Evaluating Features for Machine Learning Detection of Order- and Non-Order-Dependent Flaky Tests

Owain Parry[1], Gregory M. Kapfhammer[2], Michael Hilton[3], Phil McMinn[1]

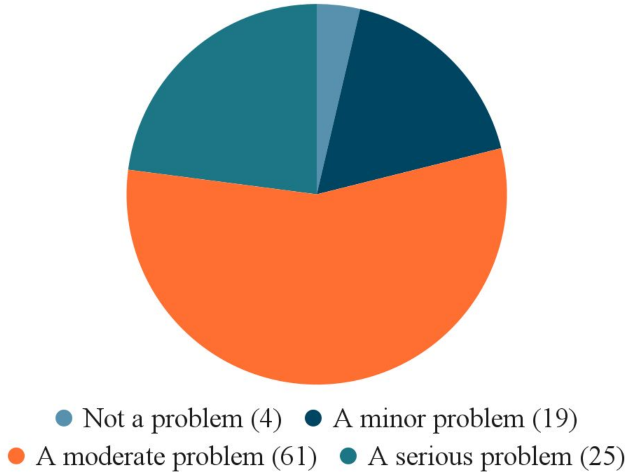
[1]University of Sheffield, UK
[2]Allegheny College, USA
[3]Carnegie Mellon University, USA

What is a *flaky test*?

- A test case that can both pass or fail without changes to the code.
- An **unreliable signal** that may waste developers' time.
- A category of flaky tests, known as *order-dependent (OD) tests*, depend on the test execution order.
- OD flaky tests can hinder the application of techniques such as test case prioritization.

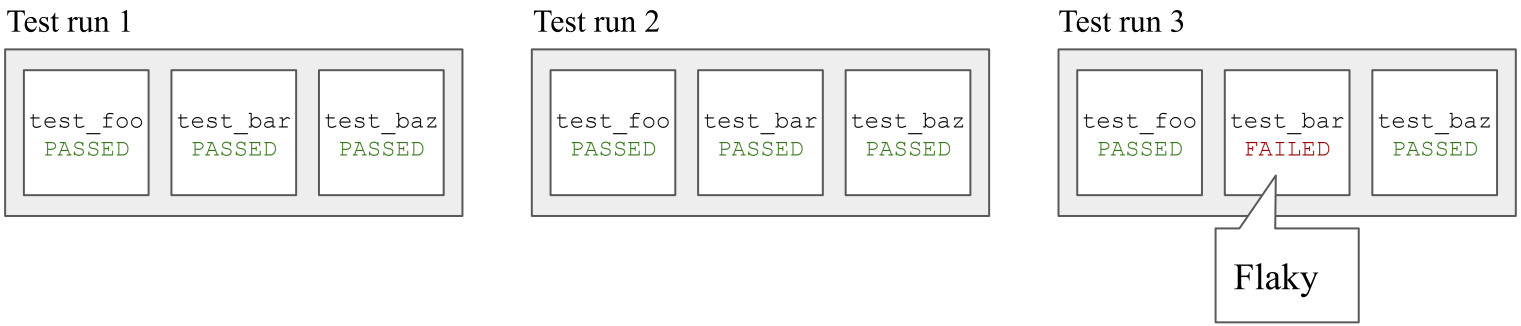
What do developers think?

A survey [Eck et. al. 2019] of 109 developers asked, "How **problematic** are flaky tests for you?".



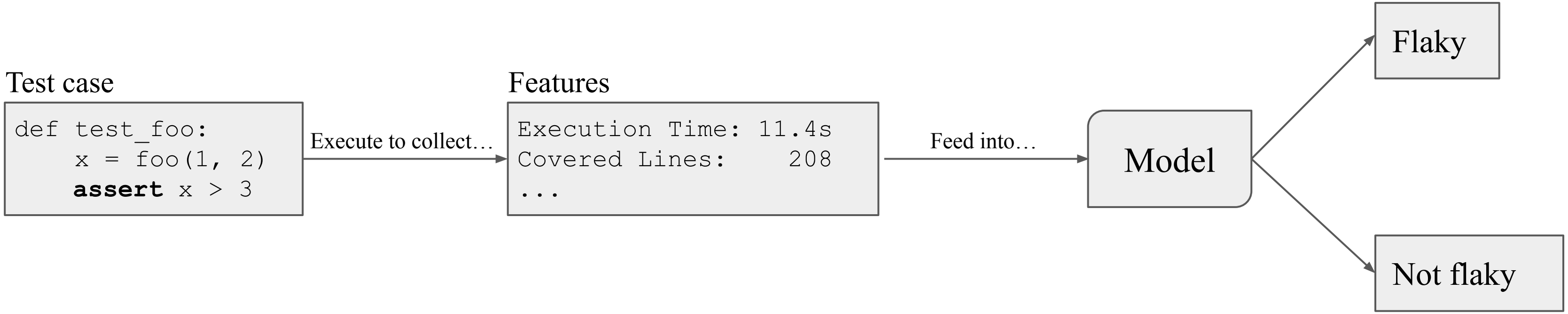
How can we detect flaky tests? Rerunning

