Evaluating Automated Program Repair Techniques using Introductory Programming Course Datasets

Tsukasa Nakamura*, Masanari Kondo*, Yasutaka Kamei*, Naoyasu Ubayashi*
*Kyushu University, Japan
Email: t.nakamura@posl.ait.kyushu-u.ac.jp, {kondo, kamei, ubayashi}@ait.kyushu-u.ac.jp

***Abstract*—Debugging erroneous programs requires a great deal of human effort. To reduce human effort, automating debugging processes has been actively studied so far. One of such automation is automated program repair techniques for syntactic errors in programs. Researchers intend to support novice programmers such as students with these techniques because fixing syntactic errors is a difficult task for novice programmers. However, there exist few datasets that consist of programs written by novice programmers in universities and can be used to evaluate these techniques. Also, it is difficult to prepare such datasets from scratch. Indeed, prior studies usually utilized the Indian Institute of Technology Kanpur (IITK) dataset only. This limitation restricts the findings and implications in prior studies as a case study in the university. In this study, we intend to clarify which findings and implications in prior studies remain the same and which ones change in another university by a case study. We prepare three datasets that consist of over 21k programs collected from an introductory programming course in different divisions at our university. We compare the state-of-the-art automated program repair techniques, DeepFix, RLAssist, and DrRepair, in these datasets. We found that (1) the best technique remains the same in all the datasets, (2) these techniques fix 8.3 % to 54.5 % syntactic errors in our datasets, which are 6.7 % to 32.4 % lower than those in the IITK dataset, (3) the error types that are fixed by them change in different datasets. Hence, the main finding in the IITK dataset remains the same; however, each technique fixes different errors.**

***Index Terms*—automated program repair, syntactic error**

I. Introduction

Debugging, which is the process of detecting and resolving errors, is one of the most costly tasks in software development. Indeed, the cost of debugging accounts for as much as 50% of the total development cost [4], [12]. A prior study argued that the most impactful factor for the cost is the manual effort of programmers. Hence, automating debugging is an important consideration for programmers to reduce the cost.

Automated program repair is widely studied in our community of software engineering to reduce the cost of debugging and improve the productivity of programming. A variety of automated program repair techniques has been proposed so far [8], [13]. For example, Gupta et al. [8] proposed DeepFix, which uses a seq2seq model to fix syntactic errors in C programs. DeepFix was evaluated on the programs written by students that were collected from the Indian Institute of Technology Kanpur (IITK dataset). Gupta et al. reported that DeepFix can perfectly fix 27% of erroneous programs and partially fix 19% of them.

The automated program repair techniques, which fix syntactic errors, have been frequently evaluated using the IITK dataset [1], [9]. The IITK dataset consists of programs written by students in an introductory programming course. Hence, the performance of automated program repair on this dataset implies the applicability of automated program repair for novice programmers. For example, Ahmed et al. [1] presented an automated program repair technique to help novice programmers and evaluated the technique in the IITK dataset.

However, there are no other datasets that consist of programs written by novice programmers in universities and are widely used to evaluate automated program repair. This limitation restricts the findings and implications in prior studies as a case study in the university. Hence, it is unclear whether the findings and implications are applicable to other universities.

In this study, we compare the performance of three well-known automated program repair techniques (i.e., DeepFix [8], RLAssist [7], and DrRepair [13]) in not only the IITK dataset but also other datasets. Specifically, we prepare three datasets from the programs in an introductory programming course for novice programmers in different divisions at our university (hereafter, "University X"). We intend to unveil which findings and implications in prior studies remain the same and which ones change in another university by a case study in University X. More specifically, we investigate the following research questions.

(RQ1) To what extent does the state-of-the-art automated program repair fix errors in the datasets from University X?

(RQ2) Does the state-of-the-art automated program repair fix different errors across different datasets?

We found that automated program repair techniques result in different performance between the IITK dataset and our datasets. Specifically, the techniques result in 6.7 % to 32.4 % lower performance in our datasets than those in the IITK dataset and the error types that are fixed by them change. However, the best technique remains the same in all datasets.

