
Journal of Informatics and Web Engineering

Vol. 4 No. 1 (February 2025)

eISSN: 2821-370X

DebugProGrade: Improving Automated Assessment of Coding Assignments with a Focus on Debugging

Amit Patel^{1*}, Hardik Joshi²

¹Veer Narmad South Gujarat University, Udhana - Magdalla Rd, Surat, Gujarat 395007, India.

²Gujrat University, Navrangpura, Ahmedabad, Gujarat 380009, India.

*corresponding author: (iamitrp@gmail.com; ORCID: 0000-0001-6607-1870)

Abstract – In education, the evaluation of programming assignments is challenging, especially to do with the debugging aspect. Self-grading technologies are unable to capture the level of understanding of students and context-bound responses. In light of these, we created DebugProGrade to take what we normally know about grading and improve it with semantic analysis and keyword extraction. DebugProGrade identified 1000 first-year BCA students who in Google Forms provided their answers to evaluate error detection and solution proposals for a basic C programming assignment. For the explanations' specificity and for the context-level evaluation, the system employs the SBERT embedding, namely, the sentence-transformer bidirectional encoding representations from transformers. We employ the methods with tuned parameters and apply academic criteria to the evaluations performed by them. Other key functionality in DebugProGrade that should be mentioned is the classification of debugging skills into competence levels providing more comprehensive view of student proficiency regarding bugs which remain unaddressed by traditional grading systems – that is the ability to identify or fix bugs. Upon optimizing the Gradient Boosting Regressor algorithm, it gives outstanding results in terms of evaluating and predicting redshift. The mean squared error is very low with a value of $MSE = 0.025107$ and the MAE is also quite low with the value 0.031335 , overall the high R^2 score 0.99932 shows that the given dataset has been predicted with high accuracy with reference to the target variable. DebugProGrade precisely flips the paradigm of conventional grading and provides us with even greater understanding of where exactly students are strong.

Keywords—DebugProGrade, Semantic Analysis, Debugging Skills, Keyword Extraction, Machine Learning

Received: 31 August 2024; Accepted: 21 December 2024; Published: 16 February 2025

This is an open access article under the [CC BY-NC-ND 4.0](https://creativecommons.org/licenses/by-nc-nd/4.0/) license.



1. INTRODUCTION

Automated grading system in large classroom size is in demand in the computer science course to reduce much time, cost, and personal bias in evaluating the programming assignment. With the high workload associated with manual grading systems, they are time-consuming, inconsistent, and depend on the instructor's emotions. This variation in student assessment occurs across larger classroom sizes and other academic workload [1], [2]. Automated grading system as the system, which prevent these challenges by building rapid, honest, and consistent systems. Because this system provides timely and accurate feedback from peers and instructor, it is useful for programming courses to sharpen their knowledge. It also makes the grading scalable [3] as well as allows the instructor to handle large class sizes utilizing quality grading. The system assesses multiple dimensions of student work (code accuracy, efficiency,

and style). It also explains what challenges you, the students face during assignment submission and common mistakes you make in the process. This is useful for an instructor to adjust the teaching strategies. It would be useful to improve the accuracy and fairness of grading programming courses in computer science through ways such as information retrieval, sentiment analysis, natural language programming/processing (NLP), and improving machine-learning techniques.

Early studies have looked into autograding system for programming assignment. Autograder [4] and GradeScope [5] system primarily used to check code correctness. While these system were effective for grading code that was syntactically accurate or not. They did not focus on important aspects which measure student behavioral skill. They focus on solution of programming assignment, not on how much student understand that particular topic, their ability to understand the flow, identify and rectify the error or the reason behind the error. Another automated grading system focused on short answer responses in various courses, but it has its own limitations. These systems relied on keyword matching and basic semantic analysis, which made them limited to fully understand and evaluate student response [6]. Latterly, more advanced approach like Syntactic, semantic analysis used in C-Rater [7], statistical methods employs in AutoSAS [8], and rule based approach employs in AutoMark [9]. Despite improvements periodically, many system struggle with subjective tasks when the grading require deeper reasoning.

In this paper, we propose DebugProGrade that introduces an innovative approach for assessing debugging skills in C programming assignments to tackle these challenges through our debugging skill quantifier model. This autograding tool aims to enhance the evaluation of students' programming by evaluating their code purely on merit as well as debugging skill. This tool enhances the grading process, allowing instructor additional time and support for students. This method uses keyword-based search to find necessary information from the dataset based on keywords [10], [11]. Besides, it uses SBERT embedding to measure sentence similarity, which provides its capability to make a much deeper and literal assessments of relevance by having semantic relations between the texts [12], [13]. Applied procedure ensure the accuracy of information extraction at syntactic and semantic level. The machine-learning model was fine-tuned and cross-validated to optimize performance, adapting challenges regarding present of noisy data and feature selection to optimize its performance.

The study compared these models with alternative machine learning techniques using the value obtained from these metrics - Root mean square error (RMSE), mean square error (MSE) and the R squared (R²) score. In this study, the understanding of the code, evaluated with respect to student's debugging skills, has been looked into and the areas where students may need some help from their teachers have been explored. What the findings emphasize is the potential of automated grading systems to improve grading, decrease teachers' workload, and improve students' learning outcomes.

The remainder of this paper is organized as follows. Section 2 provides a literature review of previous research. Section 3 presents the proposed system and Section 4 provides an in-depth discussion of the findings and results derived from the limitations of the system. In Section 5 after a summary of the main results of our study, we conclude.

2. LITERATURE REVIEW

In the traditional grading system of computer-science, programming subjects the teacher is in a position to go through each of assignments individually. The challenges include it is time consuming, irregular and sometimes the human factors can be depicted in the grading process [14], [15]. To address these challenges, there has been effort to develop automated grading system that is efficient, precise and provides more reliable outcome [16], [17]. In this section, the current automated grading systems will be discussed along with the strengths and weaknesses of each and the opportunities that DebugProGrade reaps and seeks to address.

