IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

AUTOMATED EVALUATION OF PROGRAMMING ASSIGNMENTS

Moksud Alam Mallik[1], Nousheen Begum[2], Hafsa Yasmeen[3], Dr. Mujeeb Hasan[4]

Dean R&D and Associate Professor, Department of CSE (Data Science),

Lords Institute of Engineering and Technology, Hyderabad, India[1]

UG Student's, Department of CSE (Data Science), Lords Institute of Engineering and Technology, Hyderabad, India[2]

UG Student's, Department of CSE (Data Science), Lords Institute of Engineering and Technology, Hyderabad, India[3]

Assistant professor, Department of Physics, Lords Institute of Engineering and Technology, Hyderabad, India[4]

Abstract: Grading programming assignments by hand can be laborious and prone to mistakes. Feedback with failing test cases is generated using current tools. Nevertheless, this approach is ineffective and yields insufficient outcomes. In this paper, we provide AUTOGRADER, a tool that, given a single reference implementation of the issue, automatically assesses whether programming assignments are right and generates counterexamples. Our tool looks for semantically distinct execution pathways between a student's submission and the reference implementation rather than counting the passed tests. The proposal is considered wrong if such a discrepancy is discovered; if not, it is considered to be the right answer. To capture the semantics of execution paths and identify possible path variations, we employ symbolic execution and weakest preconditions. AUTOGRADER is the first automated grading tool that relies on program semantics and generates feedback with counterexamples based on path deviations. It also reduces human efforts in writing test cases and makes the grading more complete. We implement AUTOGRADER and test its effectiveness and performance with real-world programming problems and student submissions collected from an online programming site. Our experiment reveals that there are no false negatives using our proposed method and we detected 11 errors of online platform judges.

Keywords: Automated Assessment, Programming Assignments, Software Testing, Automated Grading, Evaluation of code

I. INTRODUCTION

In general, assessments relate to the instruments used by educators or educational supervisors to assess, measure, or ascertain a student's educational capacity, preparedness to learn what has been taught to him/her throughout time, and educational requirements. Stassen et al. define assessment as "the systematic collection and analysis of information to improve student learning." Since human evaluations are crucial for monitoring development across all age groups. This has often been accomplished by using tests developed by different educational organizations and the different standards established. There are various assessments that show how far a pupil has mastered the ideas or knowledge the teacher is attempting to instruct them [1]. Many academic fields employ subjective assignments with descriptive text to assess students' conceptual understanding of a subject. This has been further reinforced by situations like COVID-19, which have drastically altered the traditional classroom learning approach where students and teachers communicate in person. Instead, students and teachers now connect digitally through e-learning. All of these elements should be taken into account by an automated assessment system. Large text assignments made in response to a prompt are accepted as input by an automated evaluation system. After evaluating its quality in terms of subjectivity, substance, language, and readability, it should finally give the assignment a numerical score [2]. Functionality, design, and programming style are some of the main factors used to evaluate students' programming projects. Additionally, because the instructor must review every student assignment, providing feedback on their work is a demanding endeavor. For evaluating student assignments and providing feedback, numerous computer-aided techniques (CAAs) have been put forth [1]. Among the various methods employed in CAAs to evaluate student programs is the use of software metrics [3]. One key indicator of how well software systems are maintained is the readability of the source code. Up to 40% of the money allocated for software development may go toward maintenance. One of the key elements that significantly affects the readability of source code among software engineering methodologies is coding convention. A collection of coding rules known as coding conventions When writing code, software writers must adhere to certain rules. Coding norms are guidelines that cover every aspect of enhancing the quality of code. File structure, indentation, comments, white space, identifier names, declarations, statements, and programming language specific practices are some frequent categories for these principles [4]. Testing is a validation and verification procedure that establishes whether a certain system satisfies the initial criteria.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

