REFERENT: Transformer-based Feedback Generation using Assignment Information for Programming Course

Jinseok Heo
Department of Electrical and Computer Engineering, Sungkyunkwan University
Suwon, Korea
mrhjs225@skku.edu

Hohyeon Jeong
Department of Electrical and Computer Engineering, Sungkyunkwan University
Suwon, Korea
jeonghh89@skku.edu

Dongwook Choi
Department of Computer Science and Engineering, Sungkyunkwan University
Suwon, Korea
dwchoi95@skku.edu

Eunseok Lee
College of Computing and Informatics, Sungkyunkwan University
Suwon, Korea
leees@skku.edu

***Abstract*—Students require feedback on programming assignments to improve their programming skills. An Automated feedback generation (AFG) technique proposes to provide feedback-corrected submissions for incorrect student programming submissions in programming courses. However, these techniques are limited as they rely on the availability of correct submissions as a reference to generate feedback. In situations where correct submissions are not available, they resort to using mutation operators, which can lead to a search space explosion problem. In this work, we propose *REFERENT*, Transformer-based feedback generation using assignment information. *REFERENT* uses transfer learning on a pre-trained model with data from students' submission history from the past assignment. To generate assignment-related feedback, we use a title, tag, assignment description, and test case as assignment information. *REFERENT* can generate feedback without a reference program in limited resources. We conducted a preliminary study to confirm the effectiveness of *REFERENT* and the feasibility of using assignment information. *REFERENT* generated feedback for 32.7% of incorrect submissions without reference programs and that its performance increased up to 50.7% when reference programs were used. We also check whether the submission history, assignment information, and repair knowledge of open-source software help generate feedback.**

***Index Terms*—Programming Assignment, Automated Feedback Generation, Transformer, Transfer Learning, Assignment Information**

I. Introduction

The growth of online learning platforms such as MOOCs (Massive Open Online Courses) has increased the number of online programming courses [1]. With this system, programming classes have become more accessible. The instructor helps students by providing feedback on semantic errors in students' incorrect submissions. Through this feedback, it is possible to provide students with various types of information, such as scores and hints, as well as solutions [2]. However, as the number of students taking programming courses increases, manually providing feedback to each student takes more time. The previous research has indicated that not helping students is detrimental to student learning and leads to higher dropout rates in computer science [3]. AFG has been proposed to mitigate this problem [1], [2], [5]–[8]. AFG automatically generates feedback for a given semantic error of incorrect submission in a programming course. Submissions of students are divided into incorrect submissions and correct submissions depending on test cases. AFG repairs students' incorrect submissions and generates a repaired submission that passes the test cases and provides it as feedback.

Existing techniques refer to the answer programs by the instructor or correct submissions to repair the incorrect submission [2], [5], [7]–[9]. Students solve assignments with various algorithms. It is challenging to create a repaired submission if there are no reference projects with similar structures. Existing techniques have attempted to increase the number and diversity of reference programs to mitigate this problem. For example, Yang Hu et al. [2] used the refactoring rule to increase the number of reference programs with diverse control flow structures. By applying the refactoring rule to the code in the correct submissions, it generated code that was the same semantic but different syntactic code. Therefore, they could increase the success rate of matching between incorrect submissions and reference programs.

However, in the case of a newly released assignment, the available correct submissions may not be available. Further, some programming assignments, such as online coding competitions, may not have an answer program by an instructor. In these cases, generating feedback using existing techniques takes much work. To overcome this problem, Leping Li et al. [8] generated repaired submissions based on the mutation-based repair. They generated feedback by applying a mutation operator, which was to remedy the weakness of reference-based repair.

Existing reference-based AFG only uses submission of the same assignment. However, the student submission history of other assignments contains various code snippets and assignment information. This will help in generating repaired submissions for incorrect submissions. The AFG technique, which does not use a reference program, generates a repaired submission through a mutation-based repair. However, com-

binations of different mutation operators may cause a search space explosion problem [10]. Given the student's learning efficiency, providing feedback should be done quickly, within seconds to minutes [11], [12]. Therefore, generating feedback with the mutation-based technique is quite a challenge in a limited resource environment (time, computing resources).

Recently, deep learning (DL) has shown good performance in code generation and patch generation [13]–[15]. Among them, the transformer-based technique is one of the cutting-edge techniques. A program repair technique [16] trains the pre-trained model using patch history through transfer learning. We can use these concepts to repair incorrect submissions using assignment information and submission history in many assignments.