- A simple way to detect flaky tests is to repeatedly execute test suites.
- If the outcome of a test case is inconsistent across reruns then it is flaky.
- This can be combined with adjusting the test run order to catch OD flaky tests.
- This approach can be **very slow** for projects with long-running test suites!



How can we detect flaky tests? Machine Learning

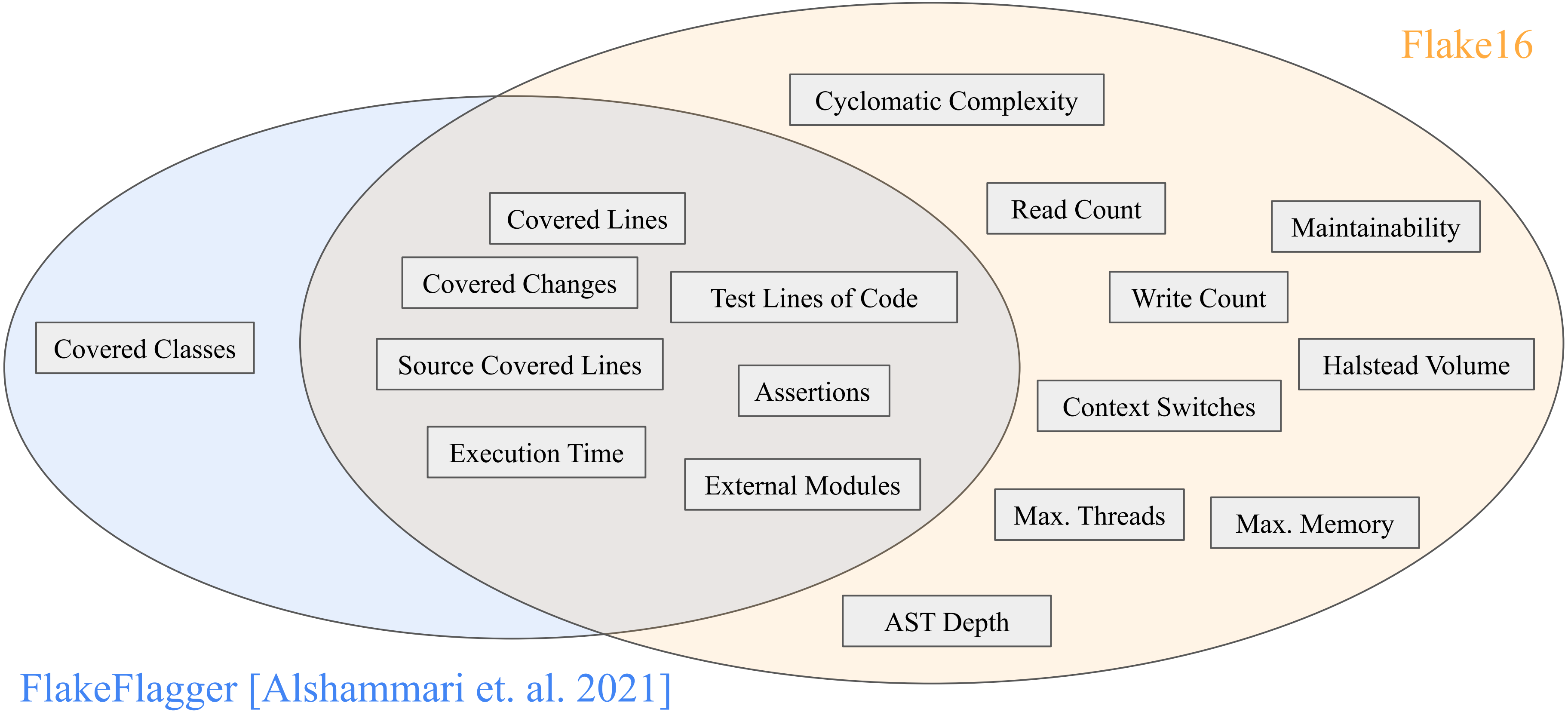
- Researchers have developed detection techniques based on machine learning models, trained using **static features** of test cases [Pinto et. al. 2020], [Bertolino et. al. 2021].
- One recent study found that combining static features with **dynamically-collected features** can result in better performance at the cost of a single test suite run [Alshammari et. al. 2021].



