selection for simulation-based testing of self-driving cars software. *Empirical Software Engineering*.

[5] Christian Birchler, Sajad Khatiri, Pouria Derakhshanfar, Sebastiano Panichella, and Annibale Panichella. 2022. Single and multi-objective test cases prioritization for self-driving cars in virtual environments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*.

[6] Christian Birchler, Sajad Khatiri, Pooja Rani, Timo Kehrer, and Sebastiano Panichella. 2025. A roadmap for simulation-based testing of autonomous cyber-physical systems: challenges and future direction. *ACM Trans. Softw. Eng. Methodol.*, 34, 5, 152:1–152:9. doi:10.1145/3711906.

[7] Christian Birchler, Tanzil Kombarabettu Mohammed, Pooja Rani, Teodora Nechita, Timo Kehrer, and Sebastiano Panichella. 2024. How does simulation-based testing for self-driving cars match human perception? *Proc. ACM Softw. Eng.*, 1, FSE, 929–950. doi:10.1145/3643768.

[8] Christian Birchler, Cyrill Rohrbach, Timo Kehrer, and Sebastiano Panichella. 2024. Sensodat: simulation-based sensor dataset of self-driving cars. In *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*. Diomidis Spinellis, Alberto Bacchelli, and Eleni Constantinou, (Eds.) ACM, 510–514. doi:10.1145/3643991.3644891.

[9] Ezequiel Castellano, Ahmet Cetinkaya, Cédric Ho Thanh, Stefan Klikovits, Xiaoyi Zhang, and Paolo Arcaini. 2021. Frenetic at the SBST 2021 tool competition. In *14th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2021, Madrid, Spain, May 31, 2021*. IEEE, 36–37. doi:10.1109/SBST52555.2021.00016.

[10] Ezequiel Castellano, Stefan Klikovits, Ahmet Cetinkaya, and Paolo Arcaini. 2022. Freneticv at the SBST 2022 tool competition. In *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022*. IEEE, 47–48. doi:10.1145/3526072.3527532.

[11] Dmytro Humeniuk, Giuliano Antoniol, and Foutse Khomh. 2022. Ambiegen tool at the SBST 2022 tool competition. In *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022*. IEEE, 43–46. doi:10.1145/3526072.3527531.

[12] Sajad Khatiri, Francisco Eli Vina Barrientos, Maximilian Wulf, Paolo Tonella, and Sebastiano Panichella. 2025. Bridging research and practice in simulation-based testing of industrial robot navigation systems.

[13] Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. 2023. Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights. In *IEEE Conference on Software Testing, Verification and Validation, ICST 2023, Dublin, Ireland, April 16-20, 2023*. IEEE, 281–292. doi:10.1109/ICST57152.2023.00034.

[14] Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. 2024. Simulation-based testing of unmanned aerial vehicles with aerialist. In *International Conference on Software Engineering (ICSE)*.

[15] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Softw. Test. Verification Reliab.*, 22, 2, 67–120. doi:10.1002/STV.430.

[16] Fiorella Zampetti, Ritu Kapur, Massimiliano Di Penta, and Sebastiano Panichella. 2022. An empirical characterization of software bugs in open-source cyber-physical systems. *J. Syst. Softw.*, 192, 111425. doi:10.1016/J.JSS.2022.111425.