In this paper, we propose *REFERENT*, transformer-based feedback generation. The key insight is that by learning students' patch history for other assignments, feedback can be generated for a wide variety of incorrect submissions without reference programs. *REFERENT* leverages transfer learning to use the open-source software's repair knowledge to generate more feedback. *REFERENT* pre-trains the transformer-based model by collecting the patch history in the same programming language as submitted in the open-source software. For transfer learning, we collect pairs of incorrect submissions and correct submissions submitted by the same user from the submission history of assignments and treat each pair as an instance. We train the pre-trained model with pairs of submissions. We additionally use assignment information to guide assignment-related codes during training. *REFERENT* generates repaired submissions for incorrect submissions with the trained model.

For a preliminary study of the *REFERENT* performance and feasibility, we collected 300 submissions in 10 assignments according to the data collection method in a previous study [8]. *REFERENT* was able to generate feedback for 32.7% of incorrect submissions without reference programs. We found that students' submission history, assignment information, and repair knowledge of open-source software had a positive effect on the generation of feedback.

The contribution of this paper is as follows

- This is the first transformer-based automatic feedback generation technique that uses assignment information that can be used without reference programs.
- We analyze the effect of different types of assignment information on feedback generation.
- We analyze the impact of assignment information, submission history, and open-source software repair knowledge on feedback generation.

II. Related work

A. Automated Feedback Generation

Most AFG techniques provide feedback in the form of a repaired program. The feedback generated by the AFG technique should have a small edit distance for student understanding [4]. It should deliver within seconds to minutes for high learning efficiency [11], [12]. Unlike automated program repair techniques [14], [16], [17], AFG techniques can use a program that has passed a test case (such as the instructor's answer program or a student's correct submission) as a reference. In incorrect submission, these reference programs using to change the control flow structure or block of source code. SARFGEN [4] computes the Euclidean distance between incorrect submissions and reference programs to generate feedback based on the nearest reference program. With the calculated distance, SARFGEN matches statements between incorrect submission and reference programs based on the order of the distance. It generates feedback by repairing an incorrect submission with minimal repairs to be easy to understand for students. Refactory [2] attempts to address the lack of correct submissions. It increases the number of correct submissions by applying refactoring rules to an existing correct submission or answer program. Refactory generates feedback by finding reference programs that have the same control flow structure as the incorrect submission.

AssignmentMender [8] generates feedback using correct submissions and incorrect submissions as reference programs. If it finds a similar code snippet in the reference programs, it tries to generate feedback by applying it to the student's incorrect submission. If there is no available reference program, it attempts to repair using mutation-based repair [19], [20]. It uses operator replacement, variable name replacement, and statement deletion as mutation operators.

Since the above techniques depending on the number of reference programs, it is difficult to apply for no available reference programs like newly released assignments. In the case of mutation-based repair, it is challenging to generate repaired code in a limited resource environment due to the search space explosion problem [10]. *REFERENT* can generate feedback regardless of the existence of a reference program in a limited resource environment.

B. Transformer-based repair

TFix [16] is a transformer-based code error repair technique. It uses transfer learning through code error data based on the T5 model [21] to repair code errors detected by static analysis tool [22]. The T5 model was pre-trained for the numerous Natural Language Processing(NLP) task. TFix can generate a patch by inputting the error code in the generated model. *REFERENT* uses transfer learning to leverage open-source software and the students' code repair knowledge. *REFERENT* also uses assignment information data to generate assignment-related feedback.

III. REFERENT

A. Generating Pre-trained Model

Figure 1 shows an overview of *REFERENT*. We build a pre-trained model to generate feedback using general repair knowledge available in open-source software We collect projects from open-source repositories with the same programming language as the submissions. In the project, we collect commits whose commit message includes patch-related keywords

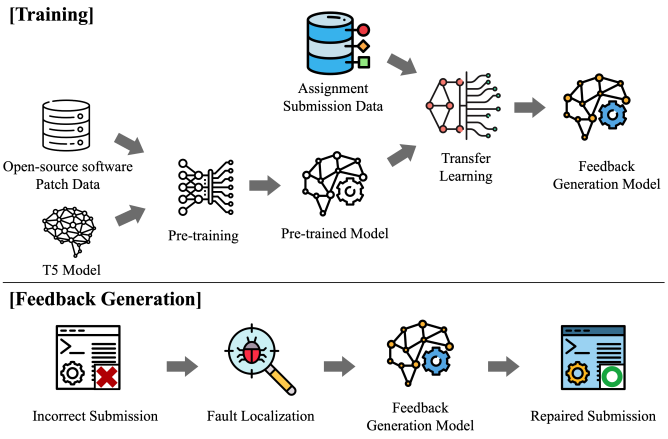

Fig. 1: Overview of REFERENT

(patch, repair, fix) as patches. This search method is popularly used in the program repair domain [24], [25].