2.1 Existing Automated Grading Systems

Moss (Measure of Software Similarity), used for plagiarism detection in programming assignments and determining a measure of software similarity. The techniques it uses for similarity identifications are tokenization, normalization, similarity detection. It has its limitations; it could get false positives and say one piece of code is similar to another that is not. Syntactic analysis is used and can miss other forms of similarity. It can also be that the result accuracy is a consequence of the code quality submitted. The importance of human oversight is thus emphasized for getting the results correct. However, a human evaluation should be combined with Moss to assure an accurate assessment. The

results of this study point to ongoing research towards better automated grading systems and solving the existing limitations [18], [19].

CodeRunner is a platform for improving coding skills of programmers. The more languages it supports the better it works well and the students can easily learn how to program. It requires a stable Internet connection for CodeRunner to perform, but CodeRunner is also likely to crash if you enter in a long series of tests. The interface might seem complicated for beginners and take some hours to know your way around it. However, despite these limitations, it is a handy tool for discussions during coding practice and development [20], [21].

Gradescope was designed to help instructors grade coding assignments and exams more quickly by automating the grading process and allowing them to quickly review and respond to student work. Teachers can submit both their handwritten and digital submissions for grading at the same time with the ability to customize grading criteria. However, new users will struggle to learn the system and tackling different types of assignments can cause headaches. For instance, as student submissions of all ranges of quality as well as descriptions of the grading criteria will also affect how effective the platform [5], [22].

The EduTools plugin for IntelliJ IDEA offers automated grading support for coding exercises across multiple programming languages, delivering immediate feedback while primarily focusing on ensuring that the code meets functional requirements, comparable to CodeRunner. Although it provides the advantage of supporting various languages and quick feedback, its main emphasis remains on functional correctness, which limits its capacity to comprehensively evaluate a student's programming skills, including aspects such as code readability and maintainability [23], [24].

Web-based Center for Automated Testing (Web-CAT) prioritizes testing and test coverage in the grading process, assessing students based on their ability to effectively test their own code. This approach fosters a deeper understanding of software testing principles and motivates students to write comprehensive tests that improve their coding abilities. However, this method requires students to have a solid grasp of writing tests, which can be challenging for beginners who are still mastering basic programming concepts [24], [25].

Machine learning is employed by the DeepCode tool to examine code submissions and provide insights on potential issues and code quality. Through training on a vast code database, DeepCode can recognize common coding mistakes and propose enhancements, making it beneficial for both educators and students. Nevertheless, DeepCode may face challenges in understanding context and might generate false positives, resulting in irrelevant recommendations. Moreover, its efficacy is contingent on the caliber of the training data [26].

AutoLep swiftly delivers feedback to students regarding syntactic and structural errors upon submission. The system conducts evaluation through dynamic testing and offers feedback indicating which test cases succeeded or failed during the submission process. Furthermore, it generates feedback via static analysis by comparing submissions to a model solution based on semantic analysis [27], [28].

AutoTutor, an intelligent tutoring system, employs latent semantic analysis (LSA) to evaluate student responses. By comparing these responses to a predetermined set of anticipated answers, AutoTutor can assess the depth of information covered and adjust the tutorial process accordingly. However, a challenge faced by AutoTutor is its potential difficulty in quickly determining whether a student's answer meets the expected criteria, which can be problematic in semantic evaluation [29].

Existing systems focus on evaluating the code correctness and plagiarism detection. It overlooks important aspects of students' learning process such as debugging skills, problem-solving skills and the cognitive processes involved in programming. Moss, IntelliJ IDEA, and Web-CAT provide technical evaluations, but they fail to assess deeper aspects of a student's understanding, especially when it comes to identifying errors and debugging code. Additionally, these systems do not offer instructors enough feedback to pinpoint learning gaps or adjust their teaching methods, which limits their ability to improve educational outcomes effectively.

The proposed system is designed to address the limitations of existing automated grading systems and introduce a comprehensive and innovative approach to evaluating programming assignments. It not only assesses the correctness of the final solution but also evaluates the student's debugging skill by asking them to identify errors, explain their sources and propose solutions. The System uses keyword extraction and semantic analysis to compare student responses with the teacher's answer by considering the student's response length, keyword matching and response similarity. This allows a detailed evaluation of the student's understanding of programming concepts and their

cognitive processes. Additionally, the system provides valuable insights to instructors, helping them identify students who may be struggling and adjust their teaching methods accordingly, which can ultimately lead to improved learning outcomes. The summary of existing tools is provided in Table 1.

Table 1. Summary of Existing Tools With Its Limitation

Tool Name	Purpose	Limitation
Moss [18, 19]	Detects software plagiarism.	Does not contribute to grading
CodeRunner [20, 21]	Interactive coding and testing environment	Focused solely on functional correctness
Gradescope [5, 22]	Streamlines grading of programming assignments and exams, supporting both handwritten and digital work.	Effectiveness influenced by submission quality and rubric clarity.
EduTools [23, 24]	Offers automatic grading of coding exercises, supporting multiple languages with instant feedback.	Focuses on functional correctness, which limits holistic assessment of programming skills, including code readability and maintainability.
Web-CAT [24, 25]	Emphasizes testing and test coverage as part of the grading process, encouraging comprehensive test writing.	Requires students to have a good understanding of writing tests, which can be a barrier for beginners.
DeepCode [26]	Analyzes code submissions using machine learning to detect bugs and suggest improvements.	May struggle with contextual understanding, leading to false positives; effectiveness depends on the quality of training data.
AutoLep [27, 28]	Provides instant feedback on syntactic and structural errors	Does not contribute to grading
AutoTutor [29]	Assess Student Responses.	Difficulty in promptly assessing whether a student's response aligns with the expected criteria

3. RESEARCH METHODOLOGY

The aim of this research was to create a sophisticated automatic grading mechanism for coding assignments. The system's primary focus was on evaluating students' understanding of coding errors, their capacity to solve problems, and their ability to express solutions effectively. The methodology encompasses data collection, preprocessing, feature engineering, rubric-based marking, similarity measurement, machine learning models, and final result-prediction modules. The system flow is illustrated in Figure 1. First, the inputs collected from the user and consisted of keywords, solutions, and answers.