seeks to find flaws, mistakes, or unmet requirements in software or systems that have been produced. The procedure gives stakeholders accurate information regarding the quality of the product. Software testing is crucial for a number of reasons, including cost savings. by spotting and resolving defects early in the development process, protecting user privacy, raising product quality, boosting customer satisfaction, and streamlining the process. To guarantee that software modules match the required quality, extensive software testing must be carried out throughout the software development lifecycle. There are two categories of software testing: static and dynamic. Code reviews, syntax checks, and walkthroughs are examples of passive code examination techniques used in static testing. Dynamic testing, on the other hand, allows security checks to be carried out while the code or application is running [5]. These days, software is an essential part of most systems and is incorporated into society's daily activities. With the development of technologies like open systems Software systems are growing increasingly complicated, as are highly automated or networked equipment. Furthermore, a number of People with a variety of specialties are typically needed. to participate in a software project, which also raises its degree of complexity. Because software is created by humans People are fallible beings; therefore, in every Errors usually happen in commercial software, and as the as complexity rises, these error ratios grow even greater. As a result, mistakes must be found and fixed as quickly as possible during the development process. In order to improve the quality of the software, various activities are performed under the heading of software testing. This procedure is an essential part of the software development life cycle that aims to create more dependable and superior software products and is both technically and economically necessary for a high-quality software product [6]. A key component of assessing the efficacy of education is the measurement and evaluation of students' knowledge acquisition. Summative assessment techniques, such as written exams, are essential instruments for monitoring learning outcomes, while formative assessment gives students continuous feedback through assignments. In order to improve productivity, lessen grading bias, and deliver timely feedback, educational institutions have been investigating automated grading systems (AGSs) driven by artificial intelligence (AI) more and more in recent years. Although automated grading techniques are used in many academic fields [7]. Programming assignment evaluation is a crucial yet resource-intensive aspect of computer science education that is often constrained by subjectivity and inefficiency. The burden on traditional grading techniques has increased due to the sharp rise in student enrollment, underscoring the need for automated solutions. By creating a machine learning system that automates the grading and feedback process for programming assignments, this study addresses these problems. The method ensures consistent, accurate, and quick evaluation by using natural language processing for feedback and supervised learning for grading. By overcoming the limitations of manual grading, this approach enhances learning outcomes, reduces teacher effort, and promotes a scalable, equitable, and effective grading process. Systems for automated feedback enhance instruction by providing tailored, data driven feedback [8]. In the domains of software engineering and natural language processing (NLP), automatic evaluation of code production is important and promising. Because code generation has the potential to revolutionize programming styles and lower development costs, both business and academics have given it a great deal of attention. Although code generation has advanced significantly in recent years, CEMs still need to catch up. Without appropriate CEM, it is difficult to assess the competitiveness of different strategies, which impedes the advancement of sophisticated code generating methods [9]. In the domains of software engineering and natural language processing (NLP), automatic evaluation of code production is important and promising. Because code generation has the potential to revolutionize programming styles and lower development costs, both business and academics have given it a great deal of attention. Although code generation has advanced significantly in recent years, CEMs still need to catch up. Without appropriate CEM, it is difficult to assess the competitiveness of different strategies, which impedes the advancement of sophisticated code generating methods [10]. Seizures, which are recurrent occurrences of brief interruptions or disruptions in neuronal communication, are the result of epilepsy, a catastrophic brain illness. Refractory seizures affect nearly one-third of epileptic patients, and many others have seizures with no apparent reason [11]. A major global public health concern, DR is a serious and progressive retinal consequence of diabetes mellitus. It is one of the main reasons why people of working age have irreversible eyesight loss, especially in underdeveloped and resource constrained areas. Early-stage DR is asymptomatic, which emphasises the necessity of frequent, universal screening to guarantee prompt identification and treatment.[12] Diabetic Retinopathy (DR) is a consequence of diabetes that results from damaged blood vessels (BVs) in the retina brought on by elevated blood sugar levels, which causes BV to leak and expand.[13].

II. LITERATURE REVIEW

Manual Grading

In introductory programming courses such as CS1, manual grading is commonly used to assess student submissions. A team of teaching assistants (TAs) is responsible for evaluating the assignments. Each TA is assigned a specific set of submissions to grade. The grading process involves two major components: automated test suite evaluation and manual inspection of the student\'s code. Manual inspection plays a particularly important role in CS1 because many first-year students, while understanding the underlying logic of programming, often struggle with writing syntactically correct code. These minor issues, such as syntax or runtime errors, can cause a program to fail entirely in automated tests, even

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

if the logic is mostly correct. To address this, TAs often award partial credit for submissions that are close to a correct solution. In some cases, they may even manually correct small mistakes to help determine if the student's logic is sound before running the test suite again. Additionally, TAs are expected to provide constructive feedback on why certain test cases failed, which is essential for student learning and improvement.

However, this manual grading approach has two significant drawbacks. The first is ****time**.** Introductory programming courses often have large enrollments, sometimes with hundreds of students. Manually reviewing each submission takes a considerable amount of time, making it difficult for TAs to return graded assignments quickly. This delay in feedback can hinder students\' ability to learn from their mistakes and apply that knowledge to future assignments or concepts. The second major issue is **variability** in grading. TAs usually come from diverse educational backgrounds, often having completed their own undergraduate studies at different institutions. This leads to inconsistencies in their programming knowledge and experience. Despite the presence of detailed grading guidelines and rubrics, the human element introduces subjective bias, resulting in unintentional discrepancies in how similar submissions are evaluated. These issues make it challenging to maintain fairness and consistency across the entire course.

