SBFT Tool Competition 2024 - Python Test Case Generation Track

Nicolas Erni
Zurich University of Applied Sciences
Switzerland

Al-Ameen Mohammed Ali Mohammed
Zurich University of Applied Sciences
Switzerland

Christian Birchler
Zurich University of Applied Sciences
University of Bern
Switzerland

Pouria Derakhshanfar
JetBrains
The Netherlands

Stephan Lukasczyk
University of Passau
Germany

Sebastiano Panichella
Zurich University of Applied Sciences
Switzerland

ABSTRACT

Test case generation (TCG) for Python poses distinctive challenges due to the language's dynamic nature and the absence of strict type information. Previous research has successfully explored automated unit TCG for Python, with solutions outperforming random test generation methods. Nevertheless, fundamental issues persist, hindering the practical adoption of existing test case generators. To address these challenges, we report on the organization, challenges, and results of the first edition of the Python Testing Competition. Four tools, namely UTBotPython, Klara, Hypothesis Ghostwriter, and Pynguin were executed on a benchmark set consisting of 35 Python source files sampled from 7 open-source Python projects for a time budget of 400 seconds. We considered one configuration of each tool for each test subject and evaluated the tools' effectiveness in terms of code and mutation coverage. This paper describes our methodology, the analysis of the results together with the competing tools, and the challenges faced while running the competition experiments.

KEYWORDS

Tool Competition, Software Testing, Test Case Generation, Python, Search Based Software Engineering

ACM Reference Format:
Nicolas Erni, Al-Ameen Mohammed Ali Mohammed, Christian Birchler, Pouria Derakhshanfar, Stephan Lukasczyk, and Sebastiano Panichella . 2024. SBFT Tool Competition 2024 - Python Test Case Generation Track. In *Proceedings of 2024 ACM/IEEE International Workshop on Search-Based and Fuzz Testing (SBFT '24).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This year, we organized the first edition of the Python SBFT Tool Competition. The competition has the goal to experiment with testing tools for a diversified set of systems and domains. We invited researchers to participate in the competition with their unit test generation tools for Python [12]. The tools are evaluated against benchmarks concerning code and mutation coverage as similarly done in previous SBFT tool competitions for Java [15].

SBFT '24, April 14, 2024, Lisbon, Portugal

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 THE PYTHON TESTING COMPETITION

The first edition of the Python Testing Tool Competition received one submitted tool: UTBotPython [11]. Furthermore, similarly to previous/contemporary testing competition editions of SBFT [19, 20, 25], we used various different baseline tools, namely Pynguin [21], Hypothesis Ghostwriter [23], and Klara [5] for comparison. Each test generation tool has been executed on (i.e., generated test cases for) 35 Python source code files sampled from 7 open-source projects on GitHub, which are TensorFlow [10], Django [3], Flask [4], Numpy [6], scikit-learn [8], Ansible [1], and Spark [9]. Eventually, with the submitted tool and the three baseline tools evaluated on seven benchmark projects, we get a broad performance overview of the state-of-the-art test generation tools for Python.

To guarantee a fair comparison among the competing tools, the execution of the tools for generating test suites and their evaluation has been carried out by using an infrastructure hosted on GitHub[1] and Zenodo [17]. Each tool implements the same interface provided by the aforementioned infrastructure code. The competing tools have been compared by using a set of various coverage metrics, such as line, branch, and mutant coverage. For the comparison, all tools got a one-time budget of 400 seconds to generate test cases for the aforementioned benchmark source files.

2.1 Benchmark subjects

Similarly to previous editions of SBFT tool competitions [19, 25], the selection of the projects and Python files under test to use as benchmark for test case generation has been done by considering three criteria: (i) projects must belong to different application domains; (ii) projects must be open-source for replicability purposes; and (iii) files must not have increasing complexity.

We focused on popular open-source projects on GitHub written in Python. Specifically, we picked:

- TensorFlow[2]: an end-to-end open source platform for machine learning;

[1] https://github.com/ThunderKey/python-tool-competition-2024
[2] https://github.com/tensorflow/tensorflow

Table 1: Description of the benchmarks.