3.1 Input

3.1.1 Keywords

Keywords are question-specific items essential for answering this question. These keywords play a significant role in penalizing or promoting the score evaluated by the similarity measurement module, and must only contain the essential words in the lower case.

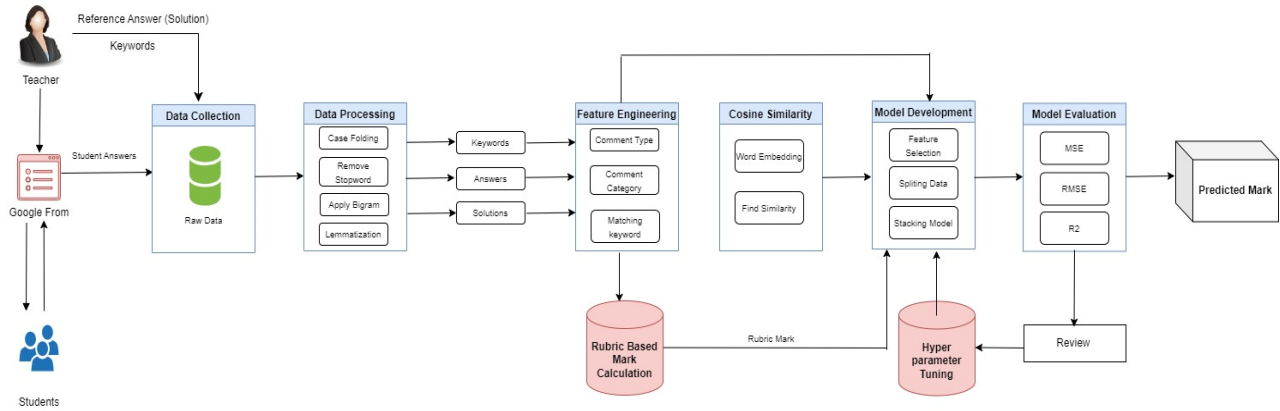


Figure 1. Flow of DebugProGrade System

3.1.2 Reference Answer (RA)

The solution is a subjective answer used to map students’ responses. This solution must contain all the keywords and contexts discussed in the answers in separate lines and paragraphs. The teacher/evaluator typically prepares solutions for this question.

3.1.3 Student Response (SR)

The answer was a subjective response from students to the evaluation. It usually contains some or all of the keywords and spans one to a few sentences depending on the student’s writing style. It contains an analysis of the program by explaining the programming error, its reason, and solution. In the context of this study, the term 'comment' is utilized to denote the written feedback offered by students in response to a programming assignment.

3.2 Data Collection Module

The foundation of this research was to collect student responses from approximately 1000 FYBCA students using Google Forms. The form presented students with a basic C programming task that required them to identify errors, provide reasons for these errors, and propose solutions. Simultaneously, for autograding, a reference answer provided by a teacher and a list of important keywords related to the assigned program were input into the system. Table 2 shows the programming task assign to students with teacher reference answer, teacher answer keywords and student responses.

Table 2. Programming Assignment Question with Teacher Reference Answer and Student Responses

Question	<pre>#include <stdio.h> void main() { a = 10; printf("The value of a is : %d", a); }</pre>
Teacher Reference Answer	Error: There is compile time errors. Variable is not declare. Solution: Variable must be declare before use. Here we have to declare variable a as integer . Add statement “int a” before “a=10”.
Teacher Answer Keywords	Compile , variable', 'declaration', 'integer', 'int', 'undeclare', 'declare', 'datatype'
Student Response-1	the code you provided is missing a declaration for the variable "a". to fix this error, you should add a line at the beginning of the main function that declares "a" as an integer.
Student Response-2	Declaration error

Student Response-3	<p>1) Variable a is used without declaration. Solution: int a=10;</p> <p>2) missing the library. Solution: #include<conio.h></p>
Student Response-4	<p>Errors in this code. First, the variable `a` is not declared before it is assigned a value. Second, the code is missing the data type of `a`. To fix these errors, we need to declare the variable `a` before assigning it a value. Here's the corrected code:</p> <pre>#include <stdio.h> int main() { int a = 10; printf("The value of a is: %d", a); return 0; }</pre>

The dataset used in this study was collected through Google Forms and distributed to students enrolled in a C programming course. The form presented a C programming assignment containing intentional errors and prompted students to identify the error, explain its reason, and propose a solution. Initially, the dataset contained the following columns (refer to Table 3):

Table 3. Dataset Attributes With Description

Attribute	Description
ID	Unique identifier for each student response.
Name	Student's name.
Gender	Student's gender.
Email Address	Contact information for the student.
Timestamp	Time of submission.
Comment1	The original student response.

