Auto-assessing Java Code with CodeT5+, Early Fusion and CatBoost

Pranav Gokhale
Dept. of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22143@bl.students.amrita.edu

Singadi Srujan Reddy
Dept. of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22151@bl.students.amrita.edu

Viraj Kisan Daule
Dept. of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22162@bl.students.amrita.edu

Nakka Narmada
Dept. of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.p2dsc21018@bl.amrita.edu

Dr. Peeta Basa Pati
Dept. of Computer Science & Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
ORC ID: 0000-0003-2376-4591

***Abstract*—With a rise in students pursuing education, the workload on teachers and professors has increased and especially during evaluation period. Taking in consideration of the evaluation of a code written by a student, is a very tedious task. By using advanced embedding models like Code T5+, components are made which are processed into embeddings that capture the semantic meaning and structure of the code. Dimensionality reduction techniques, such as Principal Component Analysis (PCA) and reduction based in Information Gain(IG), are applied to reduce the size of the embeddings, followed by scaling and fusion methods to further enhance the quality of the feature representation. A regression model is then trained on these embeddings to predict the quality of the code. This approach aims to provide an automated and scalable solution for large-scale grading systems. The model's performance is validated by achieving an R^2 score of 0.56, which demonstrates an improvement over previous methods and highlights its potential for real-world educational applications.**

***Index Terms*—Automated Code Evaluation, Weighted Regression Model, Java, Principal Component Analysis(PCA), Model Training, R^2 score, CatBoost, Fusion Technique, Explainable AI**

I. Introduction

The curriculum of education demands evaluation through tests and quizzes.Though it is a good practice to test the knowledge of a student, it carries its bad consequence too. One of which includes code evaluation by the teacher. Evaluating student-submitted code has become increasingly challenging, especially with the rise of online learning platforms and larger class sizes. Traditional methods of manual grading, while effective in small-scale scenarios, are labor-intensive, time-consuming, and often prone to inconsistencies due to subjective biases. The difficulty is inevitable as a code can be written in multiple ways, using different algorithms. In a code, there are multiple parts which contains different weightage. It is also important for a teacher to consider weightage of the code snippet too. These challenges highlight the pressing need for automated grading systems that can ensure fairness, accuracy, and efficiency. Such systems not only save educators valuable time but also provide students with timely and consistent feedback, improving the overall learning experience.

This study introduces a machine learning (ML)-based framework for automating the evaluation of student code submissions. The approach focuses on assessing the logical correctness and adherence to key programming principles within the code. Central to this framework is the use of CodeT5+ embeddings, which capture the semantic and structural information in the code. To further refine the model's performance, techniques like Principal Component Analysis (PCA) and Information Gain (IG) are applied to optimize feature representation and performing dimensionality reduction. Further, when a desired dataset is formed, it is divided into training and testing dataset in the ratio of 4:1 . A regression model is implemented over this numerical dataset. Weights are assigned to different features of the dataset according to their importance in the grading criteria, hence, making it a weighting regression model. The assigned weights are updated using hyperparameter tuning, to improve the results.

The final outcome comes out to be a numerical integer value between 0-10. This depicts the score of the student out of 10 marks. The model works with high precision and accuracy. Enabling this design in colleges and universities will help in a seamless and objective evaluation process. The leveraging of an advanced ML technique in the study, helps to addresses the growing demand for scalable and reliable code evaluation systems, benefiting educators by reducing manual workload and students by delivering prompt, accurate feedback to foster their learning.

II. Literature Survey

Zhang *et al.* [1] effectively uses weighted SVR models with differential evolution for electricity load forecasting, but it does not explore their application in software code evaluation, which is the primary focus of our project. This gap highlights the need for applying weighted regression techniques specifi-

cally to code quality assessment, an area our project aims to address.

Silhavy *et al.* [2] highlights the importance of use case points (UCP) variables and model complexity in software size estimation, with Model D outperforming simpler models and the UCP method by significantly reducing errors. Additionally, it discusses advancements in handling imbalanced data with logistic regression, introducing Rare Event Weighted Logistic Regression.

Maalouf *et al.* [3] (RE-WLR) improves classification accuracy for large, imbalanced datasets, particularly in predicting rare events, at similar processing speeds to traditional methods. These insights demonstrate the significance of weighted regression techniques in enhancing prediction accuracy across various domains.

