Auto-grading C programming assignments with CodeBERT and Random Forest Regressor

Roshan Vasu Muddaluru
Dept of Computer Science Engineering
Amrita Vishwa Vidyapeetham
Bengaluru, India
BL.EN.U4CSE20141@bl.students.amrita.edu

Sharvaani Ravikumar Thoguluva
Dept of Computer Science Engineering
Amrita Vishwa Vidyapeetham
Bengaluru, India
BL.EN.U4CSE20156@bl.students.amrita.edu

Shruti Prabha
Dept of Computer Science Engineering
Amrita Vishwa Vidyapeetham
Bengaluru, India
BL.EN.U4CSE20157@bl.students.amrita.edu

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

Ms. Roshni M Balakrishnan
Dept of Computer Science Engineering
Amrita Vishwa Vidyapeetham
Bengaluru, India
m_roshni@blr.amrita.edu

***Abstract*— Grading coding assignments manually is challenging due to complexity and subjectivity. However, auto-grading with deep learning simplifies the task. It objectively assesses code quality, detects errors, and assigns marks accurately, reducing the burden on instructors while ensuring efficient and fair assessment. This study provides an analysis of auto-grading of the C programming assignments using machine learning and deep learning approaches like regression, convolutional neural networks (CNN) and long short-term memory (LSTM). Using a code-based transformer word embedding model called CodeBERT, the textual code inputs were transformed into vectors, and the vectors were then fed into several models. The testing findings demonstrated the efficacy of the suggested strategy with a root mean squared error (RMSE) of 1.89. The contrast between statistical methods and deep learning techniques is discussed in the study.**

Keywords—CodeBERT, convolutional neural networks (CNN), long short-term memory (LSTM), validation loss, regression.

I. INTRODUCTION

Majority of the educational institutions follow a grading system where the marks of each student are entered manually. By using a set of predetermined criteria or rubrics, a human grader evaluates a student's work as part of the manual grading process. This contrasts with auto-grading, which rates the work using computer programs and Artificial Intelligence. Manual grading has several drawbacks, including: Heavy time-consumption where hand grading can take a lot of time, especially when dealing with big courses or tasks. The time and effort required to read through and assess each student's work can be challenging.

Inconsistency: The grading process can be inconsistent because variousgraders interpret and apply the grading criteria in different ways. This can lead to a variety of grades for the same assignment, from different evaluators. Bias: The subjective nature of the grading process may be compromised by the grader's own prejudices or preferences when manual gradingis involved. For instance, a grader could intentionally providehigher marks to work that is consistent with their own opinions or ideals. Lack of prompt feedback: Manual gradingfrequently takes some time to offer students with detailed feedback, which can delay the learning process and make it harder for students to make timely improvements to theirwork.

Looking at all these factors, it is necessary to put forth an effective auto grading system. Essays, programming projects, multiple-choice questions, short answer questions, and other forms of assignments can all be graded using autograding. In comparison to traditional grading techniques, autograding can offer a number of advantages, including a reduction in the time and efforts needed to evaluate assignments, the ability to give students instant feedback, and a lower risk of bias in grading.

The fact that autograding offers both students and teachers a number of advantages makes it a crucial practice in contemporary education. Once the autograding model learns the patterns required for the grading process, it is much simpler than a human grader grading assignments or tests. This will surely become a normalcy in the near future. The assignment will simply be uploaded onto the grading AI platform and it will do the rest of the job efficiently in seconds. Students will have varying styles of writing and sometimes it may affect the grading system due to recognition errors but for coding assignments where the code can just be typed, the results will be much more accurate. Deep Learning has evolved over the years, as it canhandle complex data and sometimes showing much better results in comparison to machine learning models. CodeBERT is a pre-trained language model created by Microsoft which is appreciated because of its unique featuressuch as detecting similarity in code, code summarization andcode completion. The steps done by CodeBERT to convert the C codes into vectors were to first tokenize the code into tokens taking syntax into account, then segmentation which involved breaking down the long words to small words to reduce the input size followed by masking and data augmentation. Because of the advantageous features of CodeBERT, it proved to be the best word embedding pre- processing model for this research.

