Dr. James Girard Summer Undergraduate Research Program
Faculty Mentor { Project Application

Ziad A. Al-Sharif

College of Aviation, Science, and Technology (CoAST)

Department of Engineering, Computing, and Mathematical Sciences (ECaMS)

Summer 2024

The Use of Deep Learning to Improve the Software Testing Process

1 Abstract

Software testing is a vital step in any software development process, as it helps to prevent software failures and enhance the software quality aspects such as reliability, usability, availability, etc. Software testing including regression testing can become costly and time-consuming. Test engineers are always looking to optimize the testing process by reducing the number of test cases without compromising the quality of the software product, which can become useless if testing is done poorly or the system is not tested thoroughly. Therefore, testing software often requires a large number of test cases (test suite) that takes time and e ort to design, manage, and verify. Over time, the testing process can become a challenging task given the large number of test cases, and the iterative and incremental nature of the testing process that fosters the addition of test cases over the lifespan of the development process. A test suite may become contaminated with redundant test cases, which are overlapping test cases that can be removed or reduced without compromising the software quality attributes. Nevertheless, there is no straightforward process for identifying these test cases and eliminating the overhead imposed by those useless and redundant test cases. However, in this project, we are trying to utilize modern machine learning techniques as a potential approach to tackle the optimization of these test suites. We hypothesize that the right machine-learning technique can help identify and eliminate redundant and highly similar test cases. This project will employ the execution trace of test cases. Each execution trace will correspond to building a call graph, all of which will be combined in a dataset of graphs. A graph is an abstract representation of the execution behavior of the test case that captures the code coverage structure from the execution of the test case. A machine learning model will be employed to nd the similarity scores between these abstract graphs. This score will help classify and rank the test cases by assigning a priority level for each test case. The resulting outcome is a set of clusters of test cases ranked based on their similarity scores. To ensure the most code coverage, a test engineer will be assigned to focus on the most diverse set of test cases. The lowest similarity score will indicate a high priority for the test cases, and the highest score will indicate a lower priority for that group of test cases. This allows the test engineer to con dently specify the highly diverse test cases based on their code coverage structure (low similarity score), the engineer will be able to pick and choose from the less diverse test cases indicated by the high similarity scores. This machine learning approach will allow the test engineer to gain a good understanding of how much testing would be enough to achieve the target quality level that is set for the target software project.

2 Introduction and Background

2.1 Introduction & Motivation:

Software testing is an essential part of any Software Development Life Cycle (SDLC), it reduces the risk of software failure and improves the overall software quality attributes such as reliability, usability, availability, etc. Inadequate software testing can negatively a ect the quality of the software system. However, software testing often involves a large number of test cases (test suite) that require e ort and time to manage and validate. Moreover, a test suite can easily include some of the redundant and overlapping test cases that could be reduced and eliminated without a ecting the overall quality of the software. Thus, nding these redundant or highly similar test cases is not an easy task considering the number of test cases under consideration and the incremental nature of designing, building, and adding these test cases during various stages of the development cycle.

Nevertheless, analyzing the test suite can be achieved by di erent means, one of which can employ the execution trace of various test cases [1]. A sequence of runtime execution events can be employed to uniquely identify test cases, and then these runtime execution events can help in the identi cation of execution behaviors of the test case and its code coverage structure, which can be used to build an abstract graph-based representation for that test case. These abstract graphs can help visually capture the internal behavior of the test case and would lead to identifying the similarities between these various test cases (this will lead to a graph representation of test cases). Visual graphs can help identify the degree of similarities between these test cases. However, the possibility of having hundreds or thousands of these graphs can make this visual analysis hard to achieve. Therefore, employing a machine learning

model to identify the similarity between various graphs can facilitate an automated process to help the test engineer identify, classify, and rank these test cases into levels or clusters.

Therefore, applying the right machine learning model to measure and predict the similarities between di erent test cases based on their graph representation can be valuable in improving the quality of the testing process and the quality attributes of the software under testing. Thus, this project will investigate di erent kinds of execution traces and evaluate various machine-learning techniques (models) to identify the similarities between di erent test cases. Ultimately, this project will allow test engineers to automatically detect, classify, and rank test cases based on their similarity measures. This will assist test engineers in determining the test suites that require more emphasis and those that are not very urgent due to their similarity with already considered test cases. Furthermore, the machine learning approach should present to the test engineers a similarity score, which will allow for systematic prioritization to rank the set of test cases with the most signi cant impact on the overall code coverage and ensure the highest quality of the system or a speci c quality attribute.