Azeem *et al.* [4] indicate suboptimal design choices in source code, often leading to increased change- and fault-proneness. Despite the existence of numerous detectors, previous research highlights significant limitations: developer subjectivity, low agreement between tools, and challenges in setting appropriate detection thresholds. Fontana *et al.* [5] explores the use of machine learning techniques to address these issues and enhance the accuracy and usability of code smell detection.

Messe *et al.* [6] investigates classifying code smell severity using machine learning, focusing on ordinal classification methods. While severity classification accuracy is lower than binary classification, the best models achieve a high ranking correlation of 0.88–0.96, as measured by Spearmans . S. *et al.* [7] includes an analysis of auto-grading C programming assignments using deep learning approaches like CNNs, LSTMs, and CodeBERT. The method transforms code into vectors and achieves a root mean squared error (RMSE) of 1.89, demonstrating its effectiveness compared to traditional statistical methods.

D. *et al.* [8] utilizes pre-trained language models for automated grading of "C" programming assignments to address the tedious and error-prone nature of manual grading. By using embeddings from different transformers as feature vectors for training various regressors, it is found that the T5 model combined with the CatBoost regressor achieves the lowest error, around 15%, as measured by root mean squared error (RMSE).

Souri *et al.* [9] explores weighted logistic regression, a technique designed to address challenges associated with imbalanced datasets and reject inference. It builds on foundational logistic regression and generalized linear models by integrating weights to handle uneven data. Applications in credit scoring illustrate its practical utility, while theoretical advancements underscore the importance of weights and feature selection in refining model accuracy and interpretability.

Jadhav *et al.* [10]highlights the partial evaluation approach, which converts high-level programs into optimized low-level equivalents to uncover and exploit parallelism in numerical computations. A prototype compiler has shown performance gains of 7x to 91x over traditional compilation methods. By combining partial evaluation with parallel scheduling, it effectively leverages low-level parallelism in pipelined and parallel architectures, demonstrated through applications like simulating a nine-body solar system.

Chouchen *et al.* [11] introduces introduces Modern Code Review Anti-patterns (MCRA), highlighting issues in Modern Code Review (MCR) that can harm software quality, slow integration, and degrade project sustainability and review culture. Analyzing 100 OpenStack code reviews, it identifies the causes, effects, symptoms, and frequent co-occurrence of these anti-patterns.

Sushmitha Ramaneedi *et al.* [12] highlight that textual errors arising from typing, language skills, or recognition tools hinder processing, particularly in Indian languages. Their study demonstrates that an enhanced mT5 model with transfer learning reduces Kannada text noise by 12% at a 25% Character Error Rate (CER). Vaan Amuthu Elango *et al.* [13] tackles errors in Tamil text caused by misspellings, typos, illiteracy, and text from images or voice. Using mT5, a multilingual transformer model fine-tuned on a Tamil dataset, it achieves a 97.7% reduction in CER and 89.3% reduction in WER.

Shuo Ren *et al.* [14] highlights the limitations of BLEU and accuracy in code synthesis for ignoring code syntax and semantics. To address this, it introduces CodeBLEU, combining BLEU's n-gram matching with AST-based syntax and data-flow semantics. Experiments show CodeBLEU better correlates with programmer-assigned quality scores in text-to-code, code translation, and code refinement tasks.

Muhammad Ilyas Azeem *et al.* [15] reviews 15 studies (2000–2017) on using machine learning for code smell detection. It highlights targeted code smells, model configurations, evaluation methods, and performance, identifying Decision Trees and Random Forest as the most effective classifiers. The study also outlines challenges and suggests improvements for future approaches.

Shiqi Wang *et al.* [16] presents ReCode, a benchmark for evaluating code generation model robustness against over 30 semantic-preserving prompt transformations. Using execution-based metrics, it tests models on HumanEval and MBPP, revealing CodeGen's robustness, sensitivity to syntax changes, and MBPP's stricter robustness challenges.

Pietro Liguori *et al.* [17] explores robustness in Neural Machine Translation (NMT) for code generation by proposing tailored perturbations and metrics for evaluation. Preliminary results reveal which perturbations most impact model performance, offering insights for improvement.

Till Köster *et al.* [18] leverages code generation to create efficient, model-specific simulators for domain-specific languages, combining succinct modeling with high-performance execution. Using biochemical reaction networks as a case study, the specialized simulator outperforms state-of-the-art algorithms by over an order of magnitude. Both the generic and specialized simulators are open-source and bench-marked extensively.

