International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 03 | Mar 2021 www.irjet.net p-ISSN: 2395-0072

Automated Unit Test Cases Generation using Machine Learning

Prajwal Wable[1], Mangesh Kumar[2], Shubham Thorat[3], Omkar Gaikwad[4], Prof. S. S. Kolte[5]

[1,2,3,4] *Undergraduate Students, Dept. of Computer Engineering, AISSMS COE, Pune, Maharashtra, India*
[5]*Professor, Dept. of Computer Engineering, AISSMS COE, Pune, Maharashtra, India*

Abstract - *Software has become a necessary part of daily lives. Millions of lines of codes are written and executed every day all around the world. Development tools are key factors to the success of the tech industry as the current time constraints put a lot of pressure on developers to produce high-quality software. Many of these tools developed are based on decades of academic research which is mostly articulated around a deductive-logic approach. Automated Unit Test Cases generation is one of the fields that has been the focus of extensive literature within the research community. The existing approaches usually rely on test coverage criteria, generating synthetic test cases. These test cases challenging to read and understand for developers. In recent years there is a paradigm shift towards data-driven statistics-based research. In this paper, we propose a new approach that aims at generating unit test cases by learning from developer written test cases using machine learning. The training dataset will be mined from open-source repositories hosted on GitHub. We will use this dataset to train a machine learning model to translate code snippets to the corresponding test cases.*

Key Words: Automated Unit Test generation, Unit testing, Machine-Learning, Attention and Transforms, Deep Learning.

1. INTRODUCTION

Software testing is one of the important process in software development life cycle process. Software testing helps in development of good quality software. It is achieved by unit test cases or test suites. Test case measures the errors and capability or efficiency of program. Since a long time test cases are generated manually or software tested manually due to which there occurs some errors in programs and it results into bad quality software and we cannot improve software quality.

The process of generating and maintaining unit test case is very complicated, time-consuming, and costly and developer gets frustrated at time of testing process. Hence by automatic unit test case generator we can improve the efficiency of software with low cost and in short time. So to achieve this we are presenting approach for generating automatic unit test case using deep learning.

Types of Testing:

There are various types of testing in software development life cycle (SDLC) such as functional testing, non-functional testing and many more. Here we are focusing on some types of testing such as Unit Testing, White box testing, Black box testing etc.

1.1 UNIT TESTING

Unit Testing is sub part of functional testing [1]. It plays important role in software testing process. In unit testing the software is divided into smaller testable units to analyse the source code. It can be carried out for functions, procedures or methods in Object Oriented Programming (OOP) and procedural programming.

1.2 WHITE BOX TESTING

White box testing is type of testing which uses source code to create test cases. It focuses on internal working of software.

2. RELATED WORK

Common methods for code-based test case generation are random testing [2], search-based testing [3], and symbolic testing [4]. This paper [2] benchmarked random testing and search-based testing on the closed source project [2] and found that search-based testing had at most 56.40% effectiveness, while random testing achieved at most 38%. The following are existing tools that generate automated unit test cases:

2.1 RANDOOP

Randoop [5] is a self-unit test generator for java language. It is an automated unit test generator for any class of java in Junit format. It uses feedback directed random test generation for generating unit tests. This approach smartly produces sequence of methods for the class. Randoop performs the sequence it creates making use of the results of execution to produce assertions that capture the behaviour of the program. It then creates test cases from the code sequences and assertions. After test generation and test execution they result in highly effective automated test generation.