Project	# Python Files	# Sampled Python Files
TensorFlow	3135	5
Django	2771	5
Flask	82	5
Numpy	581	5
scikit-learn	926	5
Ansible	1559	5
Spark	1134	5
Total	10 188	35

- Django[3]: a high-level Python web framework to build complex data-driven websites;
- Flask[4]: a lightweight WSGI web application framework;
- Numpy[5]: a package for scientific computing with Python;
- scikit-learn[6]: is a Python module for machine learning;
- Ansible[7]: an IT automation system for configuration management, application development, and cloud provisioning;
- Spark[8]: an analytics engine for large-scale data processing.

Based on the time and resources available for running the competition, we have only sampled a limited number of files. Specifically for the third selection criteria, we gathered a dataset of Python files, ensuring each file contained at least one function and maintained an average code length of 20 lines. Furthermore, our selection criteria explicitly excluded files that possess the potential to terminate processes, notably, those leveraging the `os`[9] module, as well as files linked to libraries with significant security implications, particularly those with capabilities to alter the file system through write or delete operations.

2.2 Competing tools

Four tools are competing in the first edition: UTBotPython [11], Pynguin [21], Hypothesis Ghostwriter [23], and Klara [5]. We received one submitted tool, UTBotPython, and used Hypothesis Ghostwriter, Klara, and Pynguin as baseline tools. All four tools implement different approaches to generate test cases:

- UTBotPython [11] generates test cases based on precise code analysis using a symbolic execution engine paired with a smart fuzzing technique.
- Pynguin [21] is a framework that implements several search-based unit test generation algorithms. For the competition, it uses its implementation of the DynaMOSA [24] algorithm, which is used to emit test suites with the highest coverage values compared to the other algorithms Pynguin provides [22].
- Hypothesis [23] is a library for creating property-based tests. Its Ghostwriter module provides a way to automatically generate these property-based testing features.

[3]https://github.com/django/django
[4]https://github.com/pallets/flask
[5]https://github.com/numpy/numpy
[6]https://github.com/scikit-learn/scikit-learn
[7]https://github.com/ansible/ansible
[8]https://github.com/apache/spark
[9]https://docs.python.org/3/library/os.html

Table 2: Statistics on the number of generated test cases for each tool and the time budget of 400 s after four runs.

Tool	# Test Cases			
	min	mean	median	max
UTBotPython	2	3	3	4
Pynguin	1	4.67	3	10
Hypothesis Ghostwriter	1	5.33	3	12
Klara	1	2.33	3	3

- Klara [5] is a set of static analysis tools to automatically generate test cases based on AST analysis.

2.3 Methodology

The methodology followed to run the competition is similar to the one adopted in the previous editions of the SBFT tool competitions for Java [19, 25]. It is important to remark that, due to time and resource constraints and the number of competing tools (four also considering the baseline approaches), we only considered a one-time budget of 400 seconds.

Public contest repository. The complete competition infrastructure is released under a GPL-3.0 license and is available on GitHub.[10] Specifically, the repository contains the set of Python files contributing to the first edition and the detailed summary of the results obtained by running each tool for each time budget. The competition participants are required to implement a given interface given by the infrastructure code. When implemented in the provided interface, the infrastructure code can evaluate in a straightforward manner the test generators based on a set of coverage criteria, which are described in more detail below.

Test generation and time budget. Each tool has been executed four times against each benchmark target file to account for the randomness of the test case generation process [13]. All executions got a time budget of 400 seconds to ensure a fair and feasible comparison with the available execution environment.

Execution environment. The infrastructure performed a total of 560 executions, i.e., 35 Python files × 4 tools × 1 time budget × 4 repetitions, to use for statistical analysis. To ensure a fair comparison, we ran each tool on the same dedicated machine, i.e., a virtual machine instance equipped with four vCPUs, 7.8 GB of RAM and 155 GB of memory running Ubuntu 22.04.3 LTS. For all the competing tools, we were able to run the planned number of executions.

Metrics computation. We compared the performance of the competing tools based on line, branch, and mutation coverage metrics. Specifically, to compute both line and branch coverage metrics, we relied on PyTest [7] and its pytest-cov [11] plugin, an open-source framework for writing unit tests. For mutation analysis, we relied on MutPy [14] and Cosmic Ray [2], considering five minutes as the maximum amount of time available for mutation analysis for each Python file.

