IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁘ Peer-reviewed & Refereed journal ⁘ Vol. 11, Issue 4, April 2024

DOI: 10.17148/IARJSET.2024.11414

Code Generation and Automated Bug Fixing

M. Anitha[1], Shaik Hazarabi[2], Neeraja kunala [3], P. Deekshitha[4]

M.Tech, Computer science & Engineering, Bapatla women's Engineering College, Bapatla, India[1]

B.Tech, Computer science & Engineering, Bapatla Women's Engineering College, Bapatla, India[2-4]

Abstract: Major standard languages such as Codex have demonstrated the ability to generate code for a wide variety of tasks. However, current models have limited performance, especially on complex tasks. One reason for this is that the language model does not understand the context of the program, causing the program to be buggy or even fail. In this article, we investigate whether automatic correction (APR) can correct the solutions produced by the language model in the LeetCode competition. The aim is to investigate whether the APR technique can increase the reliability of code generated from large language samples. Our research shows that: (1) retrieved developer code shows faults in man-made solutions and shows that APR technology can fix development rights; (2) updated Codex format that supports code fixing similar to or better than the existing Java fixing tool TBar and Logger for providing evidence of bug location, providing bug information, fixing bugs. By analyzing the experimental results produced by this tool, we make several recommendations: (1) The APR tool needs to be improved to overcome the limitation of patch area location (e.g., displaying more local crime); (2) Due to the large size of the language model, more correction models can be obtained by training more data, and future APR tools can shift the focus by adding additional models to the link structure/content, (3) By combining the language structure with APR, it is amenable to link, learning.

Keywords: Bug fixing techniques, automated and semi-automated, solutions, testings.

I. INTRODUCTION

The main benefit of bug tracking is to provide accurate, important details about the development request (bugs and improvements, including areas that are often blurry) and their status. The master list of backlogs (often called the backlog) provides useful input when defining a product or perhaps just the "next release." In a business environment, error systems can be used to generate reports about products. Programmers fix bugs in production. However, sometimes this can lead to incorrect results because the parameters may have different weights and difficulties. The severity of the error will not directly affect the difficulty of fixing it. There may be differences of opinion between managers and architects.

BACKGROUND & RELATED WORK

The purpose of our article is to identify all automation and debugging solutions that can solve software problems. These errors can occur during the requirements, design and coding, implementation, and maintenance phases of the software development lifecycle (SDLC). In our study, we consider various errors that occur at each stage of SDLC. Errors may occur during the assembly, design, coding and operation stages [11]. In our study, errors, defects, mistakes and malfunctions are considered as errors. We have identified different processes, methods, procedures, methods and tools that can fix bugs at different stages of the software development life cycle and include various solution problems.

The first step in the repair process and the goal of troubleshooting is to identify the code causing the malfunction. The second is a misunderstanding that will help understand the root of the problem. Finally the bug was fixed; this is the process of modifying the code to eliminate the root cause [41]. All these activities are collectively called debugging. Many debugging techniques have been proposed to help software developers debug software.

II. METHODOLOGY

The main purpose of this work is to bring together all automatic debugging and automatic error correction in software e ngineering, 15 methods, procedures and methods (such as solving problems). Automatic error correction technology tha t helps reduce correction and measurement costs.

Problem Definition:

DOI: 10.17148/IARJSET.2024.11414

All apps have bugs; As long as people write code, bugs will appear in software. Some errors are minor, while others are serious. Open source programs such as Word Press require the participation of the user community to identify software bugs and plan new features.

[1.1] Presently Available System:

In the existing system, the project manager assigns the projects to the developers. The developers develop the projects as per customer requirements. The project manager itself assigns the developed applications to the tester for testing. In the testing phase, when the tester encounters no. of bugs then he reports to the project manager and developer about the bug information. Bottlenecks of the Existing System:

- •The tester report which is called “Bug Report” is in the form of physical document. If the document is damaged then the total information about the bug will be lost.
- •The bug information is not stored in the database for future reference.

[1.2] Proposed System:

Proposed System: The purpose of the Automated Bug fixing System is to test the application for the bugs and report it to the project manager and developer. The main intention behind the Automated Bug fixing System is that to track bugs and Corrects for them. Store the bug information with a unique id in the database for future reference. So, this makes the job of handling the bugs easy.