What did we do?

- Prior research on features to encode a test case is limited and does not consider the detection of OD flaky tests, despite being prevalent in test suites [Lam et. al. 2019].
- We introduced *Flake16*, a new feature set for encoding test cases for flaky test detection.
- It offered a **13% increase** in F1 score compared to a previous feature set when detecting *non-order-dependent (NOD)* flaky tests and a **17% increase** when detecting OD flaky tests.

The Flake16 feature set

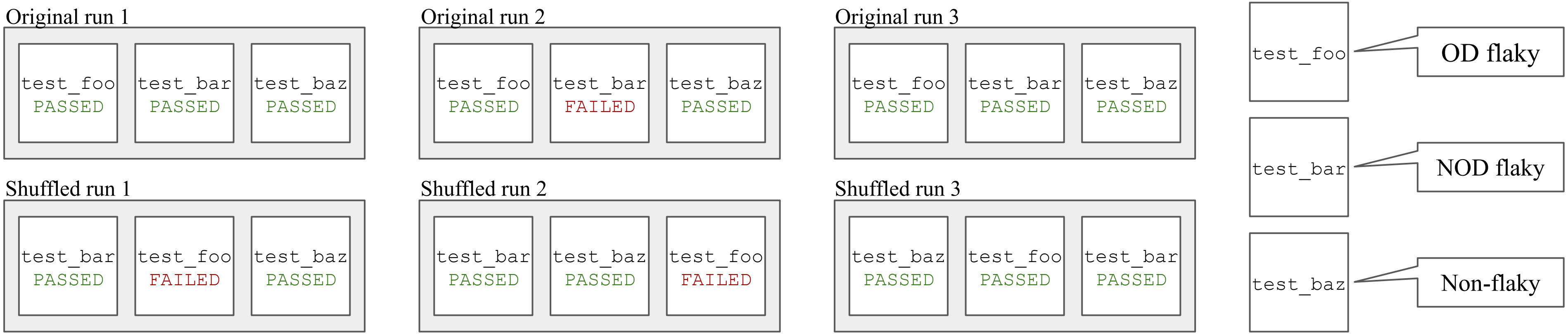


Our empirical evaluation

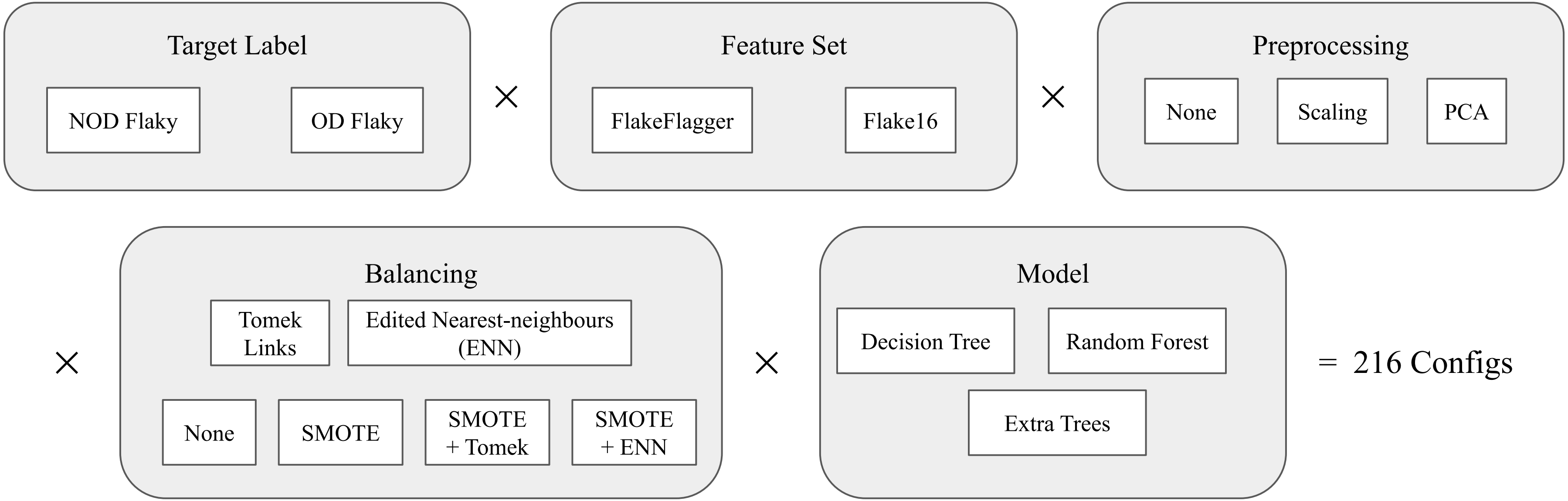
- **RQ1**. Compared to the features used by FlakeFlagger, does the Flake16 feature set improve the performance of flaky test case detection with machine learning models?
- **RQ2**. Can machine learning models be applied to effectively detect order-dependent flaky test cases?
- **RQ3**. Which features of Flake16 are the most impactful?