3.3 Data Pre-processing Module

The preprocessing step is a very important step when it comes to working with raw texts and converting them into a form that can be fed into the feature extraction, as well as into the model for training. After receiving inputs from the user, both the solution and answer go through a preprocessing step. First, this process involves case folding, which transforms all letters into a lower case so that the data are standardized and not influenced by factors such as capitalization. Punctuation marks were eliminated to concentrate only on words with meanings for the analysis. The text is tokenized, which is the process of dividing text into smaller units or tokens, in this case words, such that each word can be processed in the following steps. Low informative words and punctuations for instance 'and,' 'the,' and 'is' are normally omitted as they do not add on value to the content. Finally, Lemmatization was applied to both student comments and the teacher's reference answer to reduce words to their base forms, considering parts of speech (POS) tags such as verbs, adjectives, adverbs, and nouns. The Gensim library facilitated this process, which also contributed to a more refined word frequency analysis and the creation of a word cloud that visually represented the most frequently used terms.. The results of this stage include clean and tokenized text data suitable for feature engineering.

Pre-processing converts raw text into clean text by lowercasing, removing punctuation, tokenizing, removing stop words, and lemmatizing. Algorithm 1 gives information about all the pre-processing steps.

Algorithm 1 Pre-processing Data

Input:

Raw data (X)

Output:

Lemmatized text data.

Process:

1: Convert text to lowercase

 $X_lower = \text{Lowercase}(X)$

2: Remove punctuation

 $X_no_punct = \text{RemovePunctuation}(X_lower)$

3: Tokenize the text

 $X_tokens = \text{Tokenize}(X_no_punct)$

4. Remove stopwords

 $X_clean = \text{RemoveStopwords}(X_tokens)$

5. Lemmatize the tokens

 $X_lemmatized = \text{Lemmatize}(X_clean)$

6. Return X_lemmatized

3.4 Feature Engineering Module

Feature engineering is crucial for capturing relevant attributes of student responses. The following features were derived:

Comment Type(CT): Classified as 'basic' (CT=1) if the response had fewer than a threshold number of words (e.g., 15 words) and 'detailed' (CT=2) otherwise. This represents the depth of students' answers.

Comment Category(CC): Comments were categorized into four types based on their content: error-only (CC=1), solution-only (CC=2), both error and solution (CC=3), or none (CC=4). This represents the relevance of the content, which was achieved by creating lists of synonyms for errors and solutions and then identifying the presence of these words in the comments.

Keyword Matching Percentage(KM): calculated as the ratio of matching keywords between the student response and reference answer as in Equation (1).

$$KM = \left(\frac{MK}{TK} \right) \times 100 \quad (1)$$

where KM represents the Keyword Matching percentage, MK represents the number of matching keywords, and TK represents the total number of keywords in the reference.

3.5. Rubric Based Mark Module

The Rubric-Based Mark Module is designed to evaluate and assign scores to student responses based on a predefined set of criteria. These criteria assess various aspects of a student's answer, including the level of detail, content focus, and relevance of the response to the reference answer. The module considers three primary features: Comment Type (CT), Comment Category (CC), and Keyword Matching (KM). These features collectively contributed to the overall score, which was scaled to a maximum of 30. A base mark was awarded to answer the question. Table 4 lists the coefficient values assigned to each feature based on their responses as in Equation (2).

$$Final_{RebricMark} = Scaling_{Factor} \times R_{Score} \quad (2)$$

$Scaling_{Factor}$ and R_{Score} , used in Equation (2) is calculated using following formula.

A scaling factor was applied to ensure that the total score did not exceed the maximum score as in Equation (3).

$$Scaling_{Factor} = \frac{M_{max} - B_{Mark}}{R_{score}} \quad (3)$$

where B_{Mark} is the base mark and M_{max} is the mark from which the final mark is calculated. R_{score} was calculated using Equation (4).

$$R_{score} = \left(\frac{\beta_{CT}}{3}\right) \times 5 + \left(\frac{\beta_{CC}}{3}\right) \times 5 + \left(\frac{\beta_{KM}}{3}\right) \times 5 \quad (4)$$

Where, β_{CT} , β_{CC} and β_{KM} are coefficient value of Comment Type, Comment Category and Keyword Matching.

Table 4. Rubric for Feature Evaluation in Automated Grading

Feature	Level	Description	Threshold	Coefficient (β)
Comment Type (CT)	Basic	Short response	1-10 words	1
	Medium	Moderately detailed response	11-20 words	2
	Detailed	Highly detailed response	21+ words	3
Comment Category (CC)	None	No relevant content	Not applicable	0
	Error Only	Focus on identifying errors	Describes error only	1
	Solution Only	Focus on providing solutions	Describes solution only	2
	Both Error and Solution	Covers both aspects	Describes both error and solution	3
Keyword Matching (KM)	None	No keyword match	0% matching	0
	Low	Few keywords match	1-30% matching	1
	Medium	Moderate keyword match	31-70% matching	2
	High	High keyword match	71-100% matching	3

3.5. Cosine similarity Module

To further refine the evaluation, cosine similarity was calculated using the SBERT model from the sentence transformer library. This model, all-MiniLM-L6-v2, was employed to measure the similarity between student responses and the teacher's reference answer by adding another layer of assessment to the autograding system.

Cosine Similarity functions take two sentences or word vectors and return their similarities. It is calculated using Equation (5).

$$CS = \frac{A \cdot B}{\|A\| \|B\|} \quad (5)$$

In Equation (5), SBERT embedding for student and teacher responses is represented by A and B, respectively. The thresholds employed for cosine similarity are enumerated in Table 5.

Table 5. Threshold Value for Cosine Similarity

Feature	Level	Description	Threshold	Coefficient (β)
Cosine Similarity (CS)	Low	Low semantic similarity	0.0 - 0.3	1
	Medium	Moderate semantic similarity	0.31 - 0.7	2
	High	High semantic similarity	0.71 - 1.0	3

3.6. Model Development

In the process of developing a model for student grades using their answers as input data we use an ensemble stacking model that merges various machine learning algorithms to boost predictive accuracy. We opt for the stacking method because it combines the traits of multiple models leading to better accuracy and reliability overall. This section outlines the procedures involved in building, training and assessing the model along with a comparison of its performance, with that of individual models.