This paper is a continuation of the following works:

- Nakka Narmada *et al.* [19] proposes automating the grading of ”C” programming assignments using pre-trained language model embeddings and regressors. Evaluated with RMSE, the T5 model embeddings with a CatBoost regressor achieve the lowest error, approximately 15%.
- R. V. Muddaluru *et al.* [20] uses machine and deep learning techniques, including regression, CNNs, and LSTMs, for auto-grading C programming assignments, addressing challenges in manual grading. CodeBERT transforms code into vectors for model input, achieving an RMSE of 1.89, demonstrating the approach's feasibility and efficiency.

III. Data Description

The data used in this research consists of 1232 rows of Code Snippets of Java programs solving Undergraduate-level problems, such as different pattern printing, concepts of arrays like merging arrays, finding max and min element in an array etc. The target attribute is marks awarded to the students for their codes by a subject matter expert. The objective is to compare and analyze the performance of different regression models by dividing the student submitted code into different parts.

A. Data Creation

The data is manually collected from the evaluations conducted for Java codes in colleges. 1232 codes are collected and put under one feature of ”Student Submitted Code”. Other column includes the correct code generated under ”Correct Code” column by the subject expert. The last feature contains marks allotted to that code out of 10 marks under the feature name ”Final Marks”.

B. Data Exploration

The dataset used in this research consists of Java programs collected from the students. The dataset including 1232 entries, are defined by three columns: (i) ”Student Submitted Code” column, (ii) the ”Correct Code”, written by a Subject - Matter expert, which shows the code which would be assigned complete marks and (iii) the ”Final Marks” column which is the score of the student submitted code in comparison with the correct code. There were no null values found as the marks were assigned manually. The distribution of Marks awarded to code snippets is shown in Fig. 1.

C. System Architecture

The student-submitted Java code was first categorized into several distinct groups based on its structural components. These categories were designed to capture key syntactic and semantic features of the code. The categories included:

- Basic Syntax Elements: Keywords and operators that form the building blocks of Java code, such as ”include”, ”for”, ”:”, ”(”, ”)” etc.
- Control Flow Structures: Structures that control the flow of execution in the program, including loops and conditional statements.

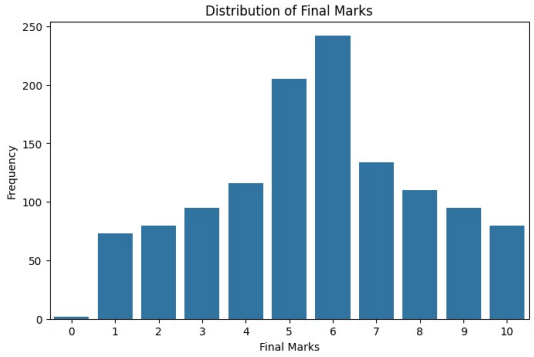

Fig. 1. Distribution of marks in dataset

- Method Declarations: Method signatures, including return types, method names, and parameters.
- Class and Object Instantiation: Object creation, focusing on the use of declaration of classes.
- Complex Expressions: Complex expressions within control flow structures such as an expressions including function call, recursion, mathematical calculation, etc.
- Libraries and Imports: External Java libraries imported for use in the code.
- Custom Data Structures: User-defined classes and custom data structures within the code.

Once these categories are identified, they are consolidated into two upper attributes: ”Keywords” and ”Code Logic”. The ”Keywords” attribute combines all relevant code components, such as syntax elements, control flow structures, method declarations, and complex expressions. The ”Code Logic” attribute specifically includes user-defined classes and object instantiations, which are the components of Custom Data Structures. These attributes are then used as input for CodeT5+, a pre-trained advanced version of the CodeT5 model which is tailored for generating embeddings of code snippets while capturing both the semantic and syntactic features. The model generates vector spaces for each attribute. This approach allows to differentiate parts of the code as different attributes, thereby, allowing to scale it based on its influence on the model performance. Fig. 2 shows flow of attribute extraction by application of regex/ string manipulation, while Table 1 shows extraction with an example code snippet.