The main objectives of the Automated Bug fixing System are:

- •Identifying the bugs in the developed application.
- •No bug will be unfixed in the developed application.
- •Not merely identifying the bugs but also providing the bug Solutions.
- •As soon as the bugs are identified. They are reported to the project manager and developer.
- •To ensure that who needs to know about the bug can learn soon after it is corrected.

[1.3] Viability of the System:

A viable Automated Bug fixing System should permit an administrator to formulate permissions related to status, move the bug to another status, or delete it. Certain systems also notify interested parties when additional status changes have been made.

A Automated Bug fixing System should also provide a comprehensive overview of development requests that have been made and their current status. In addition, a list of pending items, whichis prioritized, provides valuable information for everyone involved in producing the software.

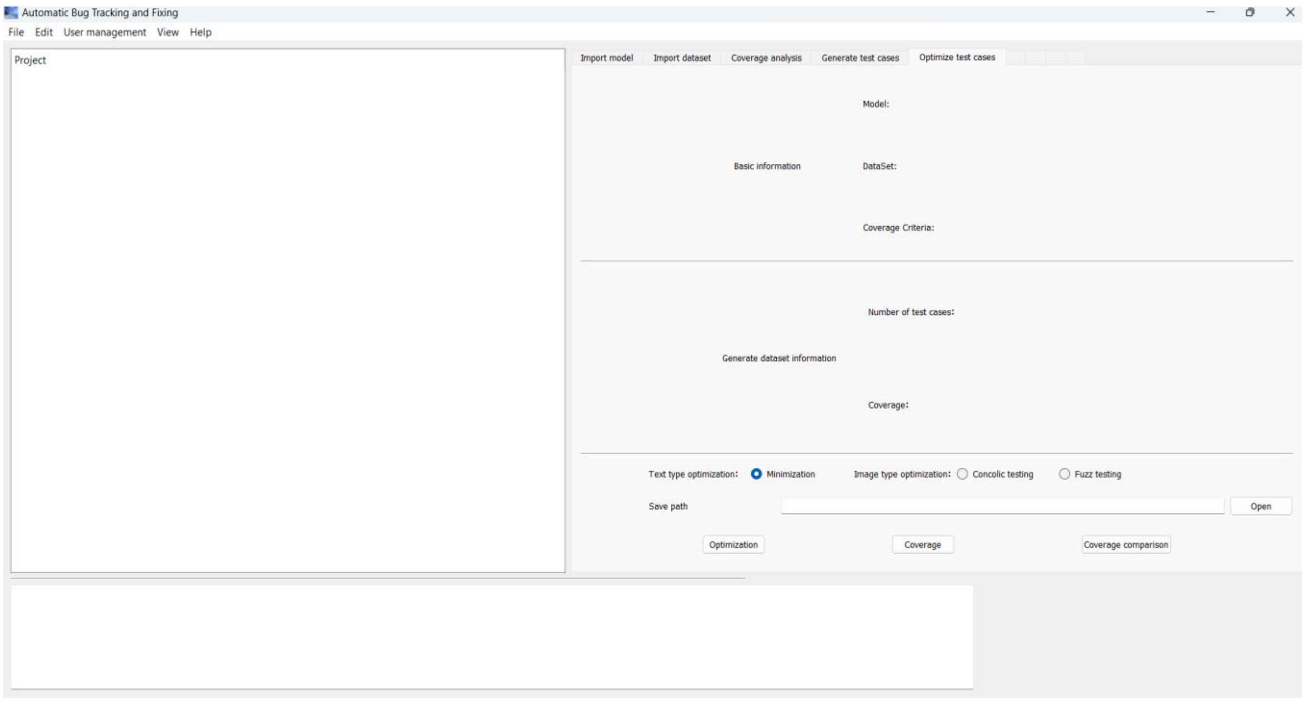
In a large organization, it can generate reports related to the productivity a programmer who has been assigned to repair the bugs, keeping in mind that these problems vary in complexity and severity, and that managers and program designers sometimes disagree on the best approach to take. Application support professionals often use a local bug tracker (LBT) as well to monitorusers’ complaints that may be irrelevant in the actual software development process.

The software is a web application which can be run on a windows operating system having Javaplatform (JDK 1.6 or higher) .