3.6.1 Feature Selection

The model utilized several key features for training, including Comment Type (CT), Comment Category (CC), Keyword Matching (KM), Cosine Similarity (CS). $Final_{Rubric_Mark}$ is consider as label. To assure standardization across data inputs, each features underwent appropriate normalization and scaling.

3.6.2 Stacking Ensemble Model

The stacking model includes a group of base learners and a meta student. Base learners are the models used in machine learning. Examples of these base learners include Linear Regression models and Decision Trees along, with Support Vector Machines (known as SVM). Every model gathers information, from the input characteristics to make its unique prediction. Meta Learner refers to a model that uses the predictions made by the base learners to produce a prediction outcome.

In research studies and analysis tasks a Gradient Boost Machine (GBMachine), known for its ability to capture patterns effectively is employed as the meta learning model. The prediction made by the base learners are as follows;

Let the predictions from the base learners be denoted as in Equation (6):

$$\hat{y}_1 = f_1(X), \hat{y}_2 = f_2(X), \dots \dots \hat{y}_n = f_n(X) \quad (6)$$

In Equation (6), X stands for the input features (CT, CC, KM, and CS), and \hat{y}_i represents the prediction from the i-th base learner.

The meta-learner then combines these predictions is depicted in Equation (7).

$$\hat{y}_{final} = g(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n) \quad (7)$$

In Equation (7), $g(\cdot)$ represent the function that the meta-learning system has acquired. This function provides the students predicted grade.

3.6.3 Model Training and Hyper Parameter Tuning

To ensure model training on diverse examples and evaluation on unseen data, the dataset was divided into training and testing subsets using a 70-30 split. The 70% of data (training set) was used for model training and hyperparameter tuning, while remaining 30% data (testing set) was reserved for evaluation. The stacking ensemble's foundation

comprised Linear Regression, Decision Tree Regressor, and Support Vector Machine with a linear kernel as base models. These were chosen for their varying complexity and interpretability levels, providing a solid basis for the stacking approach.

The stacking ensemble utilized a Gradient Boosting Regressor as its meta-learner. To enhance this meta-learner's performance, GridSearchCV with 10-fold cross-validation was employed on training set for hyperparameter optimization. The 10-fold cross-validation was part of training phase only. The tuning process focused on adjusting the learning rate, number of estimators, and maximum tree depth in the Gradient Boosting Regressor, aiming to achieve optimal model performance.

The model's training incorporated 10-fold cross-validation to ensure a robust learning process. The training goal was to minimize the mean squared error (MSE) between the predicted and actual rubric-based marks. The loss function used during training is depicted in Equation (8)

$$MSE = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2 \quad (8)$$

where N is the number of samples, \hat{y}_i is the predicted mark for the i-th sample, and y_i is the actual rubric-based mark.

3.6.4. Evaluation and Comparison

The evaluation phase is important in that it determines the effectiveness of the trained model through metrics and visual displays. The final mark prediction by the ensemble-stacking model incorporates the features used in the rubric-based scoring and adjusts them using the stacking model's learned weights in Equation (9).

$$\hat{y}_{ML} = Scaling_{Factor} \times \left(\frac{\beta_{CT}}{3} \times 5 + \frac{\beta_{CC}}{3} \times 5 + \frac{\beta_{KM}}{3} \times 5 + \frac{\beta_{CS}}{3} \times 5 \right) \quad (9)$$

where \hat{y}_{ML} is the predicted mark by the machine learning model β_{CT} , β_{CC} , β_{KM} , β_{CS} are the coefficients for the respective features.

To selecting appropriate evaluation, metrics is critical to assess the model's performance accurately. These metrics offer a quantitative foundation for comparing stacked model with other regression models and for ensuring the robustness and reliability of our automated grading system. The following formulations and the rationale for their selection. Furthermore, to evaluate the robustness and transferability of our model's results, we implement cross-validation techniques.

3.6.4.1 Cross-Validation

In machine learning, cross-validation is a statistical technique employed to evaluate model performance. This method offers a dependable approach to determine how well a statistical analysis will apply to an independent dataset. Our research utilizes k-fold cross-validation, which divides the data into k subsets. The model is trained on k-1 subsets and tested on the remaining one. This procedure is repeated k times, ensuring each subset serves as the validation set once. We implemented this technique on 70% of training dataset to find the best hypermeters for the model. To evaluate model we use testing dataset, remaining 30% of final data, which was not involved in cross-validation process. It is used to confirm the model's reliability and ability to generalize. The resulting cross-validation score indicates the model's effectiveness across various data subsets.

Root Means Square Error (RMSE): It is derived by extracting the square root of the Mean Squared Error (MSE) (See Equation (10)). This measure provides an error estimate in the same units as the predicted variable. RMSE is an essential metric or assessing the accuracy of model predictions. The objective is to reduce RMSE as much as possible,

thereby ensuring that the model's predictions correspond as accurately as feasible to the actual values (see Equation (10)).

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2} \quad (10)$$

where R-squared (R^2): It measures the extent to which the variation in the dependent variable can be explained by the independent variables. Also represents the fraction of the dependent variable's variance that is predictable based on the independent variables. R^2 Calculate using formula shown in Equation (11).

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \bar{y})^2}{\sum_{i=1}^N (y_i - \bar{y})^2} \quad (11)$$

In Equation (11), \bar{y} is the mean of the true marks. In our study, R^2 Score shows how well the model fits the data. This metric helps us understand how good the model is at explaining the variances in the data, showing how well it captures the patterns behind student performance. A high R^2 score suggest that the DebugProGrade model does a good job of explaining a large part of the variation in student grades.

4. RESULT AND DISCUSSION

The experiment was set up using a Python notebook on Google Colab's web-based platform, employing a computing environment with 12 GB of RAM and over 100 GB of HDD storage. The experiment did not employ a GPU to ensure the results reflect a more standard computational environment.

