Energy Implications of Test Prioritisations in the Continuous Integration Process

Rohan Cyr Lambert[1]

Supervisor(s): Carolin Brandt[1]**, Xutong Liu**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 25, 2026

Name of the student: Rohan Cyr Lambert
Final project course: CSE3000 Research Project
Thesis committee: Carolin Brandt, Xutong Liu, Benedikt Ahrens

Abstract

Test Prioritisation (TP) is one among many different methods to optimise pipelines in Continuos Integration (CI). However very little research has been performed on the energy usage of this process. This paper documents the simulation of TP on multiple Java open source projects and presents the energy consumption of this part of the pipeline. The results of the experiment show that the costs of performing this CI technique under realistic conditions outweigh the benefits in terms of energy spending. Other Ci optimisation processes such as Test Selection may prove more valuable and combining them with TP might be redundant.

1 Introduction

In the field of Continuous Integration (CI), development teams push code changes on a frequent basis. Those changes are then checked through a pipeline that will, among other operations, run a plethora of tests to check the code for errors or bugs, making sure new additions do not break the previously built codebase.

While for a small team, running the pipeline for every code change is feasible, this does not scale well to larger projects which have much more numerous teams leading to large amounts of commits, with in some cases like with Google projects a commit every second[1]. This combined with larger and larger test suites means running the full pipeline can be costly both in terms of time that the developers have to wait, and in terms of computational resources which will also be referred as "Energy" within this paper.

To save this energy, specifically in regards to testing for errors, there are multiple techniques, this paper will focus on Test Prioritisation (TP), by ordering the tests and running the ones more likely to fail first, the energy usage can be reduced in the cases where some of the tests fail as the rest of the pipeline does not need to be processed once an error is found.

This among other techniques of optimisation of the CI pipeline has not been explored thoroughly yet, earlier works have for example demonstrated the energy usage of the whole process together[2] but individual elements of it have not been measured on their own.

These narrower parts of the process are important to check individually to help build insight into this field and justify their use as a way to save money as well as well as reduce the environmental impact and help the sustainability of future projects. As such this paper will focus on TP and try to answer the following question: "**What is the Energy Impact of Test Prioritization during CI Pipeline?**"

In order to reach a conclusion on the aforementioned question, this research will be split into 4 subquestions to lead to a thorough answer:

1. What is the Energy Impact of Training Test Prioritisation models?
2. What is the Energy Impact of Ranking Tests using Test Prioritisation models?
3. How does the Energy Impact of Running a Test Suite with Test Prioritisation compare to the Energy impact of Running a Test Suite without?
4. How does the combined Energy Impact of using Test Prioritisation compare to not performing this CI technique?

This paper is organised in the following manner: section 2 will explore related works; section 3 will describe the experiment and related information; section 4 will present the results; those results will then be discussed in section 5; section 6 will explore the ethical considerations of this paper; in the end section 7 will close this paper and look to the future of research in this domain.

2 Related Works

2.1 Test Selection

Test Selection is the concept of running only part of your tests suite by "selecting" to run only the relevant tests. It is always an option to run the entire test suite (Retest-all) and if developer wait time was the only concern, then implementing Regression Test Selection (RTS) would be unnecessary in a majority (62%) of average projects[3]. Still, it is the case that in a majority of projects (56%), static class level test selection based on code changes will perform significantly better in terms of time.

There will also be no significant losses in in failure detection at class level, but there are still improvements to be made as tests can slip through the cracks if external libraries are excluded or due to reflection (code manipulating is own structure at runtime). Research into Dynamic RTS is also starting to be developed as an alternative.[4]

Another option for Test Selection is to also predict failure detection in tests from past data. This method can be useful in for example system testing as "Traceability links between code and test cases are not always available or easily accessible when test cases correspond to system tests."[5]; adaptive methods such as reinforcement learning are currently being studied in these cases, though they need special considerations for when new tests are introduced.