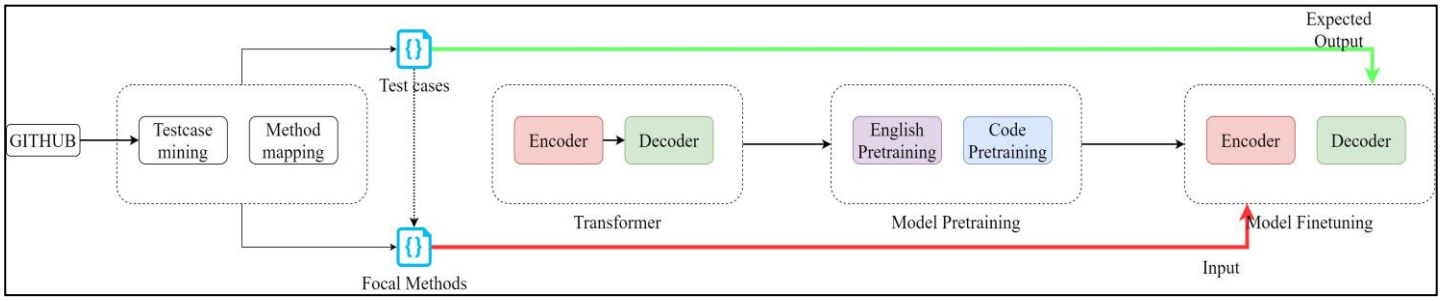

Figure 1: Overview of proposed system

2.2 EVOSUITE

It is a similar tool that generates test cases for java automatically. Evosuite [6] is a progression algorithm to produce Junit tests. It can be used via command line and also it has extension to integrate in software like Eclipse, Maven etc. To accomplish this Evosuite uses a hybrid approach to generate test cases. After test cases are generated Evosuite suggests possibilities by adding small sets of assertions to summarize current behavior. It is open source and freely available to modify. Evosuite is openly handed and supported by Google focused research test amplification.

2.3 PONICODE

It is Artificial Intelligence based unit test generator platform for JavaScript which focuses on modifying and visualizing the unit test cases [7]. Ponicode uses a machine learning algorithm which help to test each and every test cases including edge cases. It first accesses the JavaScript file it tries to check if it can be used to assisted with the Ponicode then it provides test case suggestion using Artificial Intelligence, apart from this Ponicode creates and writes the test file using correct jest syntax. It also provides an extension to VScode. It is an AI powered extension for automated test case generation for a given JavaScript code.

3. PROPOSED SYSTEM

Figure 1 provides an summary of the our approach. Starting dataset of open-source codebases mined from GitHub, we mine test cases and map them to the corresponding focal methods. Finally, we fine-tune a transformer model, which will be pre-trained on the English language as well as on the source code corpora, for the task of generating unit test cases.

3.1 DATA COLLECTION

The aim of this stage is to collect test cases and their focal methods (i.e., the methods those test cases are written for.) from a set of mined projects. The projects will be open source projects on GitHub with recent commits and good number of stars. We can parse each repository to obtain methods. We then identify each test class and its corresponding focal class. Lastly, for each test case, we will map the test case to the corresponding focal method obtaining a set of mapped test cases.

For mapping the test classes to focal methods following techniques will be used:

Path Matching: The best practice is to place the test cases in similarly named directory in the test folder. For example the test cases for the class *src/main/java/Bar.java* will be saved in *src/test/java/FooTest.java.* Taking advantage of this practice we can map test classes to the focal classes.

Name Matching: The name of a test class is often written as the name of the focal class with a "Test" prefix or suffix. For example, if the name of class is Bar.java the test would be named as TestBar.java or BarTest.java. Using this pattern, we can map the test and focal classes.

The class for which no test cases are found are discarded.

3.2 PRETRAINING

We will be using the BART transform model [8] as shown in figure 2, that uses encoder decoder for sequence to sequence translation to generate the test cases [9]. The pretraining will be done in 2 semi-supervised stages on English text and then on Code Pretraining.