For this study, a pre-trained SBERT model from Hugging Face was utilized, which effectively captures semantic similarity in textual data. The corpus, consisting of student responses and reference answers, was divided into a training and testing split of 80:20. The training data was employed to calculate initial similarity scores using cosine similarity and train the machine learning models using hyperparameter tuning developed for the DebugProGrade system.

4.1. Hyperparameter Tuning:

Parameter tuning is a critical step in optimizing the performance of machine learning models. Table 6 lists the parameters used in the hyperparameter tuning process for the Gradient Boosting Regressor, along with descriptions and the values that were explored during the tuning process.

Table 6. Hyperparameter Tuning

Parameter	Description	Values Explored
learning_rate	The learning rate shrinks the contribution of each tree in the Gradient Boosting model	0.01, 0.1, 0.2
n_estimators	The number of boosting stages (i.e., trees) to be run.	100, 200, 300
max_depth	Sets the maximum depth of each tree	3, 5, 7
Cv	The number of folds in cross-validation.	10

Using a GridSearchCV, we methodically investigated different combinations of parameters to uncover the optimal settings that result in the top model performance. This meticulous fine-tuning process is essential for building a sturdy and precise stacking model for the automated grading of student programming assignments, guaranteeing that the model accurately evaluates and scores responses.

Table 7 shows the comparison of stacked model with other regression model. The Stacked Model demonstrates impressive performance, following closely behind the Decision Tree in terms of R^2 and displaying slightly higher error rates. Compared to the Linear Regression and Support Vector Machine models, both the Stacked Model and Decision Tree showcase superior performance, emphasizing the advantages of ensemble techniques in enhancing the accuracy of predictions for the autograding system.

Table 7. Model Comparison With Evaluation Metrics

MODEL	MSE	MAE	R2
Linear Regression	0.331628	0.491522	0.991019
Decision Tree	0.007812	0.007812	0.999788
Support Vector	0.378022	0.446137	0.989763
Stacked Model	0.025107	0.031335	0.99932

The histogram in Figure 2 shows the distribution of differences between rubric-assigned marks and machine learning (ML) predicted marks. The data largely clusters around a difference of 0, indicating that the ML model generally aligns well with the rubric-based grading. However, the range of differences, spanning from -20 to +20, reveals occasional significant discrepancies, where the ML model either overestimates or underestimates marks. These outliers suggest areas where the model could be further refined. Overall, while the ML model effectively replicates human grading in most cases, the presence of outliers highlights the need for continued model improvement.

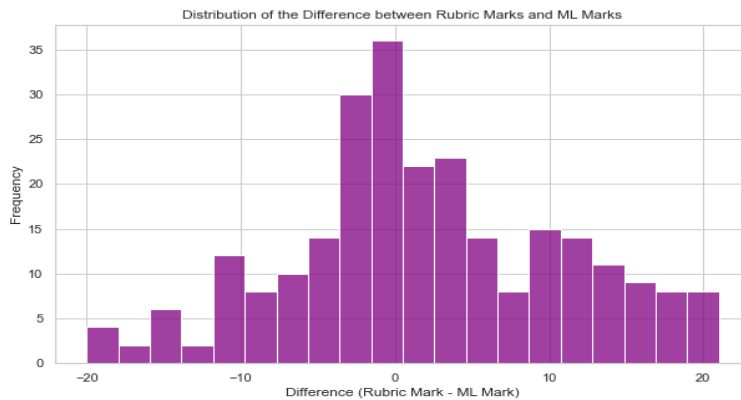


Figure 2. Distribution of Difference Between Rubric Based Marks and Predicted Mark

Figure 3 illustrates that a considerable portion of students exhibit low levels of debugging proficiency, with the "Very Poor" category having the highest concentration, followed by "Average" and "Poor." This suggests that most students face difficulties in effectively detecting and resolving programming errors. On the other hand, a smaller number of students fall into the "Very Good" and "Good" categories, indicating that only a limited group possesses strong debugging skills. This distribution highlights the importance of implementing targeted interventions to improve debugging abilities among the student population.

4.2. Discussion

The findings of this study emphasize the usefulness of the DebugProGrade system in offering a precise and dependable grading of student programming assignments. The integration of advanced machine learning approaches, such as a pre-trained SBERT model, and ensemble methods, such as the Stacked Model, has exhibited strong correspondence with human grading practices. The substantial R^2 values recorded across the models, particularly in the decision tree

and stacked models, suggest that these models are capable of capturing the intricate details of student responses with remarkable accuracy.

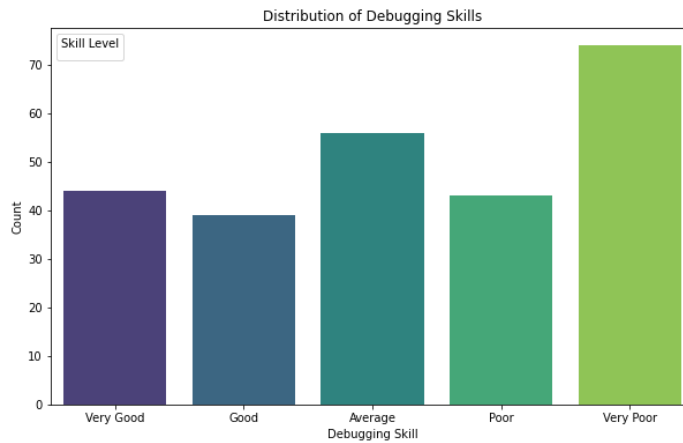


Figure 3. Distribution of Debugging Skill From "Very Good" to "Very Poor"

It is possible to explain the differences between the scores given by the rubric and the scores estimated by the machine-learning system by the fact that they are intrinsically work. This shows that there are recurring issues through outliers, where the width of shading over estimate or under estimate some facets of the student responses. These errors likely occur due to gaps in the SBERT embedding insofar as student comments are concerned. This means that enhanced future feature variable could be incorporated to enhance the model accuracy of the model or more advanced NLP approaches may be used.