Our result implies that the details of the findings (i.e., the error types that are fixed) on a dataset may change on another dataset. The takeaway message of this paper is that researchers and users who want to use automated program repair techniques to fix errors in introductory programming courses in universities can use the best technique in prior studies. However, they should recognize that each technique fixes different errors.

```
1      #include <stdio.h>
2      int main() {
3          int a[1000];
4          int i, j, n, k, m = 0;
5          scanf("%d", &k);
6          scanf("%d", &n);
7          for (i = 0; i < 1000; i++) {
8              scanf("%d", &a[i]);
9          }
10         for (i = 0; i < 1000; i++) {
11             j = k - a[i];
12             while (m < 1000) {
13                 if (a[m] == j) {
14                     printf("lucky");
15                     break;
16                 }
17 -               m++;
17 +               m++; }
18         }
19         return 0;
20     }
```

Fig. 1. An example program retrieved from the IITK dataset [8]. Note that we modified the appearance (e.g., indents) for readability.

```
#include <stdio.h>
int main() {
    int i, j, x;
    scanf("%d", &x);
    if (1 <= x <= 10) {
        for (i = 1; i <= x; i++) {
            for (j = 1; j <= x; j++) {
                printf("*");
            }
        printf("\n");
        }
    if (10 < x) {
        printf("Error");
    }
    return 0;
}
```

Fig. 2. An example program in a dataset collected from the course at University X. Note that we modified the appearance (e.g., indents) for readability.

The organization of this paper is as follows. Section II explains a motivating example. Section III introduces the studied APR. Section IV describes research questions. Section V presents the studied datasets and our preliminary analysis. Section VI shows the experimental setup and results. Section VII lists related work. Section VIII describes the threats to the validity. Section IX presents the conclusion and prospects.

II. Motivating Example

In this section, we show why the evaluation in multiple datasets is important for automated program repair. We introduce two cases in which we attempted to fix an erroneous program with DeepFix.

The first example is shown in Fig. 1. This is one of the programs in the IITK dataset, which has a syntactic error due to a missing closing parenthesis. We applied DeepFix to this program, and DeepFix inserted a closing parenthesis at line 17. This insertion correctly fixed this erroneous program.

The second example is shown in Fig. 2. This program is included in a dataset collected from a lecture at University X, and this program includes a similar error as Fig. 1, which is a missing closing parenthesis. However, DeepFix cannot fix this erroneous program while it can fix a similar error in a program in the IITK dataset.

These two cases imply that even if an automated program repair technique successfully fixes an error in a dataset, such a technique may not fix similar errors in other datasets. Hence, we compare the performance of automated program repair techniques in different datasets to unveil which findings and implications in prior studies remain the same and which ones change in another university.

III. Studied Automated Program Repair

In this study, we compared the performance of three well-known automated program repair techniques as follows:

DeepFix [8]: This technique consists of a seq2seq model and fixes syntactic errors in C instead of semantic errors. DeepFix trains its deep learning model based on syntactically correct C programs. Specifically, DeepFix uses mutation to convert the correct C programs into the incorrect C programs and prepare pairs of syntactically correct and incorrect programs as the training data.

RLAssist [7]: This technique also fixes syntactic errors in C programs. RLAssist uses reinforcement learning. The authors argued that this learning process makes fixing errors more flexible than DeepFix.

DrRepair [13]: This technique also fixes syntactic errors in C programs. As described above, DeepFix uses the pairs of syntactically correct programs and incorrect programs as the training data. DrRepair uses not only such pairs but also error messages from the compiler.

These techniques have the following characteristics.

- Fix syntactic errors in C programs
- State-of-the-art techniques[1]
- All techniques were evaluated in the IITK dataset

IV. Research Questions

DeepFix [8] and RLAssist [7] are evaluated in only the IITK dataset. DrRepair [13] is evaluated in the IITK dataset and another dataset called the SPoC dataset. However, the SPoC dataset consists of programs collected from `codeforces.com` instead of programs written by novice programmers in universities. Therefore, it is unclear whether they can achieve the same results or different results for datasets that consist of programs written by such programmers. To clarify this, we investigate the following RQs.