The users of Automated Bug fixing System:

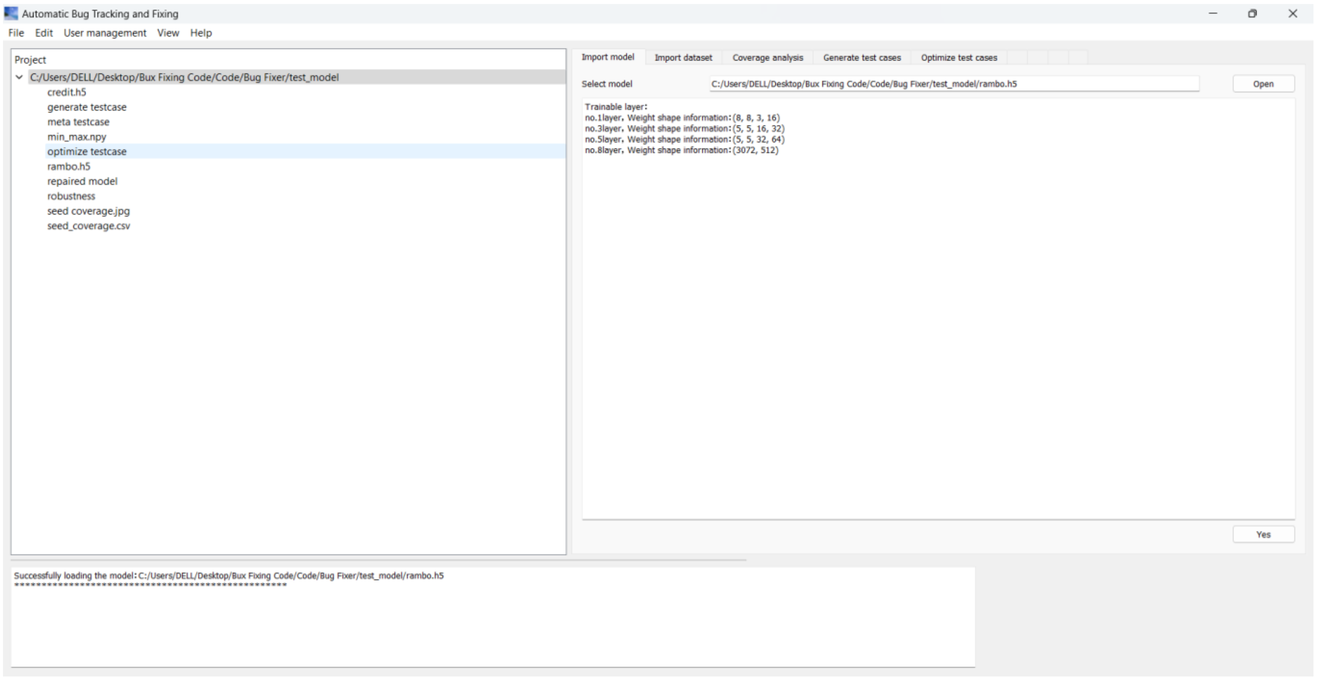
- •Project Manager
- •Developer
- •Tester

RESULTS:



Text type optimization: [✓ Minimization] Image type optimization: Concolic testing Fuzz testing