2.2 Test Prioritisation

Apart from a few papers [6], Test Prioritisation is almost always performed alongside Test Selection.The data required for the latter can also be used for the former and various test prioritisation methods such as dissimilarity as a criterion (where tests similar to those with high priority will lower in the ranking to increase diversity as early as possible) do not scale as well as test selection techniques, though there is promising research on the topic [7].

Another method of prioritisation is machine learning, there are multiple avenues of improvement to the Learning To Rank (Train a model to then use on further data) strategies being explored by for example adding quality, similarity [8], or fault-proneness based metrics[9] to the standard model.

Another avenue is Ranking To Learn (RTL), where the model is trained progressively by ranking which is explored in a 2020 study by Bertolino et al[10]. While this current

research does not implement the RTL strategies present in Bertolino's paper its framework and data are heavily reused..

2.3 Energy Measurements in Continuous Integration

There are currently very few studies focused on the topic of energy consumption in CI, the work by Zaidman et al[2] presents worrying data such as Elastic Search consuming 161 kWh in 2022, roughly equivalent to 9.7% of the average household yearly energy consumption. And with the technology sector between 15% to 20% of worldwide energy consumption, and still growing, this is not negligible data.

Another study looks at the energy implications of generating tests, also investigating the energy costs of running automatically generated tests and showing they are equivalent if not better than manually created tests at runtime.[11]

2.4 Research Gap

While the topic of Energy usage in CI is starting to be investigated, there are many further steps to be taken. This research aims to advance on this path by looking at one of the subsection of the pipeline that happens on build, which is only ever evaluated as a whole in Zaidman's paper. By looking at individual sections, they can be refined to optimise the whole.

3 Experimental Setup

3.1 Tools and basic setup

EnergiBridge

In order to inspect the "Energy Impact" of our prioritisation methods, this research will make use of a tool called EnergiBridge [12]. Using this tool allows to measure the energy consumption and power usage of software processes during execution. However this tools measures this consumption for the whole machine, hence the following precautions will be taken to obtain consistent data:

- Every operation is run on the same machine in the same environment (Intel(R) Core(TM) i7-8750H CPU, Kubuntu) as it is first and foremost comparative.
- The device will have unneeded processes reduced to a minimum, and will stay plugged throughout the operations to reduce noise, one of the processes will require Wi-Fi so this is on of the features that will stay active during the whole experiment.
- Each measurement will be run 30 times to obtain a good average and get rid of outliers that could be caused by external factors.
- To avoid the previous measurements from affecting the future ones, the runs will be measured with a wait between each to stop the computer from heating up.
- To avoid the external environment from affecting the measurement, the machine will be kept in a closed room with constant temperature and the different configurations will be randomly alternated so that if the environment varies it doesn't affect only one of the configurations as a block.

Understand

Understand by Scitools [13] is a multi-tool with features to assist the production of code such as static analysis and an integrated IDE. Mirroring the study by Bertolino et al [10], it is used in this study to perform dependency and static code analysis. The dependency search will be performed when looking at commit to perform Test Selection by looking up which tests will be affected by the file changes. The static code analysis will then used to extract data on the tests that have been selected such as cyclomatic complexity, line count, or code to comment ratio. These features will then be used to train the models and to rank the tests.

3.2 Data

This research will be performed on 4 JAVA open source projects:

- Commons-Codec
- Commons-Imaging
- Commons-Io
- Commons-Lang

Some of the data from Bertolino et al's [10] paper will be reused, notably the datasets post Test Selections for between 300 and 650 commits ranging from 2011 to 2019 for each of the projects. This data will be reused to save on processing power and time usage as well as allowing for better comparison and referencing between the two studies. These datasets will be used for the training and so none of those commit should be used to actually rank and run their tests, therefore the test selection will be performed forward from the latest commit of this data to obtain both the next commit with at least one test selected and the next commit where the selected tests fail, these two will be refereed to as next commit and first failure respectively.