[1]The reason for this decision is that the prior study [13] that proposed the latest technique, DrRepair, compared its technique with the rest two techniques.

TABLE I
THE STATISTICS OF THE DATASETS

Dataset	Task	Erroneous	Correct	Total
IITK	93	6,978	46,500	53,478
ED	47	3,157	7,339	10,496
PS	57	1,551	3,834	5,385
LA	21	1,861	3,936	5,797

TABLE II
SUMMARY OF ERRORS AND THEIR COMMON CAUSES

ID	Error messages	Common causes
e_1	expected declaration or statement	missing closing parenthesis
e_2	expected identifier	duplicating closing parenthesis
e_3	undeclared	undeclared variables and functions
e_4	expected misc. tokens	missing misc. tokens such as semicolons

(RQ1) To what extent does the state-of-the-art automated program repair fix errors in the datasets from University X?

(RQ2) Does the state-of-the-art automated program repair fix different errors across different datasets?

V. DATASET

A. Dataset Collection

In this study, we prepared three new datasets: ED, PS, and LA collected from the course for undergraduate students at University X. In the course, students study some basic programming tasks for C programmers. The datasets consist of answers by students for these tasks. The numbers of tasks, erroneous programs, and correct programs in the datasets are shown in Table I. The difference between the datasets is the faculty of the majority of students. The ED, PS, and LA datasets mainly consist of programs written by students who belong to the faculty of Education (ED), Pharmacy (PS), and Law (LA), respectively. Note that some programs were written by students who belong to other faculties. However, such students account for less than 10% of all students.

To prepare the datasets, we collect the logs when compiling C programs with SFTP (SSH File Transfer Protocol) from the server with the same approach as the prior study [3]. This is because University X provides students with the server to compile their C programs.

We describe the details of the course at University X in which we collect programs. In this course, students attend some lectures and write a C program for each lecture. Each lecture has a topic of programmings such as mathematical functions and conditional statements. At the end of the lecture, some tasks are given to students based on the topic. The datasets described above collected these tasks, programs written by students to these tasks, and the outputs of the compiler. One of the programs with a statement change, which fixes a syntactic error is shown in Fig. 3 as an example. This is the answer to the task, which has the following question: "Given two integers n and m from the standard input, write a program to calculate the quotient and remainder of m divided by n." When we compiled this program with GCC, the following error occurred.

```
example.c: In function 'main':
example.c:6:23: error: invalid operands to binary &
(have 'int *' and 'int')
```

This error can be solved by inserting a comma in line 5; however, the error message prompts the student to correct the usage of the "&" operator. Therefore, it might be difficult for the student who is a novice programmer to fix this error. Such syntactic errors are often found in the datasets.

```
1      #include <stdio.h>
2      int main(int argc, const char *argv[]) {
3          int n, m, sum, dif;
4          printf("Enter two numbers\n");
5 -        scanf("%d %d", &n &m);
5 +        scanf("%d %d", &n, &m);
6          sum = m / n;
7          dif = m % n;
8          printf("%d/%d=%d\n", sum);
9          printf("%d%%d=%d\n", dif);
10         return 0;
11     }
```

Fig. 3. An example of a program submitted by a student at University X. Note that we modified the appearance (e.g., indents) for readability.

B. Preliminary Analysis and Results

Motivation: The IITK dataset [8] and our datasets were collected from the introductory programming courses in the universities. Such a domain similarity may make the collected programs similar to each other. If differences exist between the datasets, such differences may cause different findings and implications. Hence, in this preliminary analysis, we clarify the differences between the IITK dataset and our datasets.

Approach: We compare the IITK dataset and our datasets from three perspectives: the numbers of erroneous and correct programs, the distribution of error messages, and the source code metrics. The numbers of erroneous and correct programs and the distribution of error messages are important to train the model of automated program repair. The former corresponds to the size of the training data; the latter corresponds to the types of syntactic errors. The source code metrics are used to compare the complexity of programs.