Additionally, employing a prioritization technique that ranks the test cases from the highest priority to the lowest can be helpful to the management of the testing process. The highest priority can be di erent based on the coverage criteria, but in general, it can be de ned based on the least similarity measures between any two test cases. Thus, this approach would lead to identifying the least common test cases and then selecting the most di erent test cases rst and keeping the most common with high overlapping scores until later time. Moreover, this approach will give the ability to build a behavioral model based on the relationship between their execution traces from their test cases.

In this project, we will build a dataset of execution traces for various test cases, automatically building call graphs for each of the given test cases, and combining these call graphs into a dataset of graphs. Most importantly, we will employ and evaluate various Machine Learning models to analyze various call graphs of various test cases. Then, we will be evaluating some of the machine learning models on those call graphs to nd the similarity measures. Utilize these similarities to identify the degree of uniqueness (*low similarity indicates high uniqueness and vice versa*) for

compensate for the de ciency that previous methods consider only feature information. This matrix is used to measure the di erence in a pair of nodes.

Recentlly, researchers in [22] presented Graph matching a deep learning-based approaches that have shown their superiority over the traditional solvers while the methods are almost based on supervised learning which can be expensive or even impractical. These researchers develop a uni ed unsupervised framework from matching two graphs to multiple graphs, without correspondence ground truth for training.

2.3 Project Goals and Objectives

Software testing is an essential part of any software development process. It ensures that the software under test meets the desired speci cations and quality standards. However, software testing can be time-consuming and costly, mainly due to the large number of test cases (test suite) to run. A test suite may include some of the redundant or highly similar test cases, which means they can be eliminated or reduced without a ecting the software quality. Finding these redundant or highly similar test cases is a di cult problem, considering the large number of test cases that may be involved. This project aims to using the execution trace of various test cases, creating a graph representation for each test case, and applying di erent machine-learning techniques to identify the redundant test cases.

This project aimed at developing and experimenting with various machine learning and deep learning models concerning graph understanding and similarities [8]. Call graphs will be collected from program execution during various test cases. These graphs will be collected in a dataset and used as input to the developed machine-learning models. The goal is to run the model for a variety of call graphs from various test cases.

This research aims to assist test engineers in identifying and ranking the most important set of test cases. Speci cally, we target the test suite to be considered in future regression testing and prioritizing the set of test cases by utilizing their previously captured execution traces. A dataset of execution traces and its graph represenation is to be built, it should associate each test case with its unique sequence of the execution trace and call graph. Then, various Machine Learning models will be evaluated to identify redundant test cases, classify test cases, predict and identify the similarity measures between di erent test cases based on their runtime behavior, and nally prioritize and rank the set of test cases based on their similarity measures obtained from graph similarity. Therefore, the objectives of this project are manifold:

1. Abstraction of the test cases in the form of call graphs driven the execution traces, each test case is associated with its sequence of execution events. Di erent execution events are to be considered.
2. Building a dataset with di erent graphs for di erent test cases.
3. Building a similarity measure between di erent test cases based on their sequence of execution traces.
4. Evaluating di erent Machine Learning models to identify the similarities between di erent execution traces and therefore between their corresponding test cases. The machine learning model should be able to identify/predict the similarity measure between any set of test cases, and rank the test cases based on their similarity.
5. Evaluating di erent Machine Learning models to prioritize the test cases in a test suite. The machine learning model should be able to take a set of test cases and then rank them based on their priority or similarity.
6. Evaluating di erent Machine Learning models to identify the redundant test cases in the test suite. It may include the

3 Methodology and Approach

In this project, we will employ and evaluate various Machine Learning models to analyze various call graphs of various test cases. Building a dataset of call graphs and publishing it in a respected venue. It will include combining a dataset of execution traces for various test cases, automatically building call graphs for each of the given test cases, combining these call graphs into a data set, and evaluating some of the machine learning models on those call graphs to nd the similarity measures. Utilize these similarities to identify the degree of uniqueness for each of the undertaken test cases, this will help the test engineer to rank the test cases from the most important to least important based on the similarity measures re ected in their code coverage and execution behavior, see Figure 1.