REFERENT requires a model that can learn both the natural language (expression of assignment information) and source code (expression of submission). Recently, in the automated program repair domain, a previous study attempted to generate a patch by inputting the source code as natural language into the transformer model and obtained good performance. [13], [14], [16]. We use this concept to build pre-trained models. The T5 model was trained on large amounts of data for the NLP task ans has proven performance as a transformer-based state-of-the-art model [26]. We train the collected patch history in T5 model to build a pre-trained model.

B. Assignment Data Construction

We collect assignment information and students' submission history to apply transfer learning. As assignment information, it collects a title, tag, assignment description, and example test cases. Programming assignments in the MOOC generally contain this information. These are also provided in most assignments to explain the assignments.

It is necessary to train about the task of repairing the code to generate feedback. Therefore, we collect incorrect-correct submission pairs to train how the submissions should be changed. We collect submission pairs by selecting each correct submission from the submission history and the incorrect submission submitted just before the correct submission. We use the collected pair and assignment information for transfer learning.

C. Transfer Learning

We transform incorrect submission and assignment information into the following form for input into the transformer model.

Fix; Assignment Title, Tag, Assignment Description, Example test cases; Faulty line, Incorrect submission code;

In *REFERENT*, the first word in the input is the task type, called *Fix*. Previous research [16] has also used a task term to design a T5 model for a specific task. The following information is for assignment information and includes the title, tags, assignment description, and test case. A trailing statement is a code for an incorrect submission, with the faulty line being the specific statement that is incorrect. *REFERENT* considers the differences between incorrect and correct submissions and uses the entire incorrect code as context when generating a patch to fix the faulty line.

The output text of *REFERENT* is the same as the input text, except for the faulty line and incorrect submission code. We train *REFERENT* to modify the faulty line and incorrect submission code to repaired line and correct submission code.

D. Feedback Generation

To repair incorrect submissions, *REFERENT* performs fault localization to find a faulty line. It uses assignment information, identified faulty lines, and an incorrect submission as input to the feedback generation model. The feedback generation model generates a repaired submission and provides it to students as feedback.

IV. Preliminary Study

A. Model Construction

We used TFix [16] as a pre-trained model to implement *REFERENT*. TFix is a model that repairs code errors in programs. The submission dataset we collected was Python, which is different from the language learned by TFix. However, TFix is a state-of-the-art model of well-trained general repair knowledge that occurs in open-source software and is sufficient for the verify the concept of *REFERENT*. The TFix model we used was implemented based on T5-base. The number of model parameters for the trained TFix is approximately 220M, and it was trained on 104k instances of errors that occurred in JavaScript. The learning parameters required in training the assignment data follow the parameters used in TFix (epoch: 30, batch size: 32, learning rate: 1e-4). We used Intel Xeon-Gold 6242 CPU, 64GB RAM, NVIDIA RTX 3090 24GB 2ea for training the model.

B. Metric with Educational Scenario

When a student submits an incorrect submission, the AFG technique provides a repaired submission as feedback. The repaired submission should not only pass the test cases of the assignment but also not change the student's approach.

While it is possible to provide a completely rewritten program as feedback, this would not help students understand the assignment with their own approach. [4] Therefore, the AFG technique should generate a correct submission with minimal fixes from the incorrect submission. For this evaluation, we evaluate whether the *REFERENT* can generate a correct submission submitted by the same student immediately after the incorrect submission. The evaluation criterion is an exact match, a metric often used in existing papers [16], [28]. Exact match is a metric that determines whether strings are identical except for white space.