As the error messages, we used the messages that were extracted by Gupta et al. [8]. Gupta et al. summarized the most common four error messages and their causes in the IITK dataset. These error messages and their causes are shown in Table II. As the source code metrics, we used the LOC and the cyclomatic complexity.

Results: The IITK dataset and our datasets have differences in all perspectives. The numbers of erroneous and

TABLE III
NUMBER OF ERRORS FOR EACH DATASET

Dataset	e_1	e_2	e_3	e_4	Others	Total
IITK	507 (3.1 %)	1,659 (10.0 %)	5,468 (33.0 %)	6,790 (41.0 %)	2,132 (12.9 %)	1,6556
ED	242 (2.8 %)	485 (5.6 %)	887 (10.3 %)	2,616 (30.4 %)	4,384 (50.9 %)	8,614
PS	111 (2.4 %)	176 (3.8 %)	784 (16.7 %)	1,227 (26.2 %)	2,386 (50.9 %)	4,684
LA	182 (4.6 %)	257 (6.6 %)	937 (23.9 %)	1,139 (29.0 %)	1,408 (35.9 %)	3,923

TABLE IV
THE SOURCE CODE METRICS OF THE PROGRAMS FOR EACH DATASET

Dataset	Average of CountLine	Average of AvgCyclomatic
IITK	26.7	4.73
ED	16.2	2.31
PS	16.1	2.47
LA	16.8	2.67

correct programs are shown in Table I. We observed that the number of programs in the IITK dataset is 5.1 to 9.9 times larger than those in our datasets. On the contrary, the proportion of erroneous programs in the IITK dataset is 2.2 to 2.5 times smaller than those in our datasets.

We summarized the numbers of each error message of our studied datasets in Table III. Since e_1 to e_4 were the IDs of the errors that appeared most frequently in the IITK dataset, most of the errors found in the IITK dataset belong to one of these categories. On the contrary, we observed that the IITK dataset and our datasets have differences. Specifically, 35.9% to 50.9% errors in the ED, PS, and LA datasets do not belong to these four categories. Indeed, the proportions in the “Others” column of our three datasets are larger than that of the IITK dataset. An example of the “Other” columns is “conflicting types for X.” This error message indicates that, for example, a function that is called is not defined.

Table IV shows two source code metrics that are the LOC and the cyclomatic complexity for each dataset. “CountLine” indicates the LOC; “AvgCyclomatic” indicates the average of the cyclomatic complexities of the functions in each program. We measured these source code metrics with Understand[2]. Similar to the distributions of programs for each error message, we observed that the IITK dataset and our datasets have differences in terms of the source code metrics. The IITK dataset includes more complex programs than our datasets.

Summary of Preliminary Analysis

The size and the complexity of programs in our datasets are smaller than those in the IITK dataset. However, the proportion of erroneous programs in our dataset is larger than that in the IITK dataset. In addition, the distribution of the errors is different.

[2]https://www.scitools.com/

TABLE V
THE NUMBER OF ERRORS THAT ARE FIXED BY EACH AUTOMATED PROGRAM REPAIR TECHNIQUE IN OUR DATASETS

APR	ED	PS	LA
DeepFix	188/1,472 (12.8 %)	101/1,217 (8.3 %)	109/1,081 (10.1 %)
RLAssist	245/1,457 (16.8 %)	97/779 (12.5 %)	196/984 (19.9 %)
DrRepair	1,698/3,113 (54.5 %)	794/1,529 (51.9 %)	425/1,249 (34.0 %)

TABLE VI
THE NUMBER OF ERRORS THAT ARE FIXED BY EACH AUTOMATED PROGRAM REPAIR TECHNIQUE IN THE IITK DATASET RETRIEVED FROM THE ORIGINAL PAPERS, RESPECTIVELY

APR	IITK
DeepFix [8]	1,881/6,971 (27.0 %)
RLAssist [7]	1,854/6,975 (26.6 %)
DrRepair [13]	-/6,971 (66.4 %)

VI. EXPERIMENTS AND RESULTS

In our experiments, we applied DeepFix, RLAssist, and DrRepair to the ED, PS, and LA datasets. We evaluated the automated program repair techniques in terms of the number of erroneous programs that are correctly fixed. We aim at clarifying whether the conclusion in the prior study [13] remains, which reported that DrRepair is the best automated program repair technique.