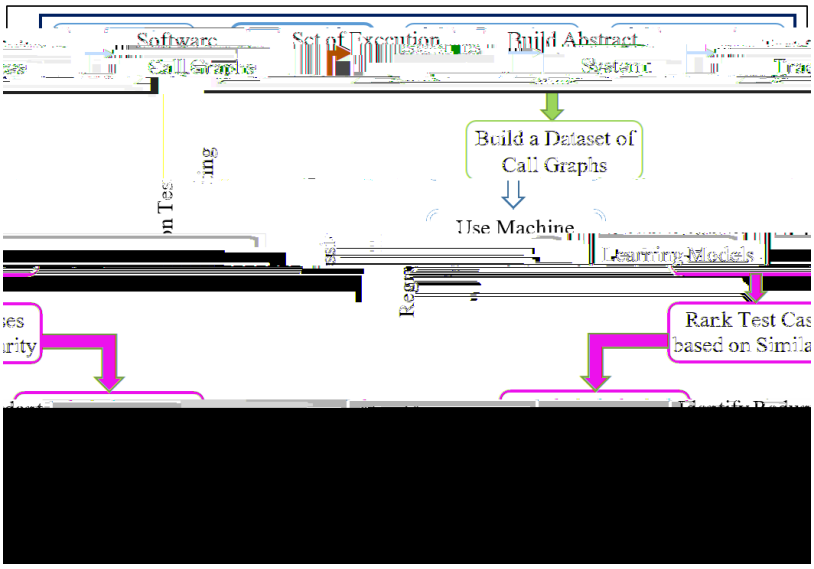


Figure 1: Proposed methodology and approach

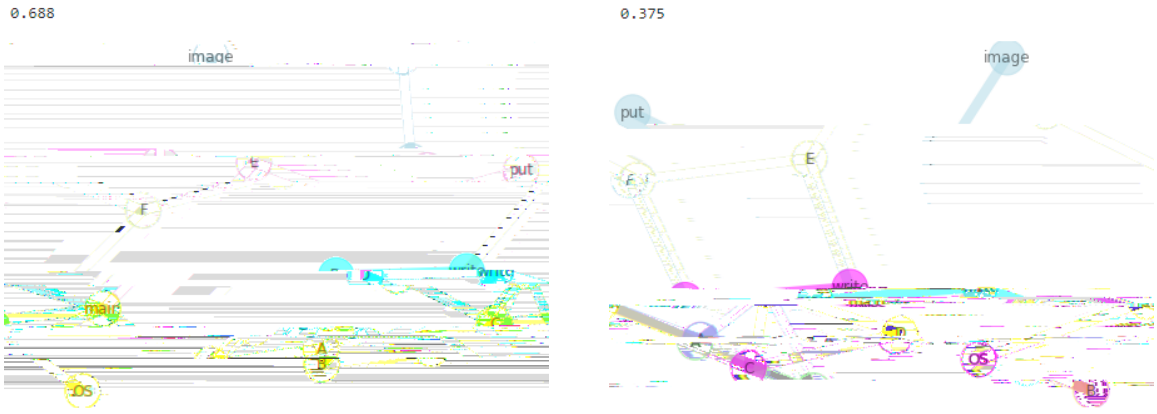
3.1 Application Framework

The proposed approach will employ program execution traces to advance the quality of the testing process. This project intends to use a research programming language called Unicon [2] and its Alamo [3] monitoring framework. This framework, besides other tracing frameworks, will be used to build di erent datasets. Nevertheless, experimentation with the Unicon language does not limit the generalization of the approach to other systems and languages nor does it prevent its applicability for other domains. The proposed approach should be general and applicable during any software testing process. The only limitation is the availability of a su cient monitoring and execution tracing mechanism. Furthermore, execution traces for other languages can be performed by di erent techniques and tools, most of which depend heavily on the running platform and the language itself [4]. For example, Dynamic Tracingis or (*DTrace*) is one of the common tracing tools that provides a system-wide tracing capability that applies to various kinds of software systems and platforms [5]. On the other hand, mainstream programming languages such as *Java*, *Python*, and *C* provide their tracing tools, frameworks, or packages.