Program Repairing

Another major challenge in CS1 courses is that novice programmers often struggle to understand compiler error messages. These messages are typically written for experienced developers and reference advanced concepts that beginners may not have encountered yet. For instance, in C-like languages, a common early mistake is forgetting to use the '&' operator when required. However, the compiler error message might refer to types like "int *," which can be confusing for students who haven't yet learned about pointers. As a result, beginners find it difficult to interpret and fix even simple syntax errors, such as missing semicolons or unmatched parentheses, especially in the early stages of the course. In practice, TAs often recognize the student's intent and deduct only minor marks for these mistakes, focusing more on the underlying logic. However, automated grading systems typically fail in these cases, as they cannot process programs that do not compile. To address this, a useful complement to automated grading is a program repairing system that can detect and correct basic syntax errors. Such a system not only fixes code but can also enhance error messages and provide meaningful feedback, helping students better understand their mistakes and improve their programming skills.

Table 1: An example of improved feedback by GradeIT.

Error Type	Missing & before a variable in scanf	
Compiler Message	format specifies type 'int *' but the argument has type 'int'	
GradeIT Feedback	You have not put an & before 'b' in the scanf statement on this line. Whenever you use scanf to input a value, you must put an & before the variable (except for pointers and strings).	
	Valid statements:	Invalid statements:
	1. `scanf("%d",&a);`	1. `scanf("%d",a);`
	2. `char st[20];` `scanf("%s",st);`	2. `char st[20];` `scanf("%s",&st);`

Automated Grading

Simple automated grading systems, like those used in programming contests, evaluate student submissions by running them against a suite of test cases, each weighted by difficulty. The final grade is the sum of the weights for the past test cases. While efficient, this approach can be limited in the quality of feedback it provides. Recent methods enhance this by using two key techniques. First, **clustering of similar submissions** groups semantically similar programs together, allowing instructors to give feedback on one representative, which can then be adapted for others in the cluster. Second, the concept of **semantic distance** compares incorrect submissions to a set of correct ones, measuring how many changes are needed to transform one into the other. This distance is used for both grading and generating targeted feedback. However, these advanced methods rely on the program being syntactically correct and successfully compiled, as they work with abstract representations like Abstract Syntax Trees (ASTs) or Control Flow Graphs. This creates a problem for CS1 students, who often struggle with syntax errors early on. Submissions that don't compile are assigned zero marks, even if they are logically close to the correct solution, which can be discouraging. In contrast, a compiling but meaningless program (like an empty one) may receive a higher score, highlighting a mismatch between grading and actual student understanding.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

System Analysis - 3.2

The workflow of AUTOGRADER is shown in Fig. 3.1. Given the reference implementation and one submission, we first compile these two programs into executable files. We then ask a white-box fuzzer to generate one program input and log the program execution trace by running the program executables with such an input. The next step is to compare the execution outputs; different execution outputs indicate the incorrectness of the submission. If the outputs are identical, however, we undertake further analysis to identify potential path deviations along the trace. As previously discussed, path deviations may not lead to real semantic differences, and we further rule out false positives by checking path equivalence between the two traces. Once a true positive (i.e., semantically different execution paths under the same input) is confirmed, we halt the analysis, grade the submission as incorrect, and output the counterexample. If no difference can be detected with the given input, we re-run the white-box fuzzer to yield new inputs and analyze further traces. A submission is concluded as correct once all the paths have been covered and no true positive deviation scan be detected. We elaborate on each of these steps in the following sections.

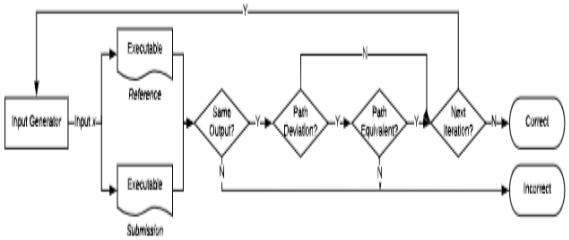

Fig 3.1 system workflow

A. Input Generation
B. Trace Logging
C. Symbolic Execution
D. Path Deviation Identification
E. Path Equivalence

The AUTOGRADER system enhances automated grading by deeply analyzing student submissions through a multi-step process. It begins by compiling both the student submission and a reference implementation, then uses **white-box fuzzing** to generate inputs and collect **execution traces** for both programs. If outputs differ, the submission is marked incorrect. If outputs are the same, further analysis checks for **path deviations**, which are potential semantic differences in logic.

To detect meaningful path deviations, AUTOGRADER performs **symbolic execution**, turning concrete traces into abstract formulas representing the logic paths. It then uses a **path deviation constraint** to find inputs that cause differences in behavior, solving it with **SMT solvers**. If a solution (i.e., a counterexample) exists, it indicates a possible error.

However, not all deviations signify real errors—students may solve problems differently but still correctly. To address this, AUTOGRADER includes a **path equivalence check**. It compares the semantics of deviating paths using a combined constraint on outputs and paths. If no input can produce differing outputs, the deviation is dismissed as a **false positive**. Only if a difference in output is confirmed, the submission is graded as **incorrect**, and the input that triggered the deviation is provided as feedback.