A. *RQ1: To what extent does the state-of-the-art automated program repair fix errors in the datasets from University X?*

In this RQ, we studied whether DeepFix, RLAssist, and DrRepair fix errors in the ED, PS, and LA datasets. Fig. 4 shows the procedure of this experiment. We applied the automated program repair techniques to our datasets and obtained the

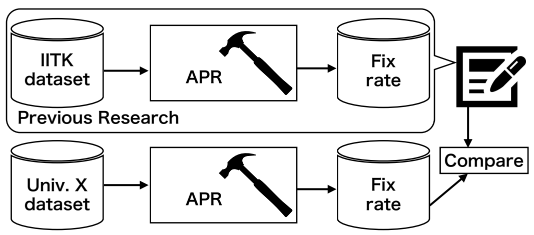

Fig. 4. The procedure of the experiment for RQ1

number of errors that are fixed. On the other hand, we obtained the number of errors that are fixed in the IITK dataset from the prior papers that proposed each technique. We compared the number of errors that are fixed between our datasets and the IITK dataset.

It should be noted that to replicate the prior studies [7], [8], [13] as much as possible, we used the same experimental setting for each automated program repair technique following the prior study that proposed the technique, respectively. Hence, each automated program repair technique has different experimental settings. For example, DeepFix can be applied to programs that have less than 450 tokens.

On the contrary, we modified the training process for the techniques to study the impact of differences of the datasets. DeepFix and RLAssist were not enough trained by the default number of epochs in our datasets. Hence, we set the number of epochs 100 for both. DrRepair has the feature to be pre-trained with the Codeforce dataset. However, we did not use this feature because we intend to study the impact of differences of the datasets only.

All techniques show lower proportions of errors that are fixed in the ED, PS, and LA datasets than that in the IITK dataset. Table V shows the number of errors that are fixed in our datasets while Table VI shows those in the IITK dataset retrieved from three prior studies [7], [8], [13]. DeepFix and RLAssist fix less than 20% of errors in our datasets while they fix over 20% of errors in the IITK dataset. Similarly, DrRepair fixes less than 60% of errors in our dataset while that fixes over 60% of errors in the IITK dataset. Specifically, the performance in our datasets is 6.7% to 32.4% lower than those in the IITK dataset. However, the proportions of errors that are fixed in our datasets are not extremely low.

The conclusion, which reports DrRepair fixes the highest number of errors, remains the same. Even in our datasets, DrRepair fixes the highest number of errors compared to DeepFix and RLAssist. Hence, the main conclusion in the prior study [13] remains the same.

Answer to RQ1

All techniques show lower proportions of errors that are fixed in the ED, PS, and LA datasets than that in the IITK dataset. However, these proportions are not extremely low. In addition, the conclusion in the prior study remains the same in our datasets.

B. *RQ2: Does the state-of-the-art automated program repair fix different errors across different datasets?*

Since there exist few datasets, prior studies did not investigate if the difference of syntactic errors that are fixed by an automated program repair technique in different datasets exists. Hence, in this RQ, we investigate whether one technique fixes different syntactic errors in different datasets.

The procedure of this RQ is shown in Fig. 5. For each dataset, we extracted the errors that were fixed for each technique only and decided the most frequently fixed errors from the extracted errors for each technique (*typical errors*).