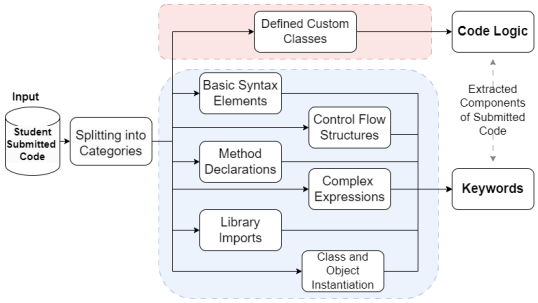

Fig. 2. Data splitting

TABLE I
CODE EXAMPLE WITH KEYWORDS AND CUSTOM DATA STRUCTURES

Category	Details

Sample Code	`import java.util.ArrayList;`
	`class CustomClass {`
	`  int id;`
	`  String name;`
	`  public CustomClass(int id, String name) {`
	`  this.id = id;`
	`  this.name = name;`
	`  }`
	`}`
	`public class MainClass {`
	`  public static void main(String[] args) {`
	`  CustomClass obj = new CustomClass(1, "Example");`
	`  if (obj.id > 0) {`
	`  System.out.println(obj.name);`
	`  }`
	`  }`
	`}`

Keywords	int, String, public, =, if, , CustomClass, main,import java.util.ArrayList;
Code Logic	class CustomClass { int id; String name; public CustomClass(int id, String name) { this.id = id; this.name = name; } }

IV. METHODOLOGY

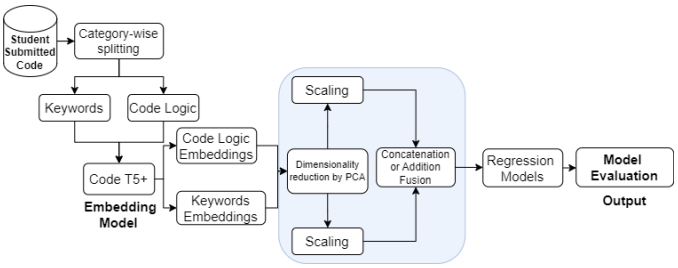

Fig. 3. Flow of Processes

This methodology describes the structured approach employed to analyze and evaluate the influence of various structural and semantic components of Java code on regression models performance for automated scoring of code snippets.The 2 feature derived dataset is processed through CodeT5+, and further optimized using dimensionality reduction methods and then scaled. The processed datasets are then fed into multiple regression models to identify strategies for improving model performance through scaling adjustments and algorithmic optimization.

RQ1: What is the impact of segregation of multiple structural components of code (e.g., library imports, control flow structures) into composite attributes, which can be scaled, on the performance of regression models?

RQ2: How does the choice of fusion technique (concatenation vs. addition) for integrating scaled and reduced structural components of code impact the accuracy and interpretability of regression models?

A. Dimensionality Reduction:

PCA and Information Gain is employed to optimize the feature space and enhance model performance.

Principal Component Analysis (PCA) and Information Gain (IG) are employed together to refine the feature space of the data. PCA reduces the dimensionality of the Keywords and Code Logic attributes, initially represented by 256-dimensional vectors, to 20 components. This step helps retain the most significant patterns while discarding less informative features. In parallel, Information Gain identifies and selects the top 20 features from both attributes based on their contribution to reducing uncertainty in the target variable. Using these methods ensures that the most relevant features are preserved, improving the model's interpretability and predictive performance by reducing noise and enhancing generalization.

B. Model Development

Following dimensionality reduction, the vector spaces were scaled to ensure that the sum of weights, by which scaling is to be done, for each attribute was equal to 1. This scaling strategy was employed to replicate real-life scenarios, where usually subject matter experts assign scores out of 1. Notably, the weight assigned to Keywords attribute was consistently higher than that for Code Logic attribute, reflecting the greater explained variance captured by Keywords compared to Code Logic. This scaling approach aimed to balance the influence in such a way that appropriate attributes have more influence on the model.

The scaled vector spaces were then combined through two fusion techniques: addition and concatenation parallely. In the addition fusion, the resulting vector size was 20, while in the concatenation fusion, the resulting vector size increased to 40. These combinations were used as inputs for the model, representing an early fusion approach, where feature integration occurred prior to model training. Various combinations of weight configurations were tested to assess their impact on the predictive performance of the models.