This approach ensures robust and fair grading by detecting logical errors beyond simple test case failures, while also minimizing false positives caused by alternative but correct student logic.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

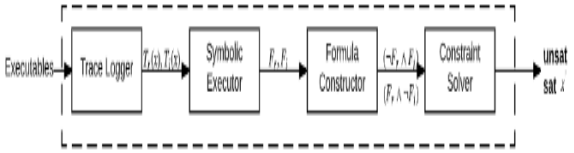

Fig. 3. Deviation identification

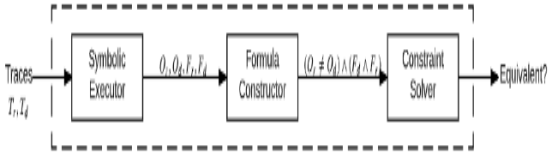

Fig. 3.2: Path equivalence checking

III. IMPLEMENTATION TECHNIQUES AND ALGORITHM

The goal of the Automated Evaluation of Programming Assignments system is to provide a platform that can automatically evaluate student programs by gathering, running, and evaluating them without the need for human intervention. Students upload their code written in languages like Python, Java, C, or C++ to the system's submission interface. The system preprocesses the program after a student submits it, looking for syntax mistakes, missing files, and dangerous code portions like hazardous commands or infinite loops. This guarantees that the code that has been provided is safe to run in the controlled environment.

Following preparation, the code is sent to the automated execution module, which executes the program in a sandbox environment built with virtual machines or Docker containers. In order to minimize unintentional or malevolent harm, this sandbox isolates the student program and limits access to system resources. Predefined sample test cases and concealed test cases are used to run the supplied software several times. The system keeps an eye on the output, runtime performance, and memory utilization of the application while it is running. Test-case matching algorithms are used to automatically compare the output produced by the student's software with the anticipated output. The test case is deemed successful if the output matches; if not, the system logs the cause of the discrepancy.

To confirm software correctness, the evaluation engine makes use of automated test frameworks like Python's unit test, Java's JUnit, or specially designed test runners. These frameworks support error handling, boundary conditions, and logical accuracy checks. The system measures execution time and memory usage in addition to accuracy. The application produces feedback like "time limit exceeded" or "memory overflow" if it goes beyond the permitted limitations. Static code analysis techniques can also be utilized to support qualitative evaluation by examining unused variables, naming conventions, and programming style.

The system provides the student with comprehensive feedback when execution and analysis are finished. Compilation errors, runtime errors, erroneous outputs, or assurance that every test case was successfully completed could all be included in the feedback. Correctness, the quantity of test cases passed, program performance, and optional code quality are all taken into account when calculating marks. Because the results are shown on the user interface, students may quickly identify their errors and make improvements to their code. Through a different interface, teachers may also see how well their students are performing.

A number of tools and software components are merged to create this system. Usually, Python (with Django or Flask), Java (with Spring Boot), or Node.js are used to construct the backend. For a clear and responsive design, the frontend interface is made with HTML, CSS, JavaScript, and Bootstrap. Test cases, evaluation results, and student contributions are stored in databases like MySQL, PostgreSQL, or MongoDB. Programs must be compiled and executed using compilers such as GCC for C/C++, JDK for Java, and Python interpreters. To guarantee security, the entire assessment procedure is carried out in a sandboxed environment that is driven by Docker or Linux containers. While editors like VS Code, PyCharm, and Eclipse aid in development, version control systems like GitHub or GitLab help manage project code.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

To put it briefly, the system receives student inputs, preprocesses them, runs them securely in a sandbox, compares outputs using automated testing frameworks, computes grades, and provides immediate feedback. The learning process is made more effective and efficient by this implementation, which also lessens the manual workload for teachers and provides students with a quicker and more accurate assessment of their programming assignments.

IV. RESULT

Testing Strategy

Implementation and Testing

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

Implementation

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modified as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

Testing with Screenshots

To implement this project, we have use following reference and submission files

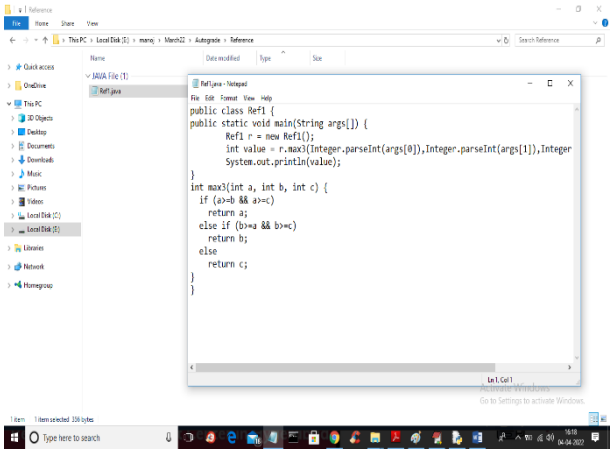

Figure 6.2.1: Sample Reference and Submission Files

In above fig 6.2.1 you can see one sample reference file and below are the submission files