TABLE I: Performance of *REFERENT*

Kind of information used by model	# of available reference program 0	10	30	50
None[1]	**98**	**132**	141	149
Title	94	130	141	144
Tag	81	126	**146**	**152**
Desc.[2]	11	119	135	143
Test Cases	74	130	139	**152**
Title + Tag + Desc.	12	116	134	145
Title + Tag + Test Cases	54	125	143	150
Title + Desc. + Test Cases	13	119	131	139
Tag + Desc. + Test Cases	10	114	127	132
Title + Tag + Desc. + Test Cases	12	116	131	132
Average	45.9	122.7	136.8	143.8
Best	98	132	146	152

[1] Not using Assignment Information
[2] Assignment Description

C. Dataset

1) ***Training dataset****:* For our training data, we collected assignment information and student submissions from CodeChef [29]. CodeChef is a programming practice website with a rich history of programming assignments and student submissions, and a previous study also used CodeChef [18]. We followed the same method as described in Chapter III-B. Specifically, we obtained pairs of incorrect and correct submissions from the submission history of individual users for each assignment. All of the submissions were written in Python, the same programming language used in previous research [2], [4], [8]. In total, we collected 30,263 submission pairs for 300 different assignments to train the pre-trained model.

2) ***Test dataset****:* Our study used a submission pair as a reference program for *REFERENT*. This makes it difficult to directly compare our approach with existing techniques [2], [8] which use a single submission as a reference program. Additionally, the evaluation dataset from the previous study is not publicly available. However, we attempted to replicate the evaluation dataset using the existing technique [8] for indirect comparison. To do this, we collected submissions from CodeForce [30], an online code contest site similar to CodeChef, on 10 assignments that had been used in previous research. We only collected submissions submitted in 2020 to match the timeline of the previous study.

Keeping the above constraints, We randomly collected 80 incorrect-correct submission pairs for each assignment. We used 30 submission pairs for testing, with an incorrect submission as the test input, and correct submission as the ground truth for evaluation. The remaining 50 pairs are used as reference programs.

D. Experiment Setup

1) ***RQ1.How many feedback can REFERENT generate without a reference program?****:* We examined the performance of *REFERENT* with and without the reference programs in RQ1. To assume that there is no available reference program, *REFERENT* generates a repaired submission using only an incorrect submission for each assignment. We also analyzed the effect of the number of reference programs on performance when reference programs were available. To do this, we increased the number of reference programs used to generate feedback to 10, 30, and 50. *REFERENT* uses the reference programs as additional fine-tuning datasets.

2) ***RQ2.How does the assignment information affect the performance of REFERENT?****:* In this RQ, we checked the effect of assignment information on *REFERENT* by comparing the model's performance when using different combinations of assignment information during the transfer learning and test.

3) ***RQ3.Ablation study****:* We examine how the component used by *REFERENT* affects feedback generation. We compared performance except for the following concepts in *REFERENT*. (1) Transfer learning of open-source software repair knowledge. (2) Assignment information. (3) Student's submission history.

```
for i in str:
    if i != 'a' and i != 'e' and (i != 'i') and (i != 'o') and (i != 'u'):
```

(a) Example of incorrect submission

```
for i in range(len(str)):
    if str[i] != 'a' and str[i] != 'e' and (str[i] != 'i') and
       (str[i] != 'o') and (str[i] != 'u') and (str[i] != 'y'):
```

(b) Feedback using the correct submission of an another student

```
for i in str:
    if i != 'a' and i != 'e' and (i != 'i') and (i != 'o') and (i != 'u')
       and (i != 'y'):
```

(c) Feedback from *REFERENT*

Fig. 2: Example of feedback generated by *REFERENT*

E. Result

1) ***Result of RQ1****:* The *Best* row of Table I shows the performance of the *REFERENT*. For 300 incorrect submissions, *REFERENT* could generate a repaired submission for 98 (32.7%) incorrect submissions without using reference programs. AssignmentMender generated repaired submissions for 11.7% of the incorrect submissions without using reference programs for the CodeForce dataset collected by the authors. It is difficult to directly compare these accuracies because the exact same data is not used, but *REFERENT* has the potential to generate more repaired submissions than existing techniques. When *REFERENT* uses reference programs, the number of repaired submissions increases, and repaired submissions could be generated up to 152 (50.7%) incorrect submissions.