C. Model Training and Evaluation

A thorough training and assessment are conducted using the fused data—resulting from the reduced and scaled vectors for Keywords and Code Logic—as input to train various regression models. The dataset is split into 80% for training and 20% for testing to evaluate the models effectively. Additionally, 10-fold cross-validation is employed during model training to ensure averaging of metrics across multiple splits of the data. The models tested include Random Forest Regressor, XGBoost, CatBoost, AdaBoost, Bagging Regressor, and Decision Tree Regressor. These models are individually evaluated

based on their performance in predicting the target variable using multiple metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R² Score, and Mean Absolute Percentage Error (MAPE).

For each of the models, hyperparameter tuning is carried out using manual grid search, various hyperparameters such as the early stopping rounds, learning rate and other regularization parameters were considered, during this different combinations of weights are applied to the Keywords and Code Logic attributes, where the sum of weights is constrained to 1.

The models with the best hyperparameters are selected based on their performance across the evaluation metrics, and the results are compared to understand the impact of scaling, weight combinations, and the fusion strategies used on overall model accuracy.

V. Results & Discussion

This section presents findings from experiments with various machine learning regressors applied to processed datasets. To analyze the impact of embedding approaches, embeddings generated by CodeT5+ for extracted Code Logic and Keywords attributes are passed through PCA and Information Gain (IG) reduction. These reduced embeddings are then fused (using addition or concatenation) after scaling. A comparison is made with the performance of directly using embeddings generated by CodeT5+ for student-submitted code as inputs to various regression models given in Table 2.

TABLE II
Model performance for comparison

Metric	CatBoost	XGBoost	Random Forest	Decision Tree	Linear Regression
Training Set					
MAE	0.56	0.16	0.55	0.035	1.84
MSE	0.52	0.09	0.53	0.06	5.29
RMSE	0.72	0.31	0.73	0.26	2.30
MAPE	12.94	3.59	15.55	0.64	57.56
R^2	0.91	0.98	0.90	0.98	0.10
Testing Set					
MAE	1.37	1.42	1.44	1.73	1.95
MSE	3.10	3.65	3.38	5.40	6.07
RMSE	1.76	1.91	1.83	2.32	2.46
MAPE	37.55	39.20	44.08	43.65	61.20
R^2	0.46	0.37	0.42	0.07	-0.04

Hyperparameter tuning is performed as discussed to optimize the performance, making the final weights as 0.1 for "code logic" and 0.9 for the "keywords" features. The resulting configurations for the best performing model, CatBoost, which was able to perform best in all mentioned cases, are presented in the Table III for the different combinations of reduction and fusion techniques. The iterations parameter, set to 2000 (or 1000 for PCA with addition fusion), specifies the number of boosting rounds, ensuring sufficient updates to minimize prediction errors. The learning rate is set to 0.05, providing a balance between convergence speed and stability. The L2 leaf regularization parameter, with values of 3 or 5 depending on the fusion method, controls overfitting by penalizing large leaf weights. The early stopping rounds parameter, fixed at 15, monitors validation performance and halts training if no improvement is observed, thereby preventing overfitting and reducing computational overhead. These tuned hyperparameters are adapted to each reduction technique and fusion method (concatenation and addition) to achieve optimal model performance and efficiency.

TABLE III
Hyperparameters for Different Reduction Techniques and Fusion Methods for CatBoost Regression Model

Reduction Technique	Fusion	Hyperparameters
PCA	Concatenation	'Iterations': 2000, 'Learning Rate': 0.05, 'L2 leaf Regularization': '5', 'Early Stopping Rounds': 15
	Addition	'Iterations': 1000, 'Learning Rate': 0.05, 'L2 leaf Regularization': 3, 'Early Stopping Rounds': 15
Information Gain	Concatenation	'Iterations': 2000, 'Learning Rate': 0.05, 'L2 leaf Regularization': 5, 'Early Stopping Rounds': 15
	Addition	'Iterations': 2000, 'Learning Rate': 0.05, 'L2 leaf Regularization': 5, 'Early Stopping Rounds': 15

The best perforance, CatBoost with the parameters discussed before in Table III, was evaluated based on the following metrics: Coefficient of Determination (R2), Mean Absolute Error (MAE), Mean Square Error (MSE), Root Mean Squared Error (RMSE), , Mean Absolute Percentage Error (MAPE) and as presented in Table IV and Table V for the 2 different reduction techniques and addition and concatenation fusion.