Fig 1: Window after running project in command Prompt



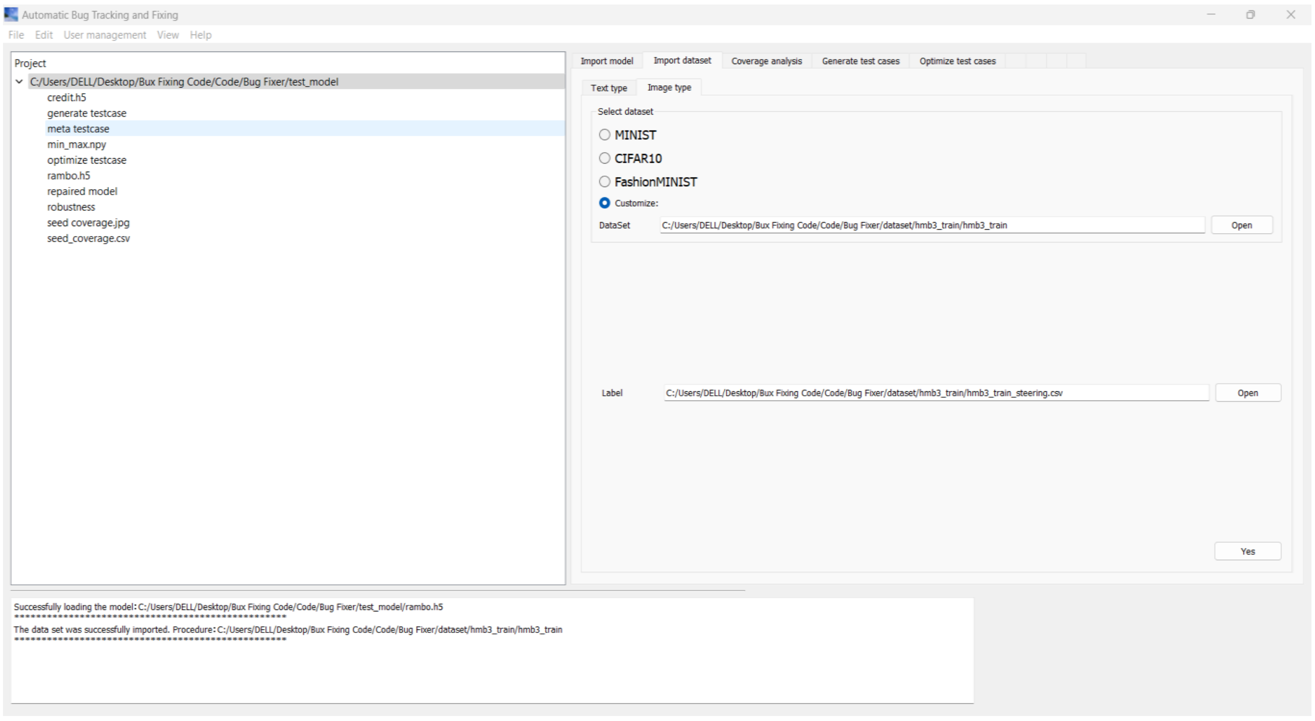
Project
- C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model
  - credit.h5
  - generate testcase
  - meta testcase
  - min_max.npy
  - optimize testcase
  - rambo.h5
  - repaired model
  - robustness
  - seed coverage.jpg
  - seed_coverage.csv

Select model: C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model/rambo.h5

```
Trainable layer:
no.1layer, Weight shape information:(8, 8, 3, 16)
no.3layer, Weight shape information:(5, 5, 16, 32)
no.5layer, Weight shape information:(5, 5, 32, 64)