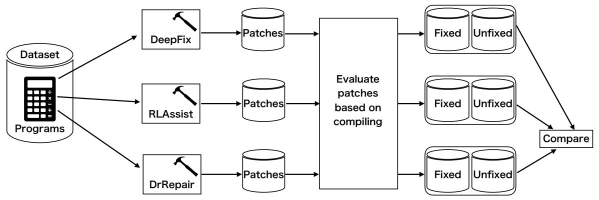

Fig. 5. The procedure of the experiment for RQ2

TABLE VII
TYPICAL ERRORS IN THE ERRONEOUS PROGRAMS THAT ONLY ONE AUTOMATED PROGRAM REPAIR TECHNIQUE COULD FIX

APR	ED	PS	LA
DeepFix	e_4 (20/8,871)	e_3 (19/2,274)	e_3 (21/2,478)
RLAssist	e_4 (31/8,871)	e_4 (26/2,274)	e_4 (123/2,478)
DrRepair	e_4 (2,800/8,871)	e_3 (573/2,274)	e_3 (661/2,478)

The typical errors change in different datasets. Table VII shows the typical errors for each technique for each dataset. Each cell shows the id of the typical error (Table II). The parentheses show the number of typical errors and all fixed errors for each dataset. We observed that the typical errors change in different datasets between e_3 and e_4.

Hence, using only one dataset to evaluate the automated program repair techniques may overlook insights. For example, RQ1 shows that DrRepair outperforms the other two techniques, whereas Table VII implies that RLAssist augments the performance of DrRepair to fix e_4 in the PS and LA datasets compared to using only DrRepair. Especially, in the LA dataset, RLAssist fixes more than 100 e_4 errors. If we only used the ED dataset, we would not conclude that RLAssist has the potential to fix more e_4 errors combined with DrRepair compared to DeepFix.

Answer to RQ2

The errors that are fixed by the state-of-the-art automated program repair techniques change in different datasets. Hence, even if the main conclusion remains the same, it is important to evaluate automated program repair techniques in different datasets.

VII. RELATED WORK

Automated program repair has been an active research field, and various techniques have been proposed so far. In this section, we describe prior studies for automated program repair techniques.

Gupta et al. [8] assumed that the most common errors are syntactic errors caused by developers' carelessness and lack of knowledge. Hence, they proposed DeepFix, and reported that DeepFix correctly fixed 27% of 6,971 erroneous programs and partially fixed 19% of them. Gupta et al. [7] also proposed RLAssist, and reported that RLAssist fixes more erroneous programs than DeepFix. Yasunaga et al. [13] proposed DrRepair, and reported that DrRepair fixed more programs than both DeepFix and RLAssist.

On the other hand, unlike these techniques, prior studies also proposed automated program repair techniques for semantic errors. Gulwani et al. [6] proposed CLARA, which fixes semantic errors. They evaluated the performance of CLARA on a dataset collected from a MOOC by MITx. They found that CLARA can fix 4,183 out of 4,293 incorrect programs (97.44%). Tufano et al. [11] investigated the applicability of Neural Machine Translation (NMT) for learning the patterns to fix erroneous programs. They empirically evaluated their NMT model using the patterns in Java programs in GitHub Archive. They found that the NMT model fixed 9-50% of 2,927 small programs and 3-28% of 1,869 medium programs.

Similar to our study, prior studies also built or used new datasets. For example, Goues et al. [5] built two datasets called ManyBugs and IntroClass. ManyBugs consists of the programs of OSS while IntroClass consists of the programs written by students. Yi et al. [14] collected programs written by students from an introductory C language course in the Indian Institute of Technology Kanpur as a new dataset, which is not the IITK dataset that we used in this paper. Kaleeswaran et al. [10] evaluated their tool, CoderAssist, using C programs written by students from the submissions on CodeChef. Birch et al. [2] used a dataset of Java and Python programs written by students to answer Computer Science coursework questions at the University of Auckland. However, the target error of these datasets is semantic errors. Hence, the lack of datasets for syntactic errors is still a challenge.

VIII. Threats to Validity

Internal validity. As described in Section V-A, we collected the logs when compiling programs as our datasets from the server at University X. In this server, students can compile their programs multiple times without any changes. Hence, our datasets may duplicate the same programs. If such erroneous programs exist and are easily/hardly to be fixed by the studied automated program repair techniques, our results might be biased by these duplicated programs.