The objective of this paper is a thorough comparative examination of several AI Models used to grade C programming assignments using CodeBERT, with a particular emphasis on how well they do in auto grading. The study's specific objective is to use CodeBERT to convert unprocessed C programs into vectors, then apply Random Forest regressor, Extra Trees Regressor, KNN and deep learning models (CNN and LSTM) to predict grades from 1 to 10. With a focus on contrasting the performance of machine learning and deep learning methodologies, the goal is to provide a clear evaluation of the efficiency and applicability of different AI models. By attaining this goal, the study hopes to enhance automated systems for grading C programming assignments and give insights on the advantages and drawbacks of various AI models in this situation.

The paper is structured as follows: Section II describes the literature survey which mentions various methods used to similar to the ones presented in this paper and discussing their work. Section III introduces the data and how it was gathered along with its understandings. Section IV discusses about the methodology that has been employed in this paper. This is followed by discussion of the results and their analysis and lastly, Section VI talks about the conclusion and future scope.

II. Literature Survey

Understanding how a person sees or evaluates a solution is crucial in order to comprehend the grading process. The strategy differs from person to person. It was advised to use rubrics to ensure some consistency. According to Srikant et al [1-3], evaluating a computer program in accordance with a rubric is a crucial component of an auto-grading system. A score (quantitative measure) on such a rubric frequently relates to the programmer's ability to solve problems. The machinelearning approach for automatically grading programs was presented by Srikant et al [3] utilizing a novelfeature language that encapsulates the key elements thathuman experts consider when assigning a mark.

An open-source code was developed to help recruiters save efforts on grading the code submitted by applicants. A precision of 89% was achieved, which was proved to be better than the precision obtained while using the bag of words concept [4]. Scientific Production and visualization analysis was done using Bibliometrix and VOSviewer software for an automated grading system for essay writing. Visualization Analysis was done after applying deep learning models such as CNN to identify sentence similarity in [5]. Thiago et al. [6] developed a system, 'SiameseQAT' to find duplication descriptions written by workers about a bug report. A recall of 85% and a AUROC value of 84% was obtained which was a good improvement in comparison to other works. A cloud service was created which evaluates a student's drawing and gives a grade. Maysa et al. [7] used neural networks to develop this model and gave an accuracy result of 91%.

An interpretable deep learning system for automatically scoring request for proposals (RFPs) was proposed. The suggested method extracts information from the text of RFPs and predicts a score for each proposal by combining natural language processing (NLP) methods and a convolutional neural network (CNN). To train and test their model, the authors compiled a dataset of RFPs and the accompanying scores. When comparing anticipated and real scores, they discovered that their suggested strategy worked better than other cutting-edge methods, reaching a correlation coefficient of 0.81 [8]. The study offered by Ardakani et al. [9], a data-driven method for predicting the feelings and attitudes represented in written language, known as affective text categorization analysis where Natural language processing (NLP) techniques were employed by the authors to extract features from a dataset of news articles and social media posts, such as word frequency, word co-occurrence, and grammatical structure. The deep learning and natural language processing (NLP) automated essay grading system that was improved suggested the method uses NLP techniques like part-of- speech tagging and dependency parsing to extract information from essay text, including syntactic and semantic data. To train and test their system, which uses a deep learning model, especially a convolutional neural network (CNN), to predict the grade of an essay, the authors employeda dataset of essays and their accompanying grades. Thesuggested solution outperformed other cutting-edge techniques in grading essays with an accuracy of 86.2% [10].

To evaluate source code files at various resolutions and find plagiarism in source code files for C, C++, and Java, the Discrete Wavelet Transform (DWT) was proposed [11]. The software has been subjected to static code analysis in order to find harmful blocks [12]. The effectiveness of static code analysis was further improved by adding compiler transformations [13]. To enhance the grading of essays, an unsupervised word sense disambiguation (WSD) method is recommended [15]. In order to determine whether a code's transformations had an effect on how well neural program analyzers worked, Md. Rabin and Md. Alipour [16] worked on neural code analyzers. They discovered that the size of the code blocks and the amount of modifications that must be used for the analysis affect the performance of the neural code analyzers. Eye-tracking was used to analyze the code blocks, according to Chandrika & Amudha [14].