no.8layer, Weight shape information:(3072, 512)
```

```
Successfully loading the model:C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model/rambo.h5
**************************************************
```

Fig 2:After selecting Project the model would be Successfully loaded



Project
C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model
credit.h5
generate testcase
meta testcase
min_max.npy
optimize testcase
rambo.h5
repaired model
robustness
seed coverage.jpg
seed_coverage.csv

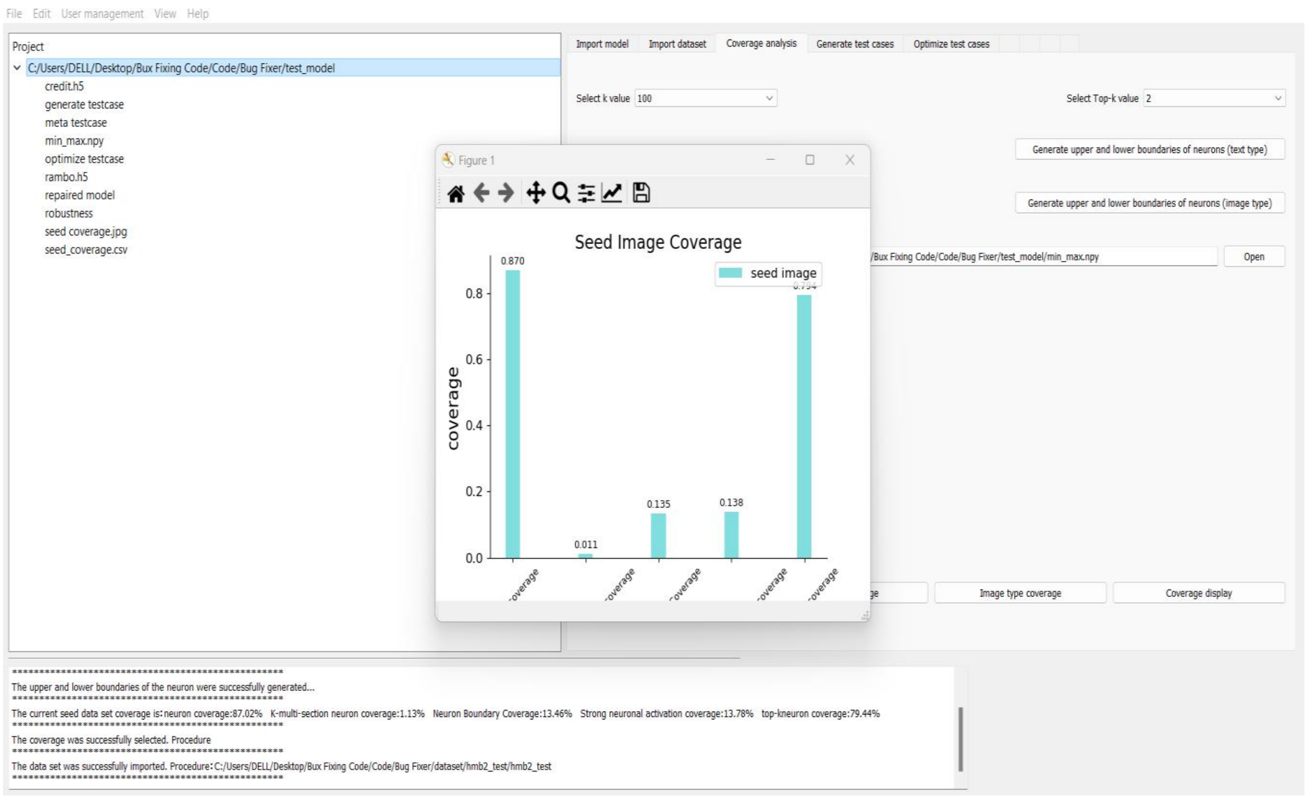
Select dataset: ○ MINIST ○ CIFAR10 ○ FashionMINIST [✓ Customize:]