3.2 Execution Trace: Runtime-Events & Dataset

Execution trace will add new insights into the testing process. The use of execution trace can be independent of the programming language or the host platform. However, considering the huge number of events that can be produced during the execution of a test case, the undertaken experimental programming language (*Unicon language*) is already equipped with a monitoring framework; Uniocn's virtual machine[1] (Unicon's VM) is already instrumented with a set of 118 kinds of unique run time events. Monitoring all execution events can signi cantly hinder the tracing process by the imposed overhead on the execution time of the target program. However, what makes Unicon language a

[1]Unicon Language:http://unicon.org



(a) G1 (b) G2

Figure 2: Sample Call Graphs

platform for experimentation is its dynamic ltering mechanism that allows the monitor program to dynamically select a subset of the runtime events to be traced. Moreover, each event has a code and value, the monitoring API allows the monitor program to specify the type and number of traced events based on both the event code and value. This ltering mechanism allows the test engineer to select and set of monitored events on the y during execution while the target program is being monitored; any change in the set of monitored events will be immediately re ected on the set of reported events; this makes it ideal for experimentation.

Moreover, in this project, we will experiment with various runtime events, the goal is to provide the test engineers with a representative set of execution events that can help uniquely identify the test cases and at the same time minimum enough to reduce the overhead imposed by the monitoring process. For example, events such as *E_Line*

Plans for engaging with the student throughout the research process:

- I will be present on campus a minimum of one day a week. During these times I will carry out hands-on training and meetings with the student as needed.
- The student will also participate in a weekly meeting that may occur over Zoom to explain and discuss various emerging questions and discussions.

Plans for developing speci c skills and techniques during mentorship: I will provide hands-on:

- Designing test cases for various programs
- Building execution traces for various test cases
- Building call graphs from execution traces
- Building a dataset of the execution graph
- Training and evaluation machine learning and deep learning techniques to measure graph similarities

The student will be trained directly on all techniques by myself or a peer.

Connection to Programs at Home Academic Institution (Lewis University): This project is not only directly related to my role at Lewis University and to my research areas of interest, but also it de nitely helps me to create more opportunities for Lewis University and for my students at the Department of Engineering, Computing, and Mathematical Sciences (ECaMS) in the forms of funded research opportunities. One of the future goals is to submit an NSF research grant. Thus, this project will signi cantly contribute in creating new opportunities for students and faculty.

5 Proposed Timeline

The following is a tentative timeline along with the project's goals:

Dates (Approximate) Project's Goals :

- June 10: Read a set of papers related to project topic along with other background materials
- June 17: Draft of Introduction, Methods, and Literature Review (assign June 14)
- June 24: Collect a set of execution traces from various test cases and develope a program to automatically generate call graphs for each of these test cases. Combineing the resulting call graphs in one dataset.
- July 1: Develop and experiment with di erent machine learning and deep learnning models that are able to predict some of the similarities betwee various graphs.
- July 8: Design a robust classi er to rank a graph based on the adopted similarity measures.
- July 15: Mock PowerPoint presentation
- July 22: Draft of Technical Report including all results and analysis (assign July 1; after the Introduction, Method, and Literature Review draft). Alternatively, the student may create a PowerPoint Presentation along with the Project's Abstract.
- July 29: Final Technical Report and Final PowerPoint presentation and the Project's Abstract.
- August 5-12th: Summer Research Symposium Presentation

6 Criteria for student applicants:

Accepted student(s) should have an intermediate computer programming skills. The student should know Python.

List of seminar topics you are willing to cover: See the highlighted topics in yellow:

- Ethics in Research/Scholarship
- Literature Search and Library Resources
- Problem-Solving Skills and the Scienti c Method
- Presentation Skills
- Data Analysis and Data Management
- Technical Writing
- Resume Writing and Marketing
- Preparing for Graduate School
- Interview Skills
- Mock Presentation Supervisor (Practice for Symposium)
- Other:

7 Future Research Directions

As a future research direction for this project, we will incorporate more execution events and try to extend our approach to larger and real-time programs. The nal product will be a robust measurement-modeling tool that can be adopted by software developers and test engineers to improve the e ciency of the regression testing process, reduce the testing time, and improve the quality of the software products in terms of reliability, availability, robustness, and more.