Figure 2 illustrates an example of the feedback generated by *REFERENT*. The red portion in Figure 2(b), (c) highlights the differences between the incorrect submission and the feedback. The feedback generated by another student's correct submission changes a lot because it uses *range(len(str))* instead of *str*. Besides that, the feedback generated by *REFERENT* shows modifications that add only one condition. Therefore, it is easy to understand where it is wrong and how to fix it, because it is minimally fixed based on the student's code.

TABLE II: Ablation Study

No.	Model	Accuracy #	Accuracy %
1	T5 + OSS[1]	144	48.0%
2	T5 + Submission History	149	49.7%
3	T5 + OSS + Submission History	149	49.7%
4	T5 + Assignment Info. + Submission History	144	48.0%
5	T5 + OSS + Assignment Info. + Submission History (*REFERENT*)	**152**	**50.7%**

[1] Open Source Software

Answer to RQ1. We found that *REFERENT* can provide feedback to students for 33% of the cases even when the instructor's answer program and reference programs are not available. If reference programs are available, *REFERENT* can provide more feedback to students.

*2) **Result of RQ2**:* Table I shows the performance according to the type of assignment information used. The first column shows the type of assignment information used by each model.

Without reference programs, the best performance was achieved by not using assignment information. Among the models that used assignment information, the model using only the Title generated feedback for 94 incorrect submissions. If reference programs are available, the average performance of each model increases significantly. The model using Tag and TC had the best performance when using reference programs. Upon analyzing the Tag information used in the test, we found that there was a common tag present even for different assignments. This likely contributed to the model's good performance, as it was able to train from similar changes that occurred in other assignments.

The overall performance of the model using assignment information could be much higher (70˜80%). The problem lies in how assignment information is used. Specifically, we simply listed the assignment information as a string and fed it into the model without any further preprocessing. Additionally, the TFix model can only handle a token length of 256, and the description exceeded this limit. To overcome this limitation, we used only the first 200 tokens, resulting in an inadequate summarization of the assignment information, leading to information loss and making it difficult for the model to accurately understand the information. This limitation can be addressed by applying techniques such as doc2vec [31] or TF-IDF to extract only sentences that can represent the assignment and provide a more accurate summarization of the information.

Answer to RQ2. Assignment information can have a positive impact on generating feedback for students. However, it should be written in consideration of information summary and loss.

*3) **Result of RQ3**:* Table II shows the ablation study results for components used by *REFERENT*. For each model, a total of 300 incorrect submissions were tested assuming 50 available reference programs for each assignment. The second column shows the components used for each model.

REFERENT versus model No.4 shows a positive effect of repair knowledge of open-source software(OSS). *REFERENT* versus model No.3 shows the assignment information also had a positive effect. However, when comparing the accuracy of the No.2 and No.4 models, the assignment information only sometimes has a good effect because the method of using the assignment information mentioned in RQ2 still needs to be developed. Second, the No.2 versus No.3 models shows the patch history of open-source software only sometimes has a significant effect. However, Since each model used 50 reference programs for fine-tuning, it can be seen that the open-source software patch history did not significantly affect assignments due to overfitting student submission history.

Answer to RQ3. The transfer learning and assignment information can have a positive effect on the feedback provided in programming courses.

V. Conclusion and Future Work

Several techniques have been proposed to generate feedback on programming assignments. The *REFERENT* is the first transformer-based automatic feedback generation technique that uses assignment information regardless of the reference programs. It uses the submission history of other assignments for the diversity of reference programs. *REFERENT* could generate feedback for 33% of incorrect submissions without reference programs. *REFERENT* with the reference program could generate feedback for half of the submissions. In addition, assignment information and repair knowledge of open-source software has a positive effect on generating feedback.

The following are future research directions that we will explore.

- **Utilization of assignment information.** Currently, *REFERENT* converts all assignment information into strings and used them as input to the model. In future research, we aim to extract abstract information and propose a suitable model for handling assignment information.
- **Variety of feedback** To the best of our knowledge, few techniques provide feedback on programming assignments beyond repaired submissions. We plan to research providing various types of feedback, such as on code efficiency and readability.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2019R1A2C2006411). This work was supported by the BK21 FOUR Project.

REFERENCES

[1] Rishabh Singh et al., "Automated Feedback Generation for Introductory Programming Assignments," Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 15–26, 2013.

[2] Yang Hu et al., "Re-factoring based program repair applied to programming assignments," 34th IEEE/ACM International Conference on Automated Software Engineering, pp. 388-398, 2019.