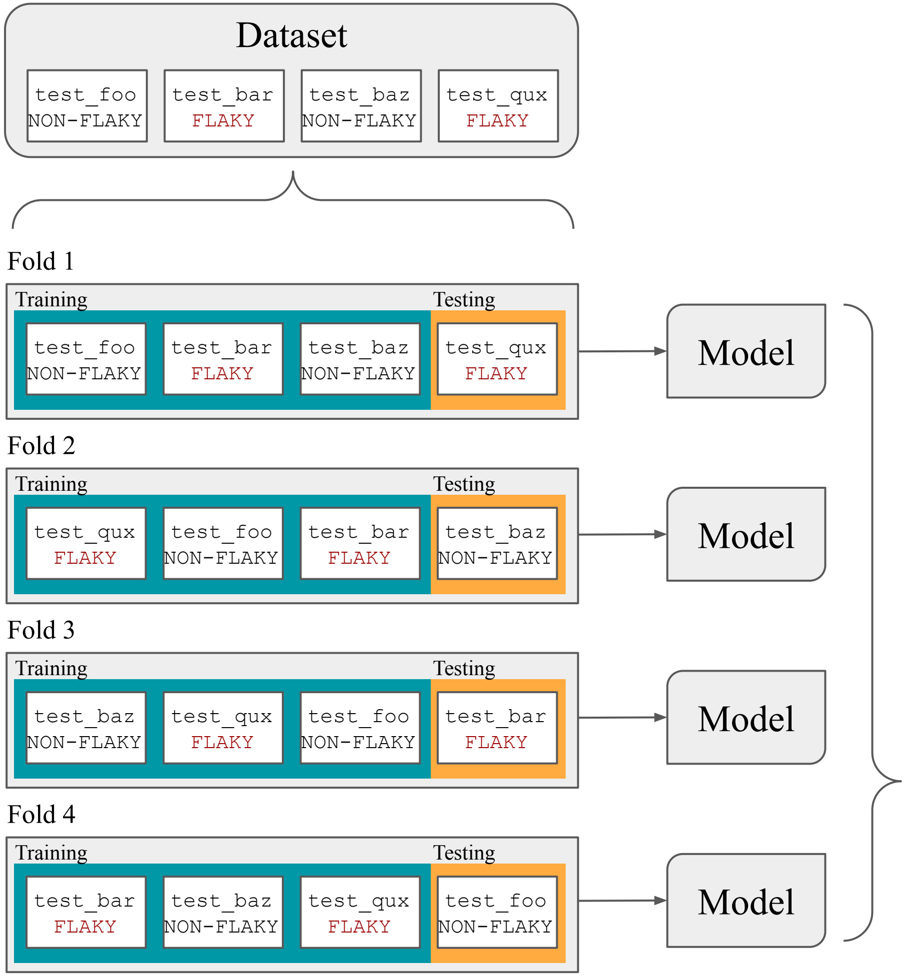
Our dataset

- A total of **67,006 test cases** from the test suites of **26 open-source Python projects** hosted on GitHub.
- Our tooling executed each project's test suite 2,500 times in its original order and 2,500 times in a shuffled order to label each test case as non-flaky, NOD flaky, or OD flaky.
- It also performed a single instrumented run of each test suite to collect feature data for each test case.
- We ended up with 145 NOD flaky tests and 1,012 OD flaky tests.