The study shows that changes in education's pedagogy require to be made to adapt to students' difficulties in programming courses. The work done here shows that most students are "Very Poor" in debugging skills and indicate a significant shortfall in skills essential for programming achievement. This might have been due to inadequate material or resource to further develop the skill of the students in this area. Consequent to these research outcomes, it will be unambiguous that teachers should come up with particularized approaches for students to improve in aspects of debugging skill. For instance, the manipulation of more external feedback involving structured debugging activities and feedback that is specific to particular students has potential for development and can take students from lower levels of skills to higher levels.

According to the findings of the research, incorporating more such tools such as DebugProGrade into programming education has several benefits. It eliminates grader's time in assessing assignments, and the risk of bias when grading, while giving accurate and fair results. Also, the recommendation feature of DebugProGrade allows teachers to understanding skill gaps that allows students get the right support needed at the right time hence better learning outcomes.

DebugProGrade has demonstrated how effectively its emulating human grading practices. This study investigate several area for potential enhancement. Improve the model's ability to assess the student's debugging skills and resolving the observed differences may further improve the effectiveness of the system for both learn and teachers in learning computer programming. As a result, for further improvements of the system, future research should consider the addition of other data sources proactively collected from the students and the instructors in real-time.

5. CONCLUSION

DebugProGrade is an autograder, which differs from traditional grading systems in that it, comprises semantic analysis and keyword extraction to score students submissions of programming assignments. This is unlike traditional grading tools that provide mainly inspections of syntax or keywords and uses much more advanced deep learning technique, specifically SBERT embedding, which really grasp the more complex meanings conveyed by student responses. It

has captured the complex relationships between what students write and the correct solutions. This ability is a unique advantage that DebugProGrade has over other autograding systems because it can provide an assessment of a student's debugging skills. While the solution is going to show you how many correct solutions you found, but not how well you found errors or how many errors you found and how you suggested fixing them. DebugProGrade can fine-tune its settings to align with human grading: that is how explanations and approaches to problem solving are evaluated. When it does, it provides helpful clues to the instructors regarding where students have problems. The platform makes it easy to use AI to automate the grading process, correcting the accuracy as well as fairness of assessments. It tackles the most crucial skills critical to programming—semantic understanding and debugging—and ultimately gives students the opportunity to improve faster through more relevant, precise feedback. In short, DebugProGrade's aggressive features make grading that much thicker, more reliable and more valuable for improving student learning.

ACKNOWLEDGEMENT

The authors wish to thank the anonymous reviewers for their constructive feedback.

FUNDING STATEMENT

The authors received no funding from any party for the research and publication of this article.

AUTHOR CONTRIBUTIONS

Amit Patel: Conceptualization, Methodology, Validation, Writing – Original Draft Preparation;
Hardik Joshi: Project Administration, Writing – Review & Editing;

CONFLICT OF INTERESTS

No conflict of interests were disclosed.

ETHICS STATEMENTS

Our publication ethics follow The Committee of Publication Ethics (COPE) guideline.



REFERENCES

- [1] H. Vimalaraj et al., "Automated Programming Assignment Marking Tool," 2022, pp. 1–8, doi: 10.1109/I2CT54291.2022.9824339.
- [2] M. Messer, Automated Grading and Feedback of Programming Assignments, vol. 2, no. 1. Association for Computing Machinery, 2022.
- [3] B. P. Cipriano, N. Fachada, and P. Alves, "Drop Project: An automatic assessment tool for programming assignments," *SoftwareX*, vol. 18, p. 101079, 2022, doi: 10.1016/j.softx.2022.101079.
- [4] X. Liu, S. Wang, P. Wang, and D. Wu, "Automatic grading of programming assignments: An approach based on formal semantics," *Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Softw. Eng. Educ. Training, ICSE-SEET 2019*, pp. 126–137, 2019, doi: 10.1109/ICSE-SEET.2019.00022.
- [5] A. Singh, S. Karayev, K. Gutowski, and P. Abbeel, "Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work," *L@S 2017 - Proc. 4th ACM Conf. Learn. Scale*, pp. 81–88, 2017, doi: 10.1145/3051457.3051466.