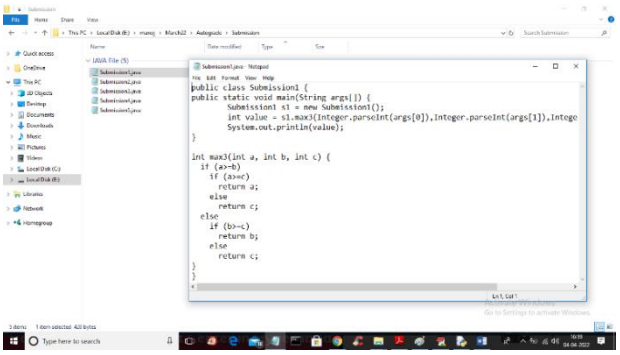

Figure 6.2.2 : File Submission Files Input to AutoGrader

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

In above Figure 6.2.2 we can see file submission files and both reference and submission files are the input to AutoGrader

SCREEN SHOTS

To run project double click on 'run.bat' file to get below output

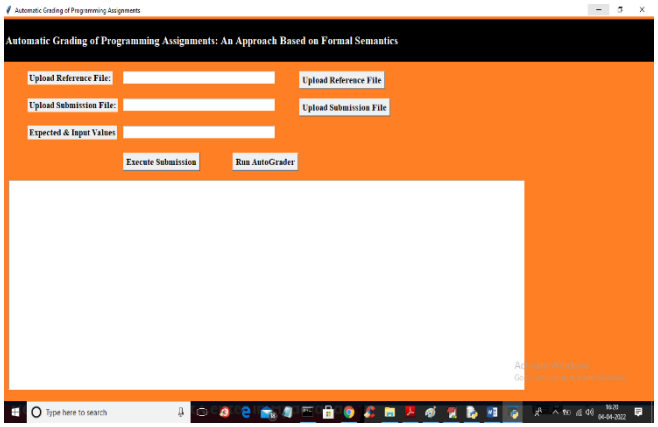

Figure 6.2.3: Upload Reference File Button

In above Figure 6.2.3 screen click on 'Upload Reference File' button to upload file

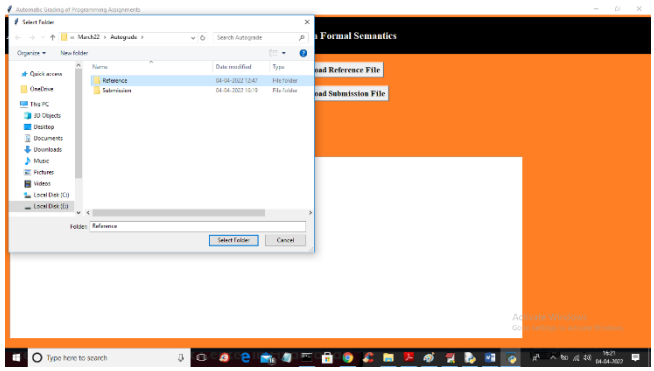

Figure 6.2.4: File Explorer

In above Figure 6.2.4 selecting and uploading 'Reference' folder which contains reference file and then click on 'Select Folder' button to load file and get below output

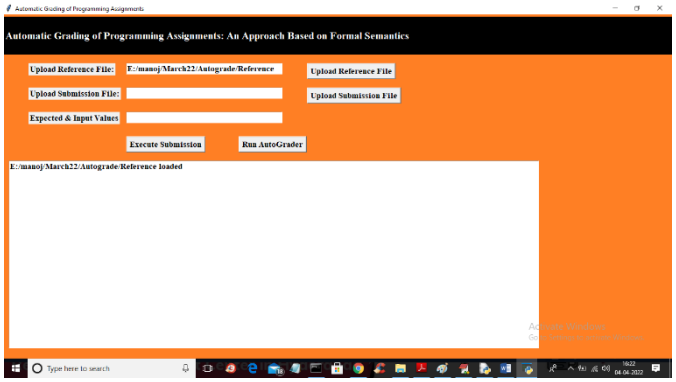

Figure 6.2.5 : File Selection

In above Figure 6.2.5 reference file loaded and now click on 'Upload Submission Files' button to upload submission files