The originality of this paper lies in its comparison of deep learning and statistical methods using the word embedding model, CodeBERT, which was then fed into deep learning models through which a comparison between a few regressors was observed.

III. Data Description

The data used in this research is a set of C programs which were collected from undergraduate, freshman, and sophomore students. The data set comprises of 765 rows and 2 columns: (i) ‘Code’ column contains all the C programs, (ii) ‘Score’ column has the marks awarded for each program. The focus is mainly on programs such as insertion, deletion and searching for an element in an array as well as the linked list implementations of stack and queue, all coded in C language. The dataset was split into three sets: a training set,a validation set, and a testing set. The data was split in such away that 50% of the data is used for training, 25% for validation & the rest 25% for testing. The goal is to analyze as to whether the statistical methods perform better than the deep learning models in order to give a more accurate prediction of the marks.

A. Dataset creation

The marks allotted were entered manually for each of these programs based on certain predefined criteria such as logic, output, syntax and the general semantics of how each program has been coded. The dataset was made manually by inducing errors into the code, and corresponding marks had been allocated to the code according to the marking scheme shown in Table. 1.

The marking scheme shows that in case of no output, 2 markswill be deducted and in case there is no logic to be found forthe corresponding code, 3 marks will be deducted. Similarly,if a program was only half completed, then, only 3 marks were awarded. The minimum marks awarded to every program was around 3 to 4.

TABLE I. MARKS REDUCED FOR EACH COMPONENT

Components	Weightage
No Output	2 marks reduction
Incorrect Syntax	1 marks reduction
Incorrect Logic	3 marks reduction

B. Data exploration

The dataset was based on a set of 5 questions: Implement the 1) stack operations using linked list, 2) queue operation using linked list, 3) insertion of an element in an array, 4) deletion of an element in an array, 5) operations on an array when the element is even or odd. The data deals with natural language texts consisting of around 200,000 words including special characters such as syntaxes like semicolon or brackets.

The maximum number if words in one row were found to be around 1400. There were no null values found in the data as the marks have been manually entered for each program.

As for the pre-processing, the text data which represents several C programming codes used the CodeBERT word embedding from the Microsoft's pretrained model. The pre-processing was followed by the train-test split where the feature ("Code") and its corresponding labels ("Score") were split into three sets: a training set, a validation set, and a testing set.

C. System Architecture

The incoming text data was converted into a vector space with the help of the word embedding model, CodeBERT. Fig. 1. shows that these vectors were then fed into various machine learning models as well as deep learning models.

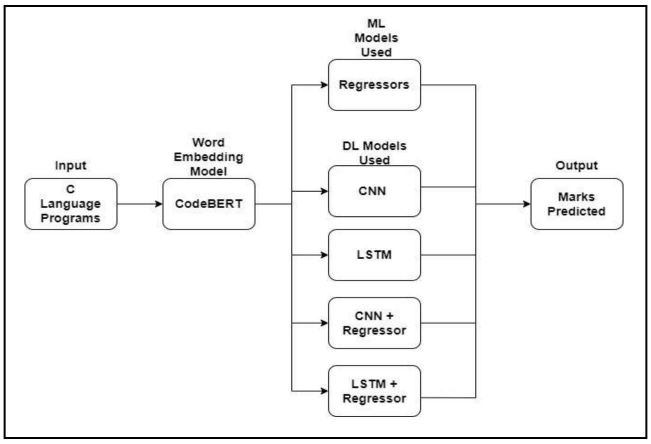

Fig. 1. System architecture for the model

The purpose of bringing in the deep learning models was to carry out experiments to check whether they perform better than statistical approaches, or if they overfit the data.