TABLE IV
Performance Metrics with PCA Reduction

Set	Metric	Concatenation Fusion	Addition Fusion
Training Set	R^2	0.91	0.89
	MAE	0.45	0.55
	MSE	0.52	0.64
	RMSE	0.72	0.80
	MAPE	9.38	11.82
Testing Set	R^2	0.56	0.52
	MAE	1.27	1.30
	MSE	2.60	2.88
	RMSE	1.61	1.70
	MAPE	33.71	35.32

TABLE V
Performance Metrics for Fusion Techniques with IG reduction

Set	Metric	Concatenation Fusion	Addition Fusion
Training Set	R^2	0.91	0.91
	MAE	0.44	0.45
	MSE	0.52	0.53
	RMSE	0.72	0.72
	MAPE	9.10	9.2
Testing Set	R^2	0.52	0.51
	MAE	1.31	1.32
	MSE	2.87	2.93
	RMSE	1.69	1.71
	MAPE	36.86	38.23

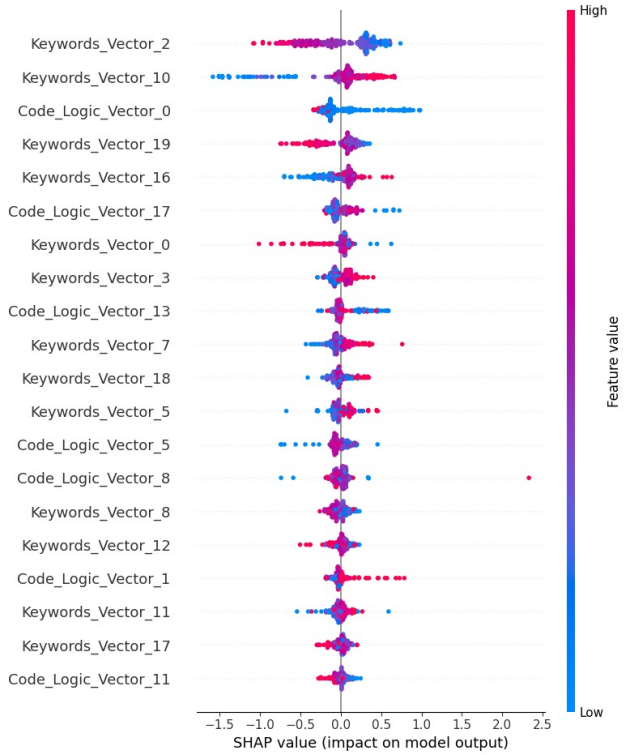

Fig. 4. SHAP Analysis of features with CatBoost for PCA Reduction

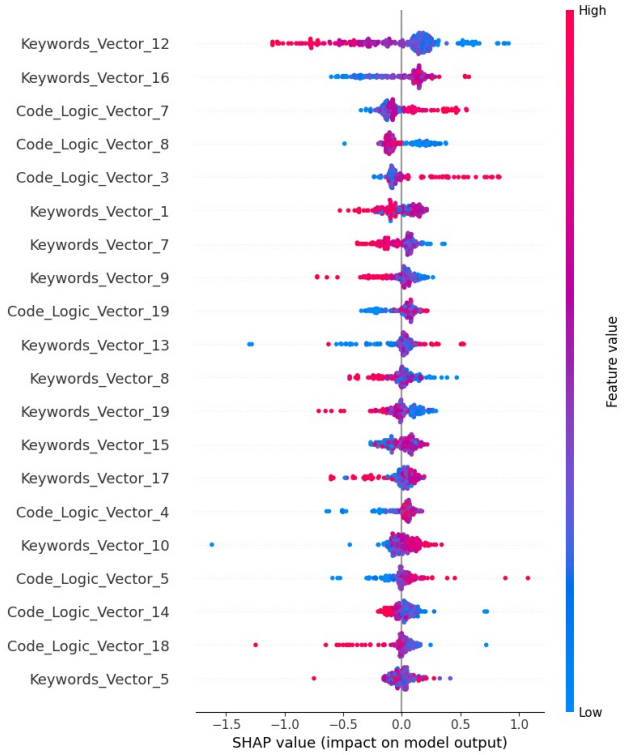

Fig. 5. SHAP Analysis of features with CatBoost for IG Reduction

A. Research question 1(RQ1)

What is the impact of segregation of multiple structural components of code (e.g., library imports, control flow structures) into composite attributes, which can be scaled, on the performance of regression models?