References

[1] Z. A. Al-Sharif, W. F. Abdalrahman, and C. L. Je ery, \Encoding test cases using execution traces," in *2021 12th International Conference on Information and Communication Systems (ICICS)*, pp. 239{244, IEEE, 2021.

[2] C. Je ery, S. Mohamed, J. Al Gharaibeh, R. Pereda, and R. Parlett, *Programming with Unicon*. GNU Free Documentation License, 2018.

[3] C. L. Je ery, *Program monitoring and visualization: an exploratory approach*. Springer Science & Business Media, 2012.

[4] E. Bousse, T. Mayerhofer, B. Combemale, and B. Baudry, \Advanced and e cient execution trace management for executable domain-speci c modeling languages," *Software & Systems Modeling*, vol. 18, no. 1, pp. 385{421, 2019.

[5] C. Chen, J. Ma, T. Qi, B. Cui, W. Qi, Z. Zhang, and P. Sun, \Firmware code instrumentation technology for internet of things-based services," *World Wide Web*, vol. 24, no. 3, pp. 941{954, 2021.

[6] A. Rahmani, J. L. Min, and A. Maspupah, \An evaluation of code coverage adequacy in automatic testing using control ow graph visualization," in *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 239{244, IEEE, 2020.

[7] H. Hemmati, \How e ective are code coverage criteria?," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, pp. 151{156, IEEE, 2015.

[8] G. Ma, N. K. Ahmed, T. L. Willke, and P. S. Yu, \Deep graph similarity learning: A survey," *Data Mining and Knowledge Discovery*, vol. 35, pp. 688{725, 2021.

[9] A. Bajaj and O. P. Sangwan, \Test case prioritization using bat algorithm," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 14, no. 2, pp. 593{598, 2021.

[10] N. Gokilavani and B. Bharathi, \Test case prioritization to examine software for fault detection using pca extraction and k-means clustering with ranking," *Soft Computing*, vol. 25, no. 7, pp. 5163{5172, 2021.

[11] A. Bajaj and O. P. Sangwan, \Discrete and combinatorial gravitational search algorithms for test case prioritization and minimization," *International Journal of Information Technology*, vol. 13, no. 2, pp. 817{823, 2021.

[12] S. Messaoudi, D. Shin, A. Panichella, D. Bianculli, and L. C. Briand, \Log-based slicing for system-level test cases," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 517{528, 2021.

[13] A. Panday, M. Gupta, M. K. Singh, and N. Ali, \Test case redundancy detection and removal using code coverage analysis," *MIT Int. J. Comput. Sci. Inf. Technol*, vol. 3, no. 1, pp. 6{10, 2013.

[14] E. K. Mece, H. Paci, and K. Binjaku, \The application of machine learning in test case prioritization-a review," *European Journal of Electrical Engineering and Computer Science*, vol. 4, no. 1, 2020.

[15] A. A. Saifan *et al.*, \Test case reduction using data mining classi er techniques.," *J. Softw.*, vol. 11, no. 7, pp. 656{663, 2016.

[16] A. Pandey and K. Malviya, \Enhancing test case reduction by k-means algorithm and elbow method," *International Journal of Computer Sciences and Engineering*, vol. 6, no. 6, pp. 299{303, 2018.

[17] V. Chaurasia, Y. Chauhan, and K. Thirunavukkarasu, \A survey on test case reduction techniques," *International Journal of Science and Research (IJSR)*, 2014.

[18] L. Xiao, H. Miao, T. Shi, and Y. Hong, \Lstm-based deep learning for spatial{temporal software testing," *Distributed and Parallel Databases*, vol. 38, no. 3, pp. 687{712, 2020.

[19] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gauzere, and M. Vento, \Graph edit distance as a quadratic assignment problem," *Pattern Recognition Letters*, vol. 87, pp. 38{46, 2017.

[20] H. Bunke and G. Allermann, \Inexact graph matching for structural pattern recognition," *Pattern Recognition Letters*, vol. 1, no. 4, pp. 245{253, 1983.

[21] C. Liu, D. Niu, X. Yang, and X. Zhao, \Graph matching based on feature and spatial location information," *The Visual Computer*, vol. 39, no. 2, pp. 711{722, 2023.

[22] R. Wang, J. Yan, and X. Yang, \Unsupervised learning of graph matching with mixture of modes via discrepancy minimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.