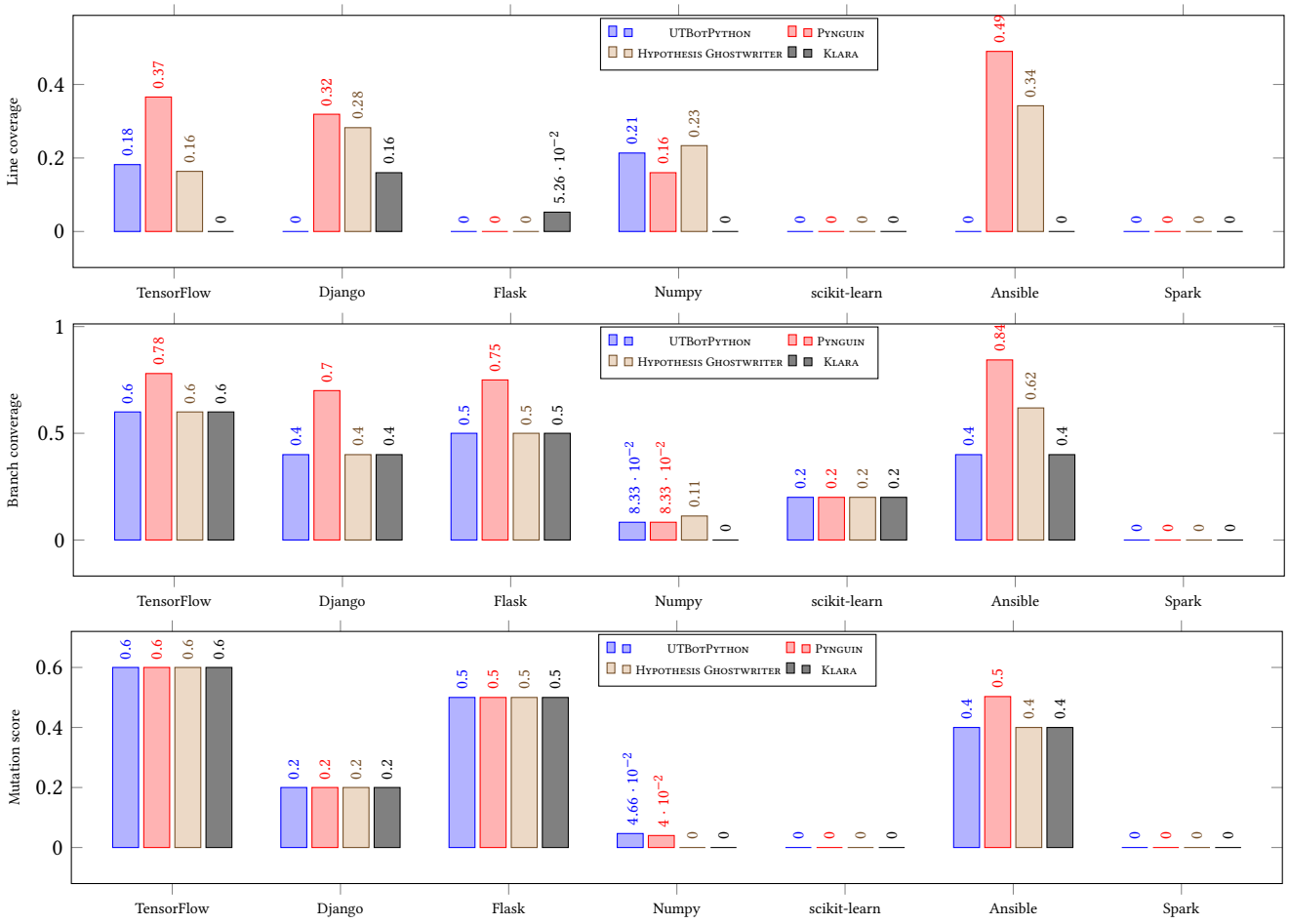

Figure 1: Average coverage and mutation score for each test generation tool after four runs.

Table 3: Average performance scores of the tools among all benchmarks.

Tool	Coverage (%)			Total Score
	Line	Branch	Mutation	
Pynguin	26.7	27.4	5.28	1.03
UTBotPython	10.1	6.63	6.60	0.498
Hypothesis Ghostwriter	18.0	9.04	2.72	0.470
Klara	3.45	2.41	2.23	0.172

2.4 Competition's results and analysis

Each tool generates a different number of test cases. We executed each tool four times with a time budget of 400 s. Numbers are rounded to three significant digits, if appropriate. Table 2 presents for each tool the minimum, mean, median, and maximum number of the generated test cases. We observed that Hypothesis Ghostwriter generates, on average, most test cases followed by Pynguin, UTBotPython, and then Klara.