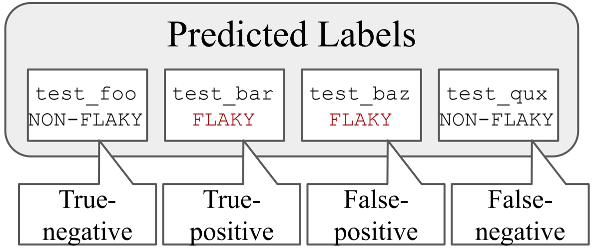
Model configurations





Model training & testing

- *Stratified 10-fold cross validation* produces 10 folds, where 90% of the dataset is for training the model and 10% for testing.
- The class balance of each fold roughly follows that of the whole dataset.
- **The testing portion of each fold is unique**, so every test case gets a predicted label.



Results: RQ1 & RQ2

	FlakeFlagger	Flake16
NOD Flaky	Preprocessing: *None* Balancing: *Tomek Links* Model: *Extra Trees* Precision: 0.75 Recall: 0.33 **F1 Score: 0.46**	Preprocessing: *PCA* Balancing: *SMOTE* Model: *Extra Trees* Precision: 0.58 Recall: 0.48 **F1 Score: 0.52**
OD Flaky	Preprocessing: *None* Balancing: *SMOTE+Tomek* Model: *Extra Trees* Precision: 0.50 Recall: 0.44 **F1 Score: 0.47**	Preprocessing: *Scaling* Balancing: *SMOTE* Model: *Random Forest* Precision: 0.50 Recall: 0.60 **F1 Score: 0.55**

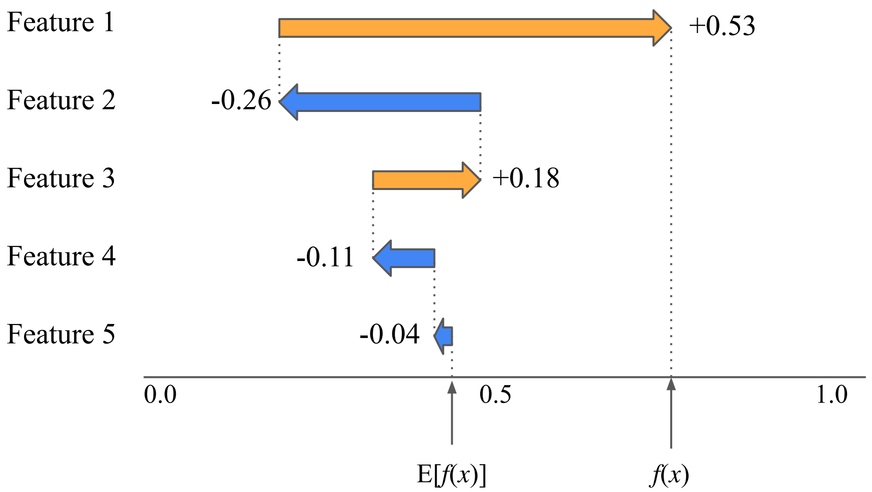
$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{F1 score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Feature impact

- To understand the impact of each feature on the model's output for a given data point, we used the *Shapely Additive Explanations (SHAP)* technique.
- In our context, a data point is a test case and the model output is the estimated probability that the test case is flaky.