[3] Jamie Gorson et al., "Why do CS1 Students Think They're Bad at Programming? Investigating Self-Efficacy and Self-Assessments at Three Universities," Proceedings of the 2020 ACM Conference on International Computing Education Research, pp. 170-181, 2020.

[4] Ke Wang et al., "Search, align, and repair: data-driven feedback generation for introductory programming exercises," Proceedings of the 39th ACM SIGPLAN conference on programming language design and implementation, pp. 481-495, 2018.

[5] Sumit Gulwani et al., "Automated Clustering and Program Repair for Introductory Programming Assignments," Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 465-480, 2018.

[6] Dongwook choi et al., "Automated Feedback Generation for Multiple Function Programs," Asia-Pacific Software Engineering Conference, pp. 582-583, 2021.

[7] Dowon Song et al., "Context-aware and data-driven feedback generation for programming assignments," Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 328-340, 2021.

[8] Leping Li et al., "Generating Concise Patches for Newly Released Programming Assignments.," IEEE Transactions on Software Engineering, preprint, 2022.

[9] Shalini Kaleeswaran et al., "Semi-supervised Verified Feedback Generation," Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 739-750, 2016.

[10] Ming Wen et al., "An Empirical Analysis of the Influence of Fault Space on Search-Based Automated Program Repair," arXiv:1707.05172, 2017.

[11] M. Choy et al., "Experiences in Using an Automated System for Improving Students' Learning of Computer Programming," International Conference on Web-Based Learning, pp. 267-272, 2005.

[12] Anne Venables et al., "Programming students NEED instant feedback!," Proceedings of the 5th Australasian Computing Education Conference, pp. 267-272, 2003.

[13] Thibaud Lutellier et al., "Coconut: combining context-aware neural translation models using ensemble for program repair," Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis, pp. 101-114, 2020.

[14] Nan Jiang et al., "Cure: Code-aware neural machine translation for automatic program repair," 2021 IEEE/ACM 43rd International Conference on Software Engineering, pp. 1161-1173, 2021.

[15] Yi Li et al., "Dlfix: Context-based code transformation learning for automated program repair," Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 602-614, 2020.

[16] Berkay Berabi et al., "TFix: Learning to fix coding errors with a text-to-text transformer," International Conference on Machine Learning, pp. 780-791, 2021.

[17] Kui Liu et al., "TBar: Revisiting template-based automated program repair," Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 31-42, 2019.

[18] Shalini Kaleeswaran et al., "Semi-supervised verified feedback generation," Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 739-750, 2016.

[19] Andrea Arcuri, "On the automation of fixing software bugs," Companion of the 30th international conference on Software engineering, pp. 1003-1006, 2008.

[20] Claire Le Goues et al., "GenProg: A Generic Method for Automatic Software Repair," IEEE Transactions on Software Engineering, Vol. 38(1), pp. 54-72, 2012.

[21] Adam Roberts et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," Journal of Machine Learning Research 21, pp.1-67, 2020.

[22] ESLint, https://eslint.org/docs/rules/, Accessed: 2020-10-01.

[23] Sahil Bhatia et al., "Neuro-Symbolic Program Corrector for Introductory Programming Assignments," 2018 IEEE/ACM 40th International Conference on Software Engineering, pp. 60-70, 2018.

[24] Samuel Benton et al., "Defexts: A Curated Dataset of Reproducible Real-World Bugs for Modern JVM Languages," 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings, pp. 47-50, 2019.

[25] Deheng Yang et al., "Where were the repair ingredients for Defects4j bugs?," Empirical Software Engineering, vol. 26, pp. 1-33, 2021.

[26] Colin Raffel et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," Journal of Machine Learning Research, Vol. 21(140), pp. 1-67, 2020.

[27] Sumit Gulwani, "Example-based learning in computer-aided stem education," Communications of the ACM, Vol. 57(8), pp. 70-80, 2014

[28] Misoo Kim et al., "An Empirical Study of Deep Transfer Learning-Based Program Repair for Kotlin Projects," Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1441-1452, 2022.

[29] CodeChef, https://www.codechef.com/, Accessed: 2022-10-01.

[30] CodeForce, https://codeforces.com/, Accessed: 2022-10-01.

[31] Quoc Le et al., "Distributed Representations of Sentences and Documents," Proceedings of the 31st International Conference on Machine Learning, pp. 1188-1196, 2014.