3.3 Experiment

Test Prioritisation in the CI process is usually done secondary to Test Selection which tends to be more evident in its benefits, this experiment follows such pipelines and so Test Selection and the data obtained and processed for it will not be measured this paper will only look at the implications of adding Test Prioritisation after that first operation. This means that some steps that would need to be measured would Test Prioritisation be used in a vacuum will be skipped.

All scripts, datasets, and results used in this paper can be found at ... (will be added once on the server)

Training Test Prioritisation models

The two models to be used as rankers for the tests will be created with Multiple Additive Regression Trees (MART) and Random Forest (RF), these Learning To Rank (LTR) algorithms having been identified as among the highest performing in their Prioritization effectiveness (RPA) within Bertolino et al's [10] paper. The training of these two models will be seeded both for the data split (done by commit) and the actual training in order to minimise variance.

The models will be trained to rank test according to chance failure, data which is obtained from previous failure rate, it

is assumed that a test that has failed more in the past is more likely to fail again as it might be either fragile or critical. A secondary feature for the ranking is speed, the idea of ranking faster tests first is that more tests performed in the same time is more likely to lead to a failure. If any given test at a commit will fail or run in a timely manner is of course unknowable without running said test so the model is trained to instead predict those characteristics from the metadata of the tests coming from the static code analysis.

The energy consumption of the code analysis will not be necessary to be obtained in the context of making the dataset for training, this is because this study follows a situation of continuous integration and while starting the process of test prioritisation deep in a project's lifecycle would indeed have be costly as it would require obtaining this metadata as well as running the tests in the past commits to build the training data, this is a one time cost and said data would be generated progressively at each iteration of the process when ranking and running the tests. Hence the only measurement in this step is the cost of training the model with the chosen algorithm on a given project.

Ranking with Test Prioritisation models

Should Test Selection not be performed then this step is where the metadata of the tests would be calculated, however when performing the dependency check required to find which tests are affected by the changes in the code, Understand performs the static code analysis that generates for each tests the features upon which they will be ranked.

The Energy measurements that will be performed are ranking the tests for both the next commit and the first failure with each model on a given project.

Running Test with and without Prioritisation

The final measurements in this process is where the failure and execution time of the data would be recorded for future training once the the model becomes obsolete in a real life scenario.

Finally will be measured the energy usage of simply running the tests (in alphabetical order for consistency of the multiple measurement) as well as with the prioritisation order for each combination of commits (next commit or first failure), model and project.

Aggregation of the Data

Once the Energy usage of the individual steps of prioritisation will have been collected, these measurements will be summed to look at the combined costs of performing Test Prioritisation in contrast to skipping it altogether. There are two cases, that of next commit representing when there are no tests failing leading to all tests running anyway and first failure representing when failures are actually found.

From the past data, an estimate of frequency of commits where the tests fail will be established and the actual Energy Impact of Test Prioritisation over time.

The hypothesis before looking at the data, is that on these small core projects with rare failures, Test Selection is sufficient and the Prioritisation process may cost more in terms of Energy that what it saves.

4 Results

Once obtained, each of the 48 configurations results went through an outlier removal script. Said script removed values over 3 standard deviations of the mean, and a single outlier point of data was deleted. The Shapiro P value was then calculated with a threshold of 0.05 for each of group of measurements. As such, it can be assumed that each configuration follows a normal distribution.

4.1 What is the Energy Impact of Training Test Prioritisation models?

In all instances of training (for all combinations of dataset and algorithm), the RPA obtained with the split test set was 1. This means that the models had, at least within that limited dataset, perfectly ranked the tests they were given.