[10]https://github.com/ThunderKey/python-tool-competition-2024
[11]https://pytest-cov.readthedocs.io/en/latest/

We computed different metrics, such as the line, branch coverage, and mutation score, to evaluate the quality of the generated test cases. Table 3 reports the average percentage of lines, branches, and mutants being covered by the tools after four executions. Regarding line coverage, Pynguin has the highest value with 26.7 %. Meanwhile, Klara had the lowest with 3.45 %. In terms of branch coverage, Pynguin also has the highest score with more than 27 %, while Klara has again the lowest with 2.41 %. Furthermore, Table 3 also reports the average mutation coverage, which is the ratio between the number of mutants that were killed by at least one test and the total number of mutants being generated. Eventually, we report the final scores of the tools based on a time budget of 400 s. The formula [16] for the final score has been created and improved during the previous editions of the tool competition and takes into account the line and branch coverage and the mutation score used by the generator. In terms of ranking, we have Pynguin as first, followed by UTBotPython, Hypothesis Ghostwriter, and Klara.

In addition to the general performance evaluation, we also evaluated the tools' performance for each benchmark project individually. Figure 1 depicts the various performance metrics of each tool among

all benchmarks. We observed that the tools could not create any tests for certain benchmarks. For example, no tool was able to generate tests for the Spark project. We can see this due to the fact that all metrics were zero for this specific benchmark. Interestingly, for the scikit-learn project, we still got some branch coverage but no line coverage. This observation contradicts the general perception that no branch can be covered when no line is covered. We think this branch coverage might occur due to empty `if` clauses where the condition is true but no executable statement is in the block except for the `pass` keyword. Furthermore, in the case of the Spark benchmark, only Klara could cover a small number of lines.

A potential cause for this low line coverage might still be the code complexity of those aforementioned benchmark projects although it was already a selection criteria not having an increasing complexity (Section 2.1). We argue for future editions of the competition, the tools should investigate the feasibility to handle complex file structures from real benchmark projects. Furthermore, it could be interesting to investigate to what extent certain newly introduced Python language features affect the tools in their performances.

3 CONCLUSIONS AND FINAL REMARKS OF THE PYTHON TESTING TOOL COMPETITION

This year marks the first edition of the Python Unit Testing Competition. We received one tool, namely UTBotPython, which competes against three baseline tools, namely, Pynguin, Hypothesis Ghostwriter, and Klara. As per results of this year, the best-performing tool overall is Pynguin followed by UTBotPython, and Hypothesis Ghostwriter while Klara seems to perform the worst on the selected benchmark subject files. The analysis of collected results by the organizers of the competition and by the participants revealed that for some of the generated test suites, it was not possible to generate tests or compute coverage and mutation analysis. The most likely cause of this is the files are too complex, contain many relative imports, or lack modularity. We plan to investigate this issue further to identify the definite root cause of the problem and to perform a fix for the next editions of the competition in order to provide better criteria for the right type of files. In addition, we envision several other possibilities for improvement, such as extending the list of criteria used for the evaluation, such as performance awareness [18] and readability [26, 27] scores.

4 DATA AVAILABILITY

We provide all code and detailed results on Zenodo [17].

ACKNOWLEDGMENTS

We thank the participants of the competitions for their invaluable contribution. We thank the Horizon 2020 (EU Commission) and Innosuisse support for the projects COSMOS (DevOps for Complex Cyber-physical Systems, Project No. 957254-COSMOS) and ARIES (Project 45548.1 IP-ICT).