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

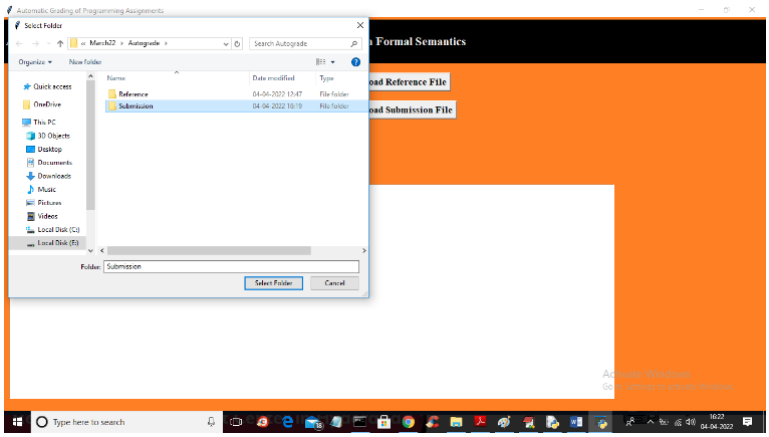

Figure 6.2.6: Status Message

In above Figure 6.2.6 selecting and uploading submission folder and then click on 'Select Folder' button to get below output

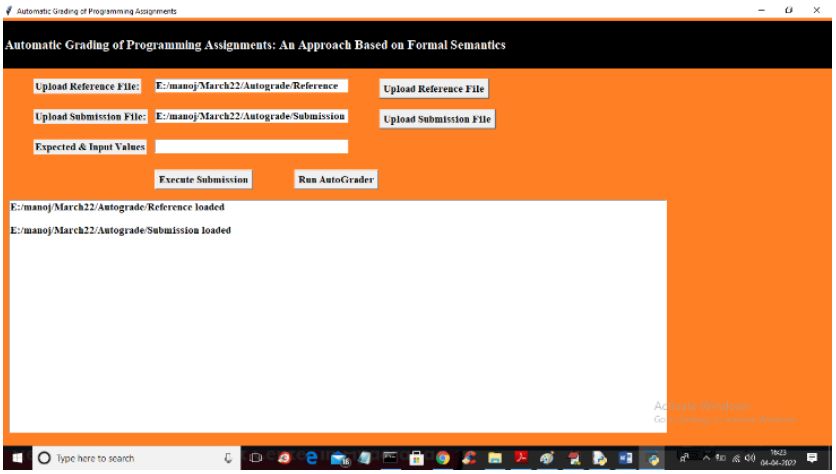

Figure 6.2.7: Output Display

In above Figure 6.2.7 both reference and submission files are loaded and now in 3rd text field enter expected output and input values separated by comma like below screen

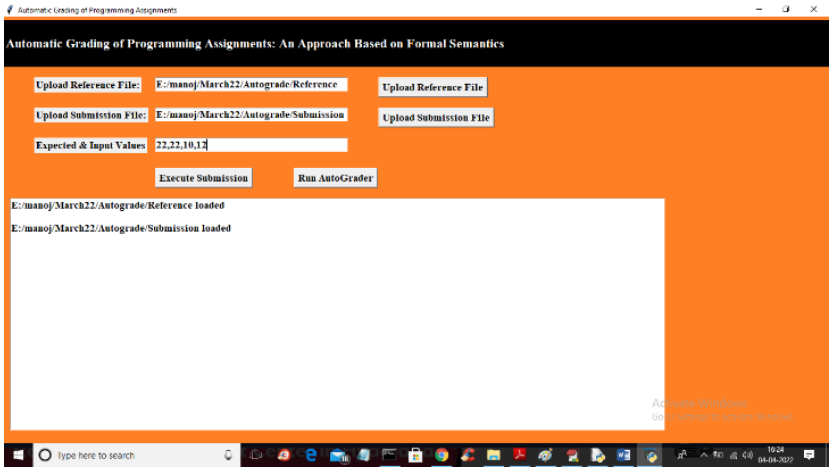

Figure 6.2.8: Input Fields

In above Figure 6.2.8 3rd field I entered first output value as 22 and then enter 3 numbers to find maximum of those 3 numbers and we can out of last 3 numbers 22 is the maximum so expected output is 22 and below is the executed output

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155179

6.3 Result Snapshots

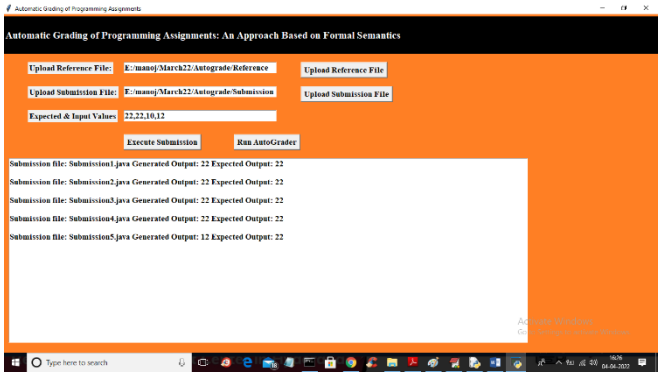

Figure 6.3.1: Generated Output

In above Figure 6.3.1 we can see generated output of first 4 submission programs are correct as 22 and last one giving as 12 so its execution output is wrong and now click on 'Run AutoGrader' button to calculate diverse path and then find correct and incorrect programs. Student may get correct output by writing some wrong logic so diverse path will check whether student logic and reference program logic is matching or not