The approach of segregation of multiple structural components of code (e.g., library imports, control flow structures) into composite attributes like keywords and Code Logic, which are scaled by 0.9 for Keywords and 0.1 for Code Logic, showed a significant positive impact on the performance of regression models. As shown in Table II, the traditional approach using unsegregated attributes resulted in suboptimal performance, with the maximum R^2 score of 0.46.

By applying segregation and scaling of the components, we observed a notable improvement in predictive accuracy. For the PCA-reduced dataset (Table IV), the CatBoost model achieved an R^2 score of 0.56 for concatenation fusion and 0.52 for addition fusion, and for the IG-reduced dataset (Table V), the R^2 score result was 0.52 for concatenation fusion and 0.51 for addition fusion, demonstrating the effectiveness of dimensionality reduction combined with the fusion of keywords and code logic features.

Figures 4 and 5 display the SHAP(Shapley Additive Explanations) value visualizations for the two models (PCA and IG reductions) with scaling followed with concatenation fusion, where we can see that keywords consistently had a higher impact on the model's output compared to code logic vectors. This suggests that segregating and scaling keywords results in better predictive performance, while code logic components contribute to the model but to a lesser extent. The SHAP values highlight the impact of individual features, where higher feature values in keywords lead to greater impacts on the final prediction.

In conclusion, the segregation and scaling approach, particularly when applied to keywords, significantly improved the performance of regression models. The R^2 scores, which rose from 0.46 peak value in the traditional method to peak of 0.56 with the proposed approach, validate the effectiveness of this method in enhancing model accuracy and interpretability.

B. Research question 2(RQ2)

How does the choice of fusion technique (concatenation vs. addition) for integrating scaled and reduced structural components of code impact the accuracy and interpretability of regression models?

The choice of fusion technique plays a significant role in both the accuracy and interpretability of regression models. In mentioned approach, concatenation fusion consistently outperformed addition fusion in terms of accuracy, with the R^2 score for concatenation being 0.56 as compared to 0.52 for addition fusion for PCA reduction and 0.52 and 0.51 in IG based reduction, as shown in Tables 4 and 5. This slight performance gain of 4-5% highlights the benefit of preserving the distinct nature of the structural components through concatenation.

In terms of interpretability, concatenation fusion offered better insights, as it kept the vectors separate, allowing for individual analysis of the components using SHAP values,

as presented in Figures 4 and 5. This separation of features allowed a clearer understanding of how keywords and code logic contributed to the final predictions. In contrast, addition fusion combined the features into a single vector space, which reduced the interpretability since the individual impact of each component was not possible to distinguish.

An observation was that addition fusion performed better after scaling the data. The scaling process allowed the individual vectors to be adjusted in magnitude, thereby making them more comparable and improving the fusion process when added. However, when concatenation fusion was applied, scaling had no significant effect on performance, as concatenation preserved the original structure of the vectors, maintaining their distinct contributions regardless of scaling.

Overall, while the performance difference between the two fusion techniques was small (6-7%), concatenation fusion provided a better balance of accuracy and interpretability, particularly for complex analyses like SHAP, as shown in Tables 4 and 5 and Figures 4 and 5. The addition fusion's better performance with scaling suggests that scaling may help align feature magnitudes, but concatenation remains the preferred method for preserving feature separability.

Summarizing the study, the use of Code T5 embeddings for vectorization, scaling of Keywords by 0.9 and Code Logic by 0.1 followed by concatenation using PCA and CatBoost as the running regression model has resulted in more than 200% improvement with respect to the previous paper [19]. The R^2 score for the study is 0.56 while for the paper whose continuation has been done in this study [19] had a R^2 score of 0.27.

VI. Conclusion and Future Scope

The impact of different fusion techniques and dimensionality reduction methods on the performance of regression models was explored. The results demonstrated that concatenation fusion consistently provided better accuracy and interpretability compared to addition fusion. Specifically, concatenation fusion yielded a notable performance improvement over addition fusion. Moreover, concatenation fusion's ability to maintain separate vectors allowed for deeper analysis through SHAP values, further enhancing the interpretability of the model. When comparing models such as CodeT5+, we observed a peak performance of 0.56 for the given setup. This result highlights the potential of integrating various code components into feature representations, but also indicates there is room for improvement.Future work should focus on investigating the effect of scaling on concatenation fusion, as this aspect could further optimize model performance. Additionally, exploring advanced fusion techniques, as well as experimenting with more complex dimensionality reduction methods, could lead to further improvements in both accuracy and interpretability.