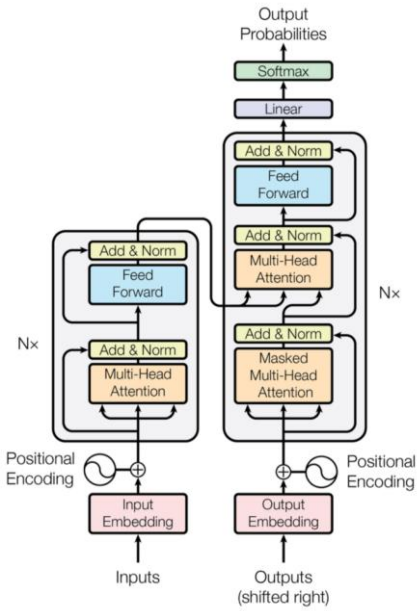

Figure 2: Transformer model architecture [8]

3.2.1 ENGLISH PRETRAINING

The model will first be trained on a large corpus of English text data. The aim of this stage is to let the model learn the semantic and statistical properties of natural language. The BART is trained unsupervised by corrupting the text and reconstructing the original text.

3.2.2 CODE PRETRAINING

The model will next be trained on the code data that was mined earlier from GitHub. The aim of this stage is to let the model learn the syntax and code properties.

3.3 FINETUNING

In this stage optimize the model on the task of creating unit test cases for a given method is done. Specially, we represent this task as a translation task, where the source is a focal method and the target is the corresponding test case originally written by a software developer.

4. FUTURE SCOPE

The model we have built is no way the most efficient and advanced work in this field. Therefore, there is still scope for improvement. In future work, with more study and research the performance of the model can be improved by including more data and novel loss functions. For now, due to limitations we have only developed the model for a single programming language, in future we can make it compatible with all testing framework and languages.

5. CONCLUSION

We present a deep learning based approach that aims at generating unit test cases by learning from real-world, developer-written test cases. Our approach relies on a deep learning model which will be pre trained both on English and program source code fine-tuned on the task of generating test cases given a method under test.

ACKNOWLEDGEMENT

We would like to acknowledge the partial support of our project guide Prof. S. S. Kolte, serving as a Professor in the All India Shri Shivaji Memorial Society's College of Engineering, Pune-01. She provided proper guidance time to time and her suggestions stood noteworthy in making this paper complete

REFERENCES

[1] M. Cohn, Succeeding with agile: software development using Scrum. Pearson Education, 2010.

[2] M Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Janis Benefelds. An industrial evaluation of unit test generation: Finding real faults in a financial application. In Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track, 2017.

[3] Gordon Fraser, Jose Miguel Rojas, Jose Campos, and Andrea Arcuri. Evosuite at the sbst 2017 tool competition. In Proceedings of the 10th International Workshop on Search-Based Software Testing, 2017.

[4] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In OSDI, volume 8, pages 209-224, 2008.

[5] C. Pacheco and M. D. Ernst, "Randoop: feedback-directed random testing for Java," in Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion - OOPSLA '07, vol. 2. New York, New York, USA: ACM Press, 2007, p. 815. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1297846.1297902

[6] G. Fraser and A. Arcuri, "Whole Test Suite Generation," IEEE Transactions on Software Engineering, vol. 39, no. 2, pp. 276–291, feb 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6152257/

[7] Hamza Sayah. Machine Learning on Source Code [Online].Available: https://www.ponicode.com/blog/machine-learning-on-source-code.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," CoRR, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[9] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," 2019.

BIOGRAPHIES

PRAJWAL WABLE,
Student at AISSMS COE, Pune-01.
Savitribai Phule Pune University,
B.E. in Computer Science.

MANGESH KUMAR,
Student at AISSMS COE, Pune-01.
Savitribai Phule Pune University,
B.E. in Computer Science.

SHUBHAM THORAT,
Student at AISSMS COE, Pune-01.
Savitribai Phule Pune University,
B.E. in Computer Science.

OMKAR GAIKWAD,
Student at AISSMS COE, Pune-01.
Savitribai Phule Pune University,
B.E. in Computer Science.

PROF. S. S. KOLTE,
Professor, Dept. of Computer Engineering,
AISSMSCOE, Pune-01.