- [6] D. M. Souza, K. R. Felizardo, and E. F. Barbosa, "A systematic literature review of assessment tools for programming assignments," Proc. - 2016 IEEE 29th Conf. Softw. Eng. Educ. Training, CSEEandT 2016, pp. 147–156, 2016, doi: 10.1109/CSEET.2016.48.
- [7] N. Süzen, A. N. Gorban, J. Levesley, and E. M. Mirkes, "Automatic short answer grading and feedback using text mining methods," Procedia Comput. Sci., vol. 169, no. 2019, pp. 726–743, 2020, doi: 10.1016/j.procs.2020.02.171.
- [8] Y. Kumar, S. Aggarwal, D. Mahata, R. R. Shah, P. Kumaraguru, and R. Zimmermann, "Get it scored using autosas - An automated system for scoring short answers," 33rd AAAI Conf. Artif. Intell. AAAI 2019, 31st Innov. Appl. Artif. Intell. Conf. IAAI 2019 9th AAAI Symp. Educ. Adv. Artif. Intell. EAAI 2019, no. Higgins 2014, pp. 9662–9669, 2019, doi: 10.1609/aaai.v33i01.33019662.
- [9] R. Siddiqi, C. J. Harrison, and R. Siddiqi, "Improving teaching and learning through automated short-answer marking," IEEE Trans. Learn. Technol., vol. 3, no. 3, pp. 237–249, 2010, doi: 10.1109/TLT.2010.4.
- [10] A. Condor, M. Litster, and Z. Pardos, "Automatic short answer grading with SBERT on out-of-sample questions," Proc. 14th Int. Conf. Educ. Data Mining, EDM 2021, pp. 345–352, 2021.
- [11] S. Haller, A. Aldea, C. Seifert, and N. Strisciuglio, "Survey on Automated Short Answer Grading with Deep Learning: from Word Embeddings to Transformers," vol. 1, no. 1, 2022, [Online]. Available: <http://arxiv.org/abs/2204.03503>.
- [12] X. Sun et al., "Sentence Similarity Based on Contexts," Trans. Assoc. Comput. Linguist., vol. 10, pp. 573–588, 2022, doi: 10.1162/tacl_a_00477.
- [13] I. G. Ndukwe, C. E. Amadi, L. M. Nkomo, and B. K. Daniel, Automatic Grading System Using Sentence-BERT Network, vol. 12164 LNAI. Springer International Publishing, 2020.
- [14] N. Ayewah, D. Hovemeyer, D. J. Morgenthaler, J. Penix, and W. Pugh, "Using static analysis to find bugs," IEEE Softw., vol. 25, no. 5, pp. 22–29, 2008, doi: 10.1109/MS.2008.130.
- [15] U. von Matt, "Kassandra: the automatic grading system," *SIGCUE Outlook*, vol. 22, no. 1, pp. 26–40, Jan. 1994, doi: 10.1145/182107.182101.
- [16] G. Singh, S. Srikant, and V. Aggarwal, "Question independent grading using machine learning: The case of computer program grading," Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., vol. 13-17-August-2016, pp. 263–272, 2016, doi: 10.1145/2939672.2939696.
- [17] H. Nguyen, M. Lim, S. Moore, E. Nyberg, M. Sakr, and J. Stamper, Exploring metrics for the analysis of code submissions in an introductory data science course, vol. 1, no. 1. Association for Computing Machinery, 2021.
- [18] M. Jukiewicz, "The future of grading programming assignments in education: The role of ChatGPT in automating the assessment and feedback process," Think. Ski. Creat., vol. 52, no. July 2023, p. 101522, 2024, doi: 10.1016/j.tsc.2024.101522.
- [19] H. Cheers, Y. Lin, and S. P. Smith, "Academic source code plagiarism detection by measuring program behavioral similarity," IEEE Access, vol. 9, pp. 50391–50412, 2021, doi: 10.1109/ACCESS.2021.3069367.
- [20] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated Grading and Feedback Tools for Programming Education: A Systematic Review," ACM Trans. Comput. Educ., vol. 24, no. 1, 2024, doi: 10.1145/3636515.
- [21] A. L. C. Barczak, A. Mathrani, B. Han, and N. H. Reyes, "Automated assessment system for programming courses: a case study for teaching data structures and algorithms," Educ. Technol. Res. Dev., vol. 71, no. 6, pp. 2365–2388, 2023, doi: 10.1007/s11423-023-10277-2.

- [22] A. Kumar, A. Walter, and P. Manolios, “Automated Grading of Automata with ACL2s,” *Electron. Proc. Theor. Comput. Sci. EPTCS*, vol. 375, pp. 77–91, 2023, doi: 10.4204/EPTCS.375.7.
- [23] M. Novak and D. Kermek, “Assessment Automation of Complex Student Programming Assignments,” *Educ. Sci.*, vol. 14, no. 1, 2024, doi: 10.3390/educsci14010054.
- [24] N. Denissov, L. M. Advisor, J. Sorva, and T. In, “Creating an educational plugin to support online programming learning A case of IntelliJ IDEA plugin for A+ Learning Management System Title: Creating an educational plugin to support online programming learning A case of IntelliJ IDEA plugin for A+ Lea,” 2021.
- [25] A. Shah, “Web-cat: A web-based center for automated testing,” *ACM J. Educ. Resour. Comput.*, vol. 3, no. 3, pp. 1–24, 2003, [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.1522&rep=rep1&type=pdf>.
- [26] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, “DeepCode: Feedback codes via deep learning,” *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 194–206, 2020, doi: 10.1109/JSAIT.2020.2986752.
- [27] R. S. Pettit, J. D. Homer, K. M. Holcomb, N. Simone, and S. A. Mengel, “Are automated assessment tools helpful in programming courses?,” *ASEE Annu. Conf. Expo. Conf. Proc.*, vol. 122nd ASEE Annual Conference and Exposition: Making Value for Society, no. 122nd ASEE Annual Conference and Exposition: Making Value for Society, 2015, doi: 10.18260/p.23569.
- [28] S. Parihar, “Automated Grading Tool for Introductory Programming,” IIT, KANPUR, 2015.
- [29] X. Hu, Z. Cai, M. Louwse, A. Olney, P. Penumatsa, and A. Graesser, “A revised algorithm for latent semantic analysis,” *IJCAI Int. Jt. Conf. Artif. Intell.*, pp. 1489–1491, 2003.

BIOGRAPHIES OF AUTHORS

	<p>Amit Patel is currently pursuing a Ph.D. degree at Gujarat University. He received his Master's degree in Computer Applications from Veer Narmad South Gujarat University, Surat, Gujarat. He has 12 years of experience, working as an Assistant Professor in the Bachelor of Computer Science department at Vidyabharti Trust College of Business, Computer Science, and Research, Umrakh, Gujarat. His research interests include Information Retrieval, Natural Language Processing, Data Mining and Machine Learning.s</p>
	<p>Hardik Joshi received his Ph.D. degree in Computer Science from Gujarat University, Ahmedabad, Gujarat. He is currently working as an Associate Professor at Gujarat University. His research interests are in Natural Language Processing and Data Mining.</p>