REFERENCES

[1] 2024. Ansible. https://github.com/ansible/ansible.
[2] 2024. Cosmic Ray. https://cosmic-ray.readthedocs.io/en/latest/.
[3] 2024. django. https://github.com/django/django.
[4] 2024. Flask. https://github.com/pallets/flask.
[5] 2024. Klara. https://github.com/usagitoneko97/klara.
[6] 2024. NumPy. https://github.com/numpy/numpy.
[7] 2024. pytest. https://docs.pytest.org/en.
[8] 2024. scikit-learn. https://github.com/scikit-learn/scikit-learn.
[9] 2024. Spark. https://github.com/apache/spark.
[10] 2024. TensorFlow. https://github.com/tensorflow/tensorflow.
[11] 2024. UTBotPython. https://github.com/UnitTestBot/UTBotPythonSBFT2024.
[12] Carol V. Alexandru, José J. Merchante, Sebastiano Panichella, Sebastian Proksch, Harald C. Gall, and Gregorio Robles. 2018. On the Usage of Pythonic Idioms. In *ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2018)*. Association for Computing Machinery, 1–11. https://doi.org/10.1145/3276954.3276960
[13] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Softw. Test. Verif. Reliab.* 24, 3 (May 2014), 219–250. https://doi.org/10.1002/stvr.1486
[14] Anna Derezinska and Konrad Halas. 2014. Experimental Evaluation of Mutation Testing Approaches to Python Programs. In *Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014 Workshops Proceedings, March 31 - April 4, 2014, Cleveland, Ohio, USA*. IEEE Computer Society, 156–164. https://doi.org/10.1109/ICSTW.2014.24
[15] Xavier Devroey, Alessio Gambi, Juan Pablo Galeotti, René Just, Fitsum Kifetew, Annibale Panichella, and Sebastiano Panichella. [n.d.]. JUGE: An infrastructure for benchmarking Java unit test generators. *Software Testing, Verification and Reliability* ([n. d.]), e1838. https://doi.org/10.1002/stvr.1838
[16] Xavier Devroey, Alessio Gambi, Juan Pablo Galeotti, René Just, Fitsum Kifetew, Annibale Panichella, and Sebastiano Panichella. 2021. JUGE: An Infrastructure for Benchmarking Java Unit Test Generators. https://doi.org/10.48550/arXiv.2106.07520
[17] Nicolas Erni, Al-Ameen Mohammed Ali Mohammed, Christian Birchler, Pouria Derakhshanfar, Stephan Lukasczyk, and Sebastiano Panichella. 2024. SBFT Tool Competition 2024 - Python Test Case Generation Track. https://doi.org/10.5281/zenodo.10554259
[18] G. Grano, C. Laaber, A. Panichella, and S. Panichella. 2019. Testing with Fewer Resources: An Adaptive Approach to Performance-Aware Test Case Generation. *IEEE Transactions on Software Engineering* (2019), 1–1.
[19] Gunel Jahangirova and Valerio Terragni. 2023. SBFT Tool Competition 2023 - Java Test Case Generation Track. In *International Workshop on Search-Based and Fuzz Testing*. IEEE, 61–64. https://doi.org/10.1109/SBFT59156.2023.00025
[20] Sajad Khatiri, Prasun Saurabh, Timothy Zimmermann, Charith Munasinghe, Christian Birchler, and Sebastiano Panichella. 2024. SBFT Tool Competition 2024 - CPS-UAV Test Case Generation Track. In *IEEE/ACM International Workshop on Search-Based and Fuzz Testing, SBFT@ICSE 2024*.
[21] Stephan Lukasczyk and Gordon Fraser. 2022. Pynguin: automated unit test generation for Python. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22)*. ACM, 168–172. https://doi.org/10.1145/3510454.3516829
[22] Stephan Lukasczyk, Florian Kroiß, and Gordon Fraser. 2023. An empirical study of automated unit test generation for Python. *EMSE* 28, 2 (2023), 36. https://doi.org/10.1007/S10664-022-10248-W
[23] David Maciver and Zac Hatfield-Dodds. 2019. Hypothesis: A new approach to property-based testing. *J. Open Source Softw.* 4, 43 (2019), 1891. https://doi.org/10.21105/JOSS.01891
[24] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering* 44, 2 (2018), 122–158. https://doi.org/10.1109/TSE.2017.2663435
[25] Sebastiano Panichella, Alessio Gambi, Fiorella Zampetti, and Vincenzo Riccio. 2021. SBST Tool Competition 2021. In *International Workshop on Search-Based Software Testing*. IEEE, 20–27. https://doi.org/10.1109/SBST52555.2021.00011
[26] Sebastiano Panichella, Annibale Panichella, Moritz Beller, Andy Zaidman, and Harald C. Gall. 2016. The impact of test case summaries on bug fixing performance: an empirical investigation. In *International Conference on Software Engineering, ICSE 2016*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, 547–558. https://doi.org/10.1145/2884781.2884847
[27] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, and Oscar Nierstrasz. 2021. How to identify class comment types? A multi-language approach for class comment classification. *J. Syst. Softw.* 181 (2021), 111047. https://doi.org/10.1016/J.JSS.2021.111047