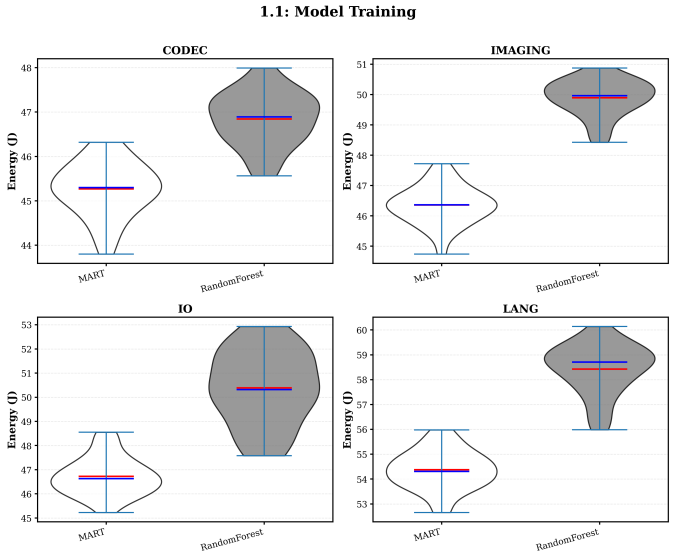

Figure 1: Energy consumption of training the models

Regarding the energy usage of training the models with the two algorithms, Mart outperforms Random Forest in all datasets with an effect size (Cohen's d) ranging from 2.6 to 5.75. This represents a considerable influence, with a welch p value for statistical significance much smaller than the required 0.05 for each dataset. The energy consumption of training can be seen in Figure 1.

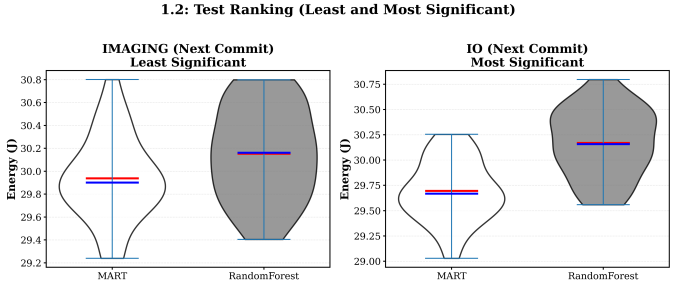

Figure 2: Energy consumption of ranking with the models

4.2 What is the Energy Impact of Ranking Tests using Test Prioritisation models?

As similarly seen with training, the energy usage of ranking the tests is much higher for Random Forest than for Mart, with effect size ranging from 0.56 to 1.48, and with a statistical significance still below 0.05. While these values are indeed still in Mart's favor as a model, they are already much closer. Imaging nc and Io nc can be found in Figure 2 as the least and most statistically significant both under 0.05.

4.3 How does the Energy Impact of Running a Test Suite with Test Prioritisation compare to the Energy impact of Running a Test Suite without?

At first when running Next Commit (nc) there is in most cases no statistical significant in consumption between the three scenarios (No prioritisation, prioritisation with Mart, and Prioritisation with Random Forest). There are exceptions, notably between the Control group and Mart, with the IO dataset where Mart had an effect size of -0.55. This is again seen between the Control and Mart with the Lang dataset, however in this instance Mart's effect size is positive, by 0.58.

When running First Failure (ff) the significance depends on the dataset. There were no significant statistical differences between the scenarios with the IO and Lang dataset. Nonetheless, there were some significant results with the other two.

With the Codec dataset, Random Forest performed significantly better with an effect size of -0.92 and -0.93 towards the control and Mart respectively. Both are statistically significant.

For the Imaging dataset, Mart outperformed the control with a statistically significant effect size of -1.15 but Random Forest outperformed both it and the control with an effect size of -0.49 and -1.80 respectively.