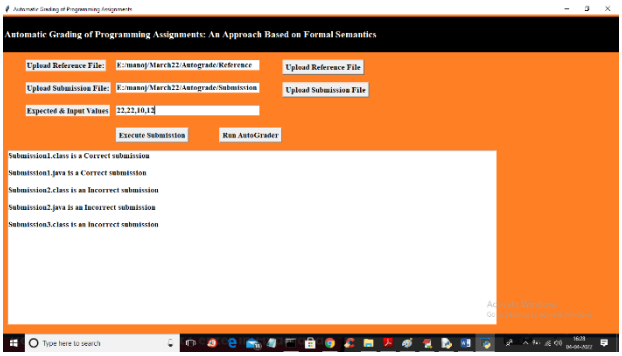

Figure 6.3.2: Program Output

In above Figure 6.3.2 first two programs output was correct and logic also correct and 3rd and 4th got correct output but its logic is not right and 5th one got wrong executed output so its logic will also wrong Similarly you can write and check any other program with reference and submission files

V. CONCLUSION AND FUTURE ENHANCEMENT

7.1 – Conclusion

We have proposed a formal-semantics-based approach for automatically grading programming assignments with a single correct reference implementation provided by the instructor. By searching for inputs that can lead to path deviations between a student's submission and the reference implementation, we can determine the correctness of the submission and provide counterexamples for an incorrect submission. We have implemented this novel approach using a tool called AUTOGRADER and evaluated our technique over a dataset containing 10,270 submissions collected from an online programming site. The experiment revealed that our proposed method yielded no false negatives, while the online programming site yielded 11 false negatives due to incomplete test suites.

7.2 – Future Work

In this section, swe discuss the limitations of AUTOGRADER and suggest directions for future work. First, AUTOGRADER only determines the correctness of programming assignments and provides counterexamples. In other words, our method decides the functional correctness. Off-the-shelf automatic graders usually rely on the number of passed tests to mark a submission's "grade" in addition to its correctness. However, a meaningful grading system should identify the shortest distance between the correct implementation and submissions. Traditional grading methods do not represent the score in this way and rely heavily on the quality of test suites. On the contrary, our method conducts the analysis with finer granularity. In the future, we plan to leverage model counters [63], which approximate the number of satisfiable models for boolean formulas, or similar techniques to calculate the distance between submissions and the

DOI: 10.17148/IJARCCE.2026.155179

correct reference. Moreover, on top of the recorded trace, we plan to detect the redundant code in a program and use this information to grade assignments in terms of "elegance".

Second, AUTOGRADER is limited by the capability of constraint solver and symbolic execution tools. During the path deviation detection process, if the constraint solver finds a sat assignment to the formula, it is truly satisfiable. However, an output of no may mean the formula is unsat, or the solver cannot find a satisfiable assignment limited by its capability. This can potentially lead to false negatives. We tackle the problem by iterating the process for many rounds, thereby reducing the probability of such cases. A similar situation may occur during the equivalence checking process, and our tool would theoretically report false positives. In our experiments, we have not seen such false positives or false negatives. Finally, AUTOGRADER suffers from the common limitations of current symbolic execution tools, such as that they cannot perform non-linear arithmetic operation or floating point calculation.

REFERENCES

[1]. Omopariola, A. V., Ogbonna, C. N., Uloko, F., & Abdullahi, M. (2023). Automated assessment system using machine learning libraries. Open Journal for Information Technology, 6(2), 97–122. https://doi.org/10.32591/coas.ojit.0602.02097o

[2]. Birla, N., Jain, M. K., & Panwar, A. (2022). Automated assessment of subjective assignments: A hybrid approach. Expert Systems With Applications, 203, 117315. https://doi.org/10.1016/j.eswa.2022.117315

[3]. Koyya, P., Lee, Y., & Yang, J. (2013). Feedback for programming assignments using Software-Metrics and Reference Code. ISRN Software Engineering, 2013, 1–8. https://doi.org/10.1155/2013/805963

[4]. Chen, H., Chen, W., & Lee, C. (2018). An automated assessment system for analysis of coding convention violations in Java programming assignments. Journal of Information Science and Engineering, 34, 1203–1221. http://jise.iis.sinica.edu.tw/JISESearch/pages/View/PaperView.jsf?keyId=164_2173

[5]. Kumar, S. (2023). Reviewing software testing models and optimization techniques: an analysis of efficiency and advancement needs. Journal of Computers Mechanical and Management, 2(1), 32–46. https://doi.org/10.57159/gadl.jcmm.2.1.23041