Feature impact

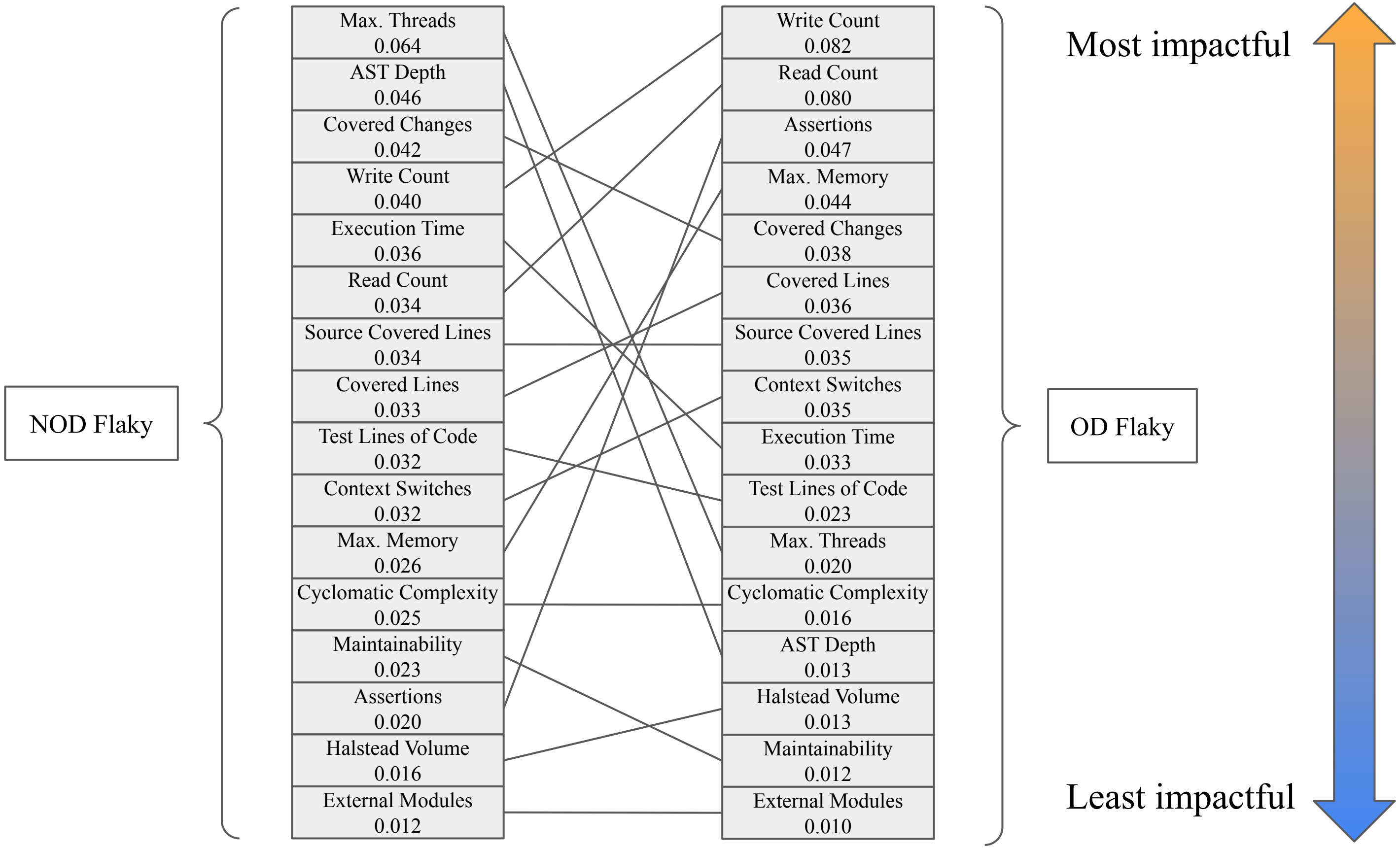
- We calculated the matrix of SHAP matrix for the best model configuration for detecting NOD flaky tests and the best configuration for OD flaky tests.
- To quantify the importance of each feature for both classification problems, we calculated the **mean absolute value** of each column in the matrix, corresponding to each feature.

Test case	Feature 1	Feature 2	Feature 3
`test_foo`	-0.030	0.089	0.061
`test_bar`	-0.036	0.031	0.094
`test_baz`	0.052	0.003	-0.033

→

Feature 1	Feature 2	Feature 3
0.039	0.041	0.063

Results: RQ3


NOD Flaky
Max. Threads 0.064
AST Depth 0.046
Covered Changes 0.042
Write Count 0.040
Execution Time 0.036
Read Count 0.034
Source Covered Lines 0.034
Covered Lines 0.033
Test Lines of Code 0.032
Context Switches 0.032
Max. Memory 0.026
Cyclomatic Complexity 0.025
Maintainability 0.023
Assertions 0.020
Halstead Volume 0.016
External Modules 0.012
Write Count 0.082
Read Count 0.080
Assertions 0.047
Max. Memory 0.044
Covered Changes 0.038
Covered Lines 0.036
Source Covered Lines 0.035
Context Switches 0.035
Execution Time 0.033
Test Lines of Code 0.023
Max. Threads 0.020
Cyclomatic Complexity 0.016
AST Depth 0.013
Halstead Volume 0.013
Maintainability 0.012
External Modules 0.010
OD Flaky
Most impactful
Least impactful

Summary

- **RQ1**: The Flake16 feature set offered a **13% increase** in overall F1 score when detecting NOD flaky tests and a **17% increase** when detecting OD flaky tests.
- **RQ2**: The performance of the best OD configuration was **broadly similar** to that of the best NOD configuration.
- **RQ3**: The most impactful feature for detecting NOD flaky tests was **Max. Threads**. For detecting OD flaky tests, **Write Count** the most impactful.