The specific values are present in Table 1

Table 1: Pairwise Energy Usage Comparisons Between Control, Mart, and RF

Combination	Comparison	Welch p	Cohen's d
io_ff	control vs mart	0.31	0.26
io_ff	control vs rf	0.75	0.08
io_ff	mart vs rf	0.51	-0.17
io_nc	control vs mart	0.036	-0.55
io_nc	control vs rf	0.24	-0.30
io_nc	mart vs rf	0.29	0.27
codec_ff	control vs mart	0.36	0.24
codec_ff	control vs rf	0.0008	-0.92
codec_ff	mart vs rf	0.0007	-0.93
codec_nc	control vs mart	0.17	-0.36
codec_nc	control vs rf	0.65	-0.12
codec_nc	mart vs rf	0.41	0.22
imaging_ff	control vs mart	0.00004	-1.15
imaging_ff	control vs rf	4.07e-9	-1.80
imaging_ff	mart vs rf	0.061	-0.49
imaging_nc	control vs mart	0.93	-0.02
imaging_nc	control vs rf	0.47	0.19
imaging_nc	mart vs rf	0.36	0.24
lang_ff	control vs mart	0.44	-0.20
lang_ff	control vs rf	0.98	0.01
lang_ff	mart vs rf	0.38	0.23
lang_nc	control vs mart	0.029	0.58
lang_nc	control vs rf	0.17	0.36
lang_nc	mart vs rf	0.32	-0.26

4.4 How does the combined Energy Impact of using Test Prioritisation compare to not performing this CI technique?

When summing together all of the energy usage from the different steps, the controls has significantly lower energy usage. Welch p values are omitted from Table 2. This is because for all comparison with the control, the values were so low they showed as 0.

While the energy consumption of the Test Prioritisation is proportionally closer to the control with the commit where tests fail (ff) for Imaging and Lang, that is not consistent with IO nor Codec.

Table 2: Pairwise Energy Usage Comparisons Between Control, Mart, and RF for the entire Test Prioritisation Pipeline

Combination	Comparison	Mean Difference	Cohen's d
codec_ff	mart vs control	77.73	2.58
codec_nc	mart vs control	70.47	2.24
codec_ff	rf vs control	68.65	2.95
codec_nc	rf vs control	75.28	2.18
imaging_ff	mart vs control	61.89	1.94
imaging_nc	mart vs control	75.99	2.32
imaging_ff	rf vs control	60.31	2.12
imaging_nc	rf vs control	82.36	2.60
io_ff	mart vs control	80.31	2.16
io_nc	mart vs control	69.41	2.12
io_ff	rf vs control	81.74	2.17
io_nc	rf vs control	76.69	2.33
lang_ff	mart vs control	80.86	1.93
lang_nc	mart vs control	94.25	2.07
lang_ff	rf vs control	88.53	2.13
lang_nc	rf vs control	94.38	2.25

5 Discussion

5.1 What is the Energy Impact of Training Test Prioritisation models?

On average, training the models on dataset ranging form 300 to 600 commits cost around 45 to 60 joules, Mart training outperformed Random Forest by a pretty significant margin

These values are pretty low, especially considering the fact that once a model has been trained it can be used for multiple

commits, how long it stays relevant depends on the amount of changes to the code that happen but it will often stay useful for a long time.

5.2 What is the Energy Impact of Ranking Tests using Test Prioritisation models?

Ranking using the models had an energy consumption ranging from 29 to 31 joules, this is still a low number but accumulates as it is required to be performed at every commit.

5.3 How does the Energy Impact of Running a Test Suite with Test Prioritisation compare to the Energy impact of Running a Test Suite without?

When running the test there were two cases, either the Test Prioritisation was irrelevant, costing a similar energy usage to simply running the tests, or the Test Prioritisation was a success. When that was the case, the energy saved was up to 20 at the extreme but closer to 10 joules for a normal success. This is less than the energy required to rank the tests to begin with.

The running of the tests itself cost between 260 and 570 joules.

5.4 How does the combined Energy Impact of using Test Prioritisation compare to not performing this CI technique?

Overall the Test Prioritisation did not reduce the energy consumption but rather increased it by 60 to 90 joules. The costs vastly outweighing the gains. Additionally, the times when there were gains are not always frequent. The data from Bertolino et al's [10] paper had at most 2.32% of commits containing tests failing, which means that the scenario where the prioritisation is successful would happen quite rarely.