[6]. Gurcan, F., Dalveren, G. G. M., Cagiltay, N. E., Roman, D., & Soylu, A. (2022). Evolution of Software testing Strategies and Trends: Semantic content analysis of software research corpus of the last 40 years. IEEE Access, 10, 106093–106109. https://doi.org/10.1109/access.2022.3211949

[7]. Tan, L. Y., Hu, S., Yeo, D. J., & Cheong, K. H. (2025). A Comprehensive review on Automated grading systems in STEM using AI techniques. Mathematics, 13(17), 2828. https://doi.org/10.3390/math13172828

[8]. Kavita, N. (2025). Automated grading and feedback systems for programming in higher education using machine learning. Journal of Informatics Education and Research, 5(1). https://doi.org/10.52783/jier.v5i1.2142

[9]. Dong, Y., Ding, J., Jiang, X., Li, G., Li, Z., & Jin, Z. (2024). CodeScore: Evaluating code generation by learning code execution. ACM Transactions on Software Engineering and Methodology, 34(3), 1–22. https://doi.org/10.1145/3695991

[10]. Shi, E., Wang, Y., Du, L., Chen, J., Han, S., Zhang, H., Zhang, D., & Sun, H. (2022). On the evaluation of neural code summarization. Proceedings of the 44th International Conference on Software Engineering, 1597–1608. https://doi.org/10.1145/3510003.3510060

[11]. K. D. Tzimourta, "Evaluation of window size in classification of epileptic short-term EEG signals using a Brain Computer Interface software", Eng. Technol. Appl. Sci. Res., vol. 8, no. 4, pp. 3093–3097, Aug. 2018.

[12]. K. V. Shanthala and N. C. Kundur, "DR-EfficientNet-L: A Distributed Deep Learning Architecture for Efficient Detection and Grading of Diabetic Retinopathy", Eng. Technol. Appl. Sci. Res., vol. 15, no. 5, pp. 28362–28367, Oct. 2025.

[13]. S. I. S. M. Shazuli and A. Saravanan, "Manta Ray Foraging Optimizer with Deep Learning-based Fundus Image Retrieval and Classification for Diabetic Retinopathy Grading", Eng. Technol. Appl. Sci. Res., vol. 13, no. 5, pp. 11661–11666, Oct. 2023

BIOGRAPHY

Moksud Alam Mallik was born in Howrah, Kolkata, India, in June 1985. He received the B.Tech. (CSE) degree (Hons.) from the Maulana Abul Kalam Azad University of Technology, West Bengal, India, in 2005, the M.Tech. (CSE) degree (Hons.) from Visvesvaraya Technological University (VTU), Karnataka, India, in 2014, and the Ph.D. (CSE) degree from International Islamic University Malaysia (IIUM), Kuala Lumpur, Malaysia, in 2023. He joined as an Associate Professor and the HOD with the Department of Computer Science and Engineering-DS, LIET, Hyderabad, India, in August 2022, and also take over as the Dean Research and Development at LIET, from 2024. He has more than 17 years of teaching and

DOI: 10.17148/IJARCCE.2026.155179

industrial experience in reputed engineering colleges and renowned MNCs. He has five book chapters and 13 research publications in reputed journals (Scopus and WoS), conferences, proceedings of which are published by Springer and IEEE. His research interests include parallel clustering algorithm, big data analytics, datamining, machine learning, and data science.*(Based on document published on 19 September 2024).*

Nousheen Begum is a third-year student pursuing a B.E. in Data Science and Computer Science and Engineering. Intelligent systems, data clustering, and machine learning are her areas of interest. The conference paper "Comparative Studies of Different Fuzzy-C-Means Clustering Algorithms for Machine Learning" (IRJIET, Paper ID: INSPIRE-25116) was published and presented by her.

Hafsa Yasmeen is a third-year student pursuing a B.E. in Data Science and Computer Science and Engineering. Intelligent systems, data clustering, and machine learning are her areas of interest. The conference paper "Comparative Studies of Different Fuzzy-C-Means Clustering Algorithms for Machine Learning" (IRJIET, Paper ID: INSPIRE-25116) was published and presented by her.

Dr. Mujeeb Hasan is currently working as an Assistant Professor in the Department of Physics at Lords Institute of Engineering and Technology. Since March 2024, he has also been serving as the Associate Dean of Research and Development at the institute.

He completed his Bachelor of Science (Hons.) in Physics and Master of Science in Physics from Aligarh Muslim University (AMU), Aligarh, India. He earned his Ph.D. in Physics from the Indian Institute of Technology (IIT) Roorkee in May 2021.

Dr. Hasan has demonstrated academic excellence, having qualified the Graduate Aptitude Test in Engineering (GATE) in Physics twice—in 2014 with an All India Rank (AIR) of 463, and in 2015 with an improved AIR of 177.

He has published six research papers in peer-reviewed journals, and has presented three papers at national and international conferences, with the proceedings published. Additionally, he has attended nine national and international conferences, reflecting his continued engagement with the broader scientific community.