External validity. In this study, we compared DeepFix, RLAssist, and DrRepair in our datasets and confirmed that the main conclusion is not changed in prior studies. However, our datasets are collected from only University X. Future studies are necessary to extend the datasets more.

IX. Conclusion

The IITK dataset is the only dataset that consists of programs written by students in an introductory programming course and are widely used to evaluate automated program repair techniques, though it is important to propose techniques for such students. In this study, we unveiled which findings and implications in prior studies remain the same and which ones change in another university as a case study. Specifically, we built the ED, PS, and LA datasets from the introductory programming course at University X and evaluated DeepFix, RLAssist, and DrRepair. Our results show that the main conclusion in the prior study [13] remains the same; however, the types of errors that are fixed by automated program repair techniques change.

We currently extend our study to different programming languages. Specifically, we collect programs written by Python from the introductory programming course at University X.

Acknowledgments

This research was partially supported by JSPS KAKENHI Japan (Grant Numbers: JP18H04097, 21H04877) and JSPS International Joint Research Program with SNSF (Project “SENSOR”: JPJSJRP20191502).

References

[1] U. Z. Ahmed, P. Kumar, A. Karkare, P. Kar, and S. Gulwani, “Compilation error repair: for the student programs, from the student programs,” in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 78–87.

[2] G. Birch, B. Fischer, and M. Poppleton, “Using fast model-based fault localisation to aid students in self-guided program repair and to improve assessment,” in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2016, Arequipa, Peru, July 9-13, 2016*, A. Clear, E. Cuadros-Vargas, J. Carter, and Y. Túpac, Eds., 2016, pp. 168–173.

[3] X. Fu, A. Shimada, H. Ogata, Y. Taniguchi, and D. Suehiro, “Real-time learning analytics for C programming language courses,” in *Proceedings of the 7th International Learning Analytics & Knowledge Conference*, 2017, pp. 280–288.

[4] L. Gazzola, D. Micucci, and L. Mariani, “Automatic software repair: A survey,” *IEEE Transactions on Software Engineering*, vol. Vol. 45, no. 1, pp. 34–67, 2019.

[5] C. L. Goues, N. J. Holtschulte, E. K. Smith, Y. Brun, P. T. Devanbu, S. Forrest, and W. Weimer, “The manybugs and introclass benchmarks for automated repair of C programs,” *IEEE Trans. Software Eng.*, vol. 41, no. 12, pp. 1236–1256, 2015.

[6] S. Gulwani, I. Radicek, and F. Zuleger, “Automated clustering and program repair for introductory programming assignments,” in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, 2018, pp. 465–480.

[7] R. Gupta, A. Kanade, and S. K. Shevade, “Deep reinforcement learning for syntactic error repair in student programs,” in *The 33rd AAAI Conference on Artificial Intelligence*, 2019, pp. 930–937.

[8] R. Gupta, S. Pal, A. Kanade, and S. K. Shevade, “Deepfix: Fixing common C language errors by deep learning,” in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017, pp. 1345–1351.

[9] H. Hajipour, A. Bhattacharyya, and M. Fritz, “Samplefix: Learning to correct programs by sampling diverse fixes,” *CoRR*, vol. abs/1906.10502, 2019.

[10] S. Kaleeswaran, A. Santhiar, A. Kanade, and S. Gulwani, “Semi-supervised verified feedback generation,” in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, 2016, pp. 739–750.

[11] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, “An empirical study on learning bug-fixing patches in the wild via neural machine translation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, pp. 19:1–19:29, 2019.

[12] Undo, “What is Reverse Debugging, and why do we need it?” https://undo.io/resources/reverse-debugging-whitepaper/, accesse Date: 2021-12-18.

[13] M. Yasunaga and P. Liang, “Graph-based, self-supervised program repair from diagnostic feedback,” in *Proceedings of the 37th International Conference on Machine Learning*, 2020, pp. 10 799–10 808.

[14] J. Yi, U. Z. Ahmed, A. Karkare, S. H. Tan, and A. Roychoudhury, “A feasibility study of using automated program repair for introductory programming assignments,” in *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 740–751.