VII. Acknowledgment

The authors would like to extend their heartfelt gratitude to Amrita School of Computing for providing the essential infrastructure and support required to carry out this research and produce this publication. The professors and faculty members who guided and assisted throughout the data collection process deserve special acknowledgment. The authors also wish to express their sincere appreciation to the students who generously contributed their time and efforts by submitting the dataset, which played a pivotal role in the success of this research endeavor.

References

[1] F. Zhang, C. Deb, S. E. Lee, J. Yang, and K. W. Shah, "Time series forecasting for building energy consumption using weighted support vector regression with differential evolution optimization technique," vol. 126, pp. 94–103, 2016.

[2] R. Silhavy, P. Silhavy, and Z. Prokopova, "Analysis and selection of a regression model for the use case points method using a stepwise approach," *Journal of Systems and Software*, vol. 125, pp. 1–14, 2017.

[3] M. Maalouf and M. Siddiqi, "Weighted logistic regression for large-scale imbalanced and rare events data," *Knowledge-Based Systems*, vol. 59, pp. 142–148, 2014.

[4] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Information and Software Technology*, vol. 108, pp. 115–138, 2019.

[5] F. Arcelli Fontana and M. Zanoni, "Code smell severity classification using machine learning techniques," *Knowledge-Based Systems*, vol. 128, pp. 43–58, 2017.

[6] 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, feb 2024.

[7] S. P. D. P. B. P. M. R. M. B. Roshan Vasu Muddaluru, Sharvaani Ravikumar Thoguluva, "Auto-grading c programming assignments with codebert and random forest regressor," 2023.

[8] D. P. B. P. Nakka Narmada, "Autograding of programming skills," 2023.

[9] A. Souri, N. J. Navimipour, and A. M. Rahmani, "Formal verification approaches and standards in the cloud computing: A comprehensive and systematic review," *Computer Standards Interfaces*, vol. 58, pp. 1–22, 2018.

[10] A. S. Jadhav and R. M. Sonar, "Evaluating and selecting software packages: A review," *Information and Software Technology*, vol. 51, no. 3, pp. 555–563, 2009.

[11] M. Chouchen *et al.*, "Anti-patterns in modern code review: Symptoms and prevalence," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, (Honolulu, HI, USA), pp. 531–535, IEEE, 2021.

[12] S. Ramaneedi and P. B. Pati, "Kannada textual error correction using t5 model," *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*, pp. 1–5, 2023.

[13] V. A. Elango and P. B. Pati, "Tamil text error correction with multilingual t5 model," in *2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN)*, pp. 1–6, 2023.

[14] S. L. L. Z. S. L. D. T. N. S. M. Z. A. B. S. M. Shuo Ren, Daya Guo, "Codebleu: a method for automatic evaluation of code synthesis," 2020.

[15] L. S. Q. W. Muhammad Ilyas Azeem, Fabio Palomba, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," pp. 115–138, 2019.

[16] H. Q. C. Y. Z. W. M. S. V. K. S. T. B. R. P. B. R. N. M. K. R. D. R. B. X. Shiqi Wang, Zheng Li, "Recode: Robustness evaluation of code generation models," 20 Dec 2022.

[17] S. D. V. R. N. B. C. Pietro Liguori, Cristina Improta and D. Cotroneo, "Can nmt understand me? towards perturbation-based evaluation of nmt models for code generation," in *In Proceedings of the 1st International Workshop on Natural Language-based Software Engineering (NLBSE '22). Association for Computing Machinery, New York, NY, USA, 59–66.*, 2023.

[18] T. Köster, T. Warnke, and A. M. Uhrmacher, "Partial evaluation via code generation for static stochastic reaction network models," in *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '20, (New York, NY, USA), p. 159–170, Association for Computing Machinery, 2020.

[19] N. Narmada and P. B. Pati, "Evaluating unixcoder embeddings for automated grading: A study across varied code perspectives," in *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–5, 2024.

[20] R. V. Muddaluru, S. R. Thoguluva, S. Prabha, P. B. Pati, and R. M. Balakrishnan, "Auto-grading c programming assignments with codebert and random forest regressor," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–6, 2023.