There are important caveats to the results obtain by this experiment however:

- The datasets fairly narrow. The datasets being limited to Java Commons library not give a variety of cases, these projects are managed in a similar way, they also by their nature do not tend to have errors on the main branch often because the failing test would be resolved on individual dev and feature branches.
- Test Selection could be overwhelming the prioritisation. Test Selection is performed as part of the pipeline for this experiment, this is quite realistic to a real life environment as prioritisation is rarely done alone. Since it is performed, the commits only have a few tests to run, less than 20 in the commit with the most to run (Which is also the best performing case for test prioritisation) and sometimes only a single test, upon which test prioritisation is undeniably a waste of resources.
- There are few measurement A total of 8 situations are taken in count in this experiment from having 2 commits each for 4 datasets, the prioritisation was compared to running the tests alphabetically, in the 4 cases where there were failing tests, it is not impossible that the tests that fail come earlier alphabetically and thus the control runs as fast by hitting the failing test immediately. This could have happened the other way as well, leading to test running the failures at the end too. This uncertainty is one of the problems with this study.

In the end, the only conclusion available to this paper is that post Test Selection, Test Prioritisation is not a worthy investment. It may become more worthwhile with larger projects, or if Test Selection is not an option. But for projects similar to the ones used in this paper it is unnecessary if not detrimental to the project's Energy usage.

6 Responsible Research

6.1 Standard considerations

This project does not involve any participants beyond the author and thus does not need to take the related ethical considerations such as consent, privacy, or data protection. While the author tried to stay objective in their work, they are unaware of all the biases they may have which notably could have affected their discourse and conclusion. To minimise this factor, this paper tries to make clear what is factual data versus interpretation, the author also received feedback on the content of the paper from multiple sources.

Much of the code used in this paper, though a large proportion of it has been heavily modified, comes from the study by Bertolino et al [10] which is openly hosted on github. The datasets used in this study are open source and accessible to everyone.

6.2 Reproducibility

All scripts, datasets, and results used in this paper can be found at (Will be stup once on the server), while the code itself allows for a similar experiment to be performed there are several caveats to reproducibility that need to be mentioned:

- The measurements depend on the machine upon which they are collected, and so any outputs from a different machine and system will not be directly comparable to the results of this paper's experiment.
- EnergiBridge is an open source tool, however Understand is a paid service, while most people who would want to reproduce this study would likely be researchers or students, giving them access to Understand for free, that is not the case for all people who may want to do so.

6.3 Large Language Models usage

GitHub Copilot and ChatGPT were used to assist development and modification of the scripts to process the data in this paper. No large language models were used to assist the writing of this document, the research process.

The analysis of the data was assisted by the usage of LLMs, however interpretation was done completely by the researcher.

7 Conclusions and Future Work

7.1 Conclusion

This paper looked to examine Test prioritisation as a tool to save Energy however it has found that Test Prioritisation after

performing Test selection is more costly than Energy saving. In the case of this study costing up to 9 times more than the Energy saved.

While the results obtained by this study are difficult to generalise, it does suggest that there can be too much optimisation leading to a waste of Energy rather than achieving the reductions that are aimed for.

7.2 Future Research

There are numerous avenues of further studies from here. First and foremost, this field is scarcely explore yet, a similar experiment to the one performed in this study could be done on bigger datasets, or within a process without Test Prioritisation.

There is also an option to research Test Prioritisation on branches other than master, there tend to be more cases of test failing there and companies often refuse to run the pipeline at all on these branches until feature complete as the costs can stack up a lot. But by running Test Prioritisation on these partial changes it might be possible to catch errors early rather than at the end of that feature's development, which could maybe reduce cost and environmental impact at in the later steps of development.

Finally other Test Prioritisation methods could be attempted to see if the results correlate to those of this paper.