DataSet C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/dataset/hmb3_train/hmb3_train

Label C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/dataset/hmb3_train/hmb3_train_steering.csv

```
Successfully loading the model:C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model/rambo.h5
*****************************************************
The data set was successfully imported. Procedure:C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/dataset/hmb3_train/hmb3_train
*****************************************************
```

Fig 3: Import datasets from the model



Project
C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model
credit.h5
generate testcase
meta testcase
min_max.npy
optimize testcase
rambo.h5
repaired model
robustness
seed coverage.jpg
seed_coverage.csv

Select k value 100

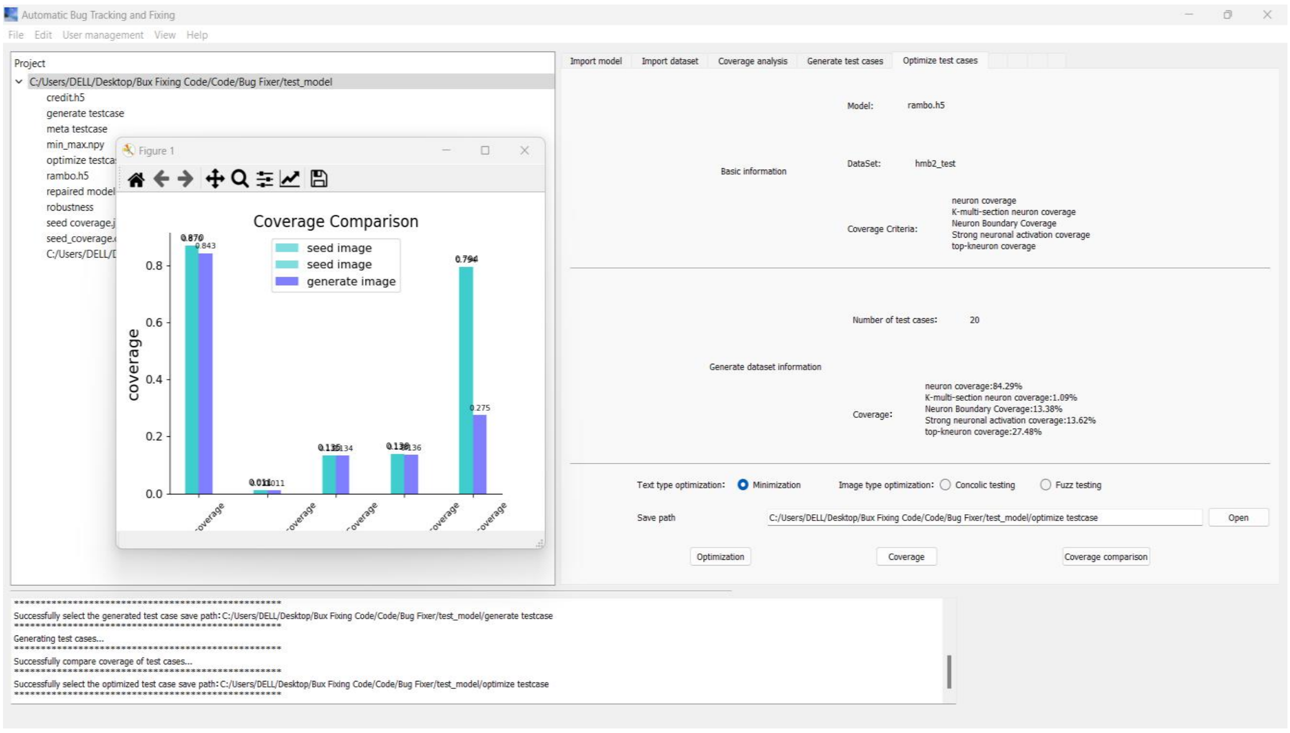
Select Top-k value 2

/Bux Fixing Code/Code/Bug Fixer/test_model/min_max.npy

```
*****************************************************
The upper and lower boundaries of the neuron were successfully generated...
*****************************************************
The current seed data set coverage is:neuron coverage:87.02%   K-multi-section neuron coverage:1.13%   Neuron Boundary Coverage:13.46%   Strong neuronal activation coverage:13.78%   top-kneuron coverage:79.44%
*****************************************************
The coverage was successfully selected. Procedure
*****************************************************
The data set was successfully imported. Procedure:C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/dataset/hmb2_test/hmb2_test
*****************************************************
```

Fig 4: The Coverage was successfully Selected and Displayed

DOI: 10.17148/IARJSET.2024.11414



Basic information

Model: rambo.h5

DataSet: hmb2_test

Coverage Criteria:
neuron coverage
K-multi-section neuron coverage
Neuron Boundary Coverage
Strong neuronal activation coverage
top-kneuron coverage

Generate dataset information

Number of test cases: 20

Coverage:
neuron coverage:84.29%
K-multi-section neuron coverage:1.09%
Neuron Boundary Coverage:13.38%
Strong neuronal activation coverage:13.62%
top-kneuron coverage:27.48%

Text type optimization: [✓ Minimization] Image type optimization: Concolic testing Fuzz testing

Save path C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model/optimize testcase

```
******************************************************
Successfully select the generated test case save path:C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model/generate testcase
******************************************************
Generating test cases...
******************************************************
Successfully compare coverage of test cases...
******************************************************
Successfully select the optimized test case save path:C:/Users/DELL/Desktop/Bux Fixing Code/Code/Bug Fixer/test_model/optimize testcase
******************************************************
```

Fig 5:The Coverage Camparisons of 2 different Datasets

III. CONCLUSION

In this project, we examine errors created by programs using standard languages such as Codex and investigate whether automatic program correction (APR) tools are available for voluntary correction of defective programs. Our study of the code generated by the code shows that: (1) Codex-generated programs show the same errors as human programmers;
(2) existing APR tools (TBar and Recoder) do not perform well at fixing bugs in auto-generated programs The fact that error (3) provides correct information, such as information from the Error field, indicates that Codex editing mode (Codex-e) is useful in code editing and its performance is better than TBar and Recorder. Our research led us to the following insights: We propose to enhance language models with software engineering artifacts, such as native bugs, to generate better code. In automatic correction (APR), inspired by the language model, we introduce scientific guidelines such as(i) extracting patch ingredients from automatically generated solution set of Codex, and (ii) making fault localization (fix localization) in program repair more flexible (iii) shifting focus from adding more fix patterns to semantic program repair approaches to improve trustworthiness of auto-generated code.The software application has been successfully calculated and passed the "test case" testing. It is user-friendly and has the necessary options through which the user can perform the necessary actions.

REFERENCES

Books:-

Cay S. Horstmann and Gary Cornell, “Core Java™” Volume I – Fundamentals 7thEd.
Dave Thau, “The Book of JavaScript: A Practical Guide to Interactive Web Pages”,Ed. 2nd .

Website:-
www.google.com
http://source.android.com/source/report-bugs.html
http://bugs.adobe.com/flashplayer/
http://en.wikipedia.org/wiki/Bug Tracking System
http://office.microsoft.com/en-in/access-help/create-an-access-database-HP005187442.aspx
http://www.easywayserver.com/blog/java-best-database-connectivity-web/