References

[1] A. Memon et al., "Taming Google-scale continuous testing," en, in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, Buenos Aires: IEEE, May 2017, pp. 233–242, ISBN: 978-1-5386-2717-4. DOI: 10.1109/ICSE-SEIP.2017.16. Accessed: Nov. 19, 2025. [Online]. Available: http://ieeexplore.ieee.org/document/7965447/.

[2] A. Zaidman, "An Inconvenient Truth in Software Engineering? The Environmental Impact of Testing Open Source Java Projects," in *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024)*, ser. AST '24, New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 214–218, ISBN: 979-8-4007-0588-5. DOI: 10.1145/3644032.3644461. Accessed: Nov. 18, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3644032.3644461.

[3] T. Yu and T. Wang, "A Study of Regression Test Selection in Continuous Integration Environments," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, ISSN: 2332-6549, Oct. 2018, pp. 135–143. DOI: 10.1109/ISSRE.2018.00024. Accessed: Jan. 25, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/8539076.

[4] O. Legunsen, F. Hariri, A. Shi, Y. Lu, L. Zhang, and D. Marinov, "An extensive study of static regression test selection in modern software evolution," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, New York, NY, USA: Association for Computing Machinery, Nov. 2016, pp. 583–594, ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2950361. Accessed: Jan. 25, 2026. [Online]. Available: https://dl.acm.org/doi/10.1145/2950290.2950361.

[5] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," en, in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, Santa Barbara CA USA: ACM, Jul. 2017, pp. 12–22, ISBN: 978-1-4503-5076-1. DOI: 10.1145/3092703.3092709. Accessed: Nov. 16, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3092703.3092709.

[6] A. Karatayev, A. Ogorodova, and P. Shamoi, *Fuzzy Inference System for Test Case Prioritization in Software Testing*, arXiv:2404.16395 [cs], Apr. 2024. DOI: 10.48550/arXiv.2404.16395. Accessed: Jan. 25, 2026. [Online]. Available: http://arxiv.org/abs/2404.16395.

[7] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino, "FAST approaches to scalable similarity-based test case prioritization," en, in *Proceedings of the 40th International Conference on Software Engineering*, Gothenburg Sweden: ACM, May 2018, pp. 222–232, ISBN: 978-1-4503-5638-1. DOI: 10.1145/3180155.3180210. Accessed: Nov. 16, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3180155.3180210.

[8] F. Palma, T. Abdou, A. Bener, J. Maidens, and S. Liu, "An Improvement to Test Case Failure Prediction in the Context of Test Case Prioritization," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE'18, New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 80–89, ISBN: 978-1-4503-6593-2. DOI: 10.1145/3273934.3273944. Accessed: Nov. 18, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3273934.3273944.

[9] M. Mahdieh, S.-H. Mirian-Hosseinabadi, K. Etemadi, A. Nosrati, and S. Jalali, "Incorporating fault-proneness estimations into coverage-based test case prioritization methods," *Information and Software Technology*, vol. 121, p. 106 269, May 2020, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2020.106269. Accessed: Nov. 18, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920300197.

[10] A. Bertolino, A. Guerriero, B. Miranda, R. Pietrantuono, and S. Russo, "Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20, New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1–12, ISBN: 978-1-4503-7121-6. DOI: 10.1145/3377811.3380369. Accessed: Nov. 18, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3377811.3380369.

[11] F. Kifetew, D. Prandi, and A. Susi, "On the Energy Consumption of Test Generation," in *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, ISSN: 2159-4848, Mar. 2025, pp. 360–370. DOI: 10.1109/ICST62969.2025.10989001. Accessed: Jan. 25, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/10989001.

[12] T. Durieux, *Energibridge: Energy measurement tool*, Accessed: 2026-01-25, 2023. [Online]. Available: https://github.com/tdurieux/EnergiBridge.

[13] *Understand: Static code analysis tool*, SciTools, 2024. [Online]. Available: https://scitools.com.