IV. METHODOLOGY

The paper brings a comparative analysis with the help of two different approaches namely machine learning and deep learning where both the following approaches take the vectors as input coming from the pre-processing word embedding model, CodeBERT.

A. Statistical approach

The statistical models used under this approach were the machine learning regressor models. The regressors used were: Random Forest (RF), Extreme Gradient boosting (XGBoost), Ridge Regression and K-nearest neighbor(KNN) regressor. Hyperparameter tuning was performed using GridsearchCV of Python sklearn package to find the best set of hyper-parameters for all the models. Various hyper-parameters such as the depth of the tree, learning rate and other regularization parameters were considered. It was fit to the training data which was then reshaped to have a single dimension equal to the number of features. The hyperparameters were tuned using cross-validation with 5 folds, and the best set of hyperparameters were determined based on the mean score across all folds. After doing so, these regressors were used to predict the marks(score) awarded for the corresponding code snippets.

B. Deep Learning approach

The deep learning approach of the analysis deals with 2 models: CNN and LSTM. This paper intends to carry out experiments for the CNN and LSTM models to see if they perform well with sequential data. The model for the CNN architecture consisted of a 1-dimensional convolutional layer with 32 filters, followed by a max pooling layer. A fully linked layer with 64 units and ReLU activation received the output of the pooling layer after it had been flattened. The finallayer had a single output unit with linear activation.

As the input data consists of textual codes, a sequential model was implemented where the experiment of working with LSTM model was carried out. The model consisted of a Long Short-Term Memory (LSTM) layer with 128 units, 20% dropout rate and 20% recurrent dropout rate. The outputof the LSTM layer was passed through a fully connected layer with 64 units and ReLU activation. Finally, the output layer had a single output unit with ReLU activation. Since the output range of the system architecture is numerical, the use of ReLU activation function was found to be better than otheractivation functions such as sigmoid and tanh.

Out of the machine learning models, the RF regressor was found to be the best to make an accurate prediction on the marks awarded. Since the random forest regressor was able to produce good results among the machine learning models, a novelty of combining the deep learning model along with the random forest was brought up and implemented to see if they would produce better results. For combining the CNN model with the random forest, the fully connected layer of the CNN model was swapped with the RF as shown in Fig. 2. A). Similarly, Fig. 2. B). shows the combination of the LSTM model with the random forest regressor after the LSTM layer was swapped with the regressor and output of the marks predicted.

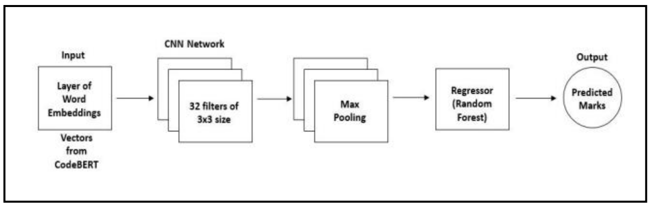

Fig. 2. A) Architecture for CNN with RF

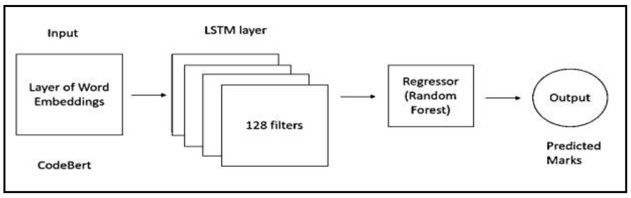

Fig. 2. B) Architecture for LSTM with RF.

After plotting the loss curves for CNN, CNN with random forest, LSTM and LSTM with random forest shown in Fig. 3., it was seen that the validation loss for the models, CNN and CNN with random forest were not consistently improving and are thus overfitting some parts of the training data. As for the LSTM model, it was demonstrated that there was a stability in the model but was unable to learn the training data and thus again fell under the overfitting category. The loss curves for the LSTM with random forest regressor indicated that the model was able to learn the training data till 4 epochs and later became stable by falling under the regular fit category. The optimal number of epochs for these 4 models were found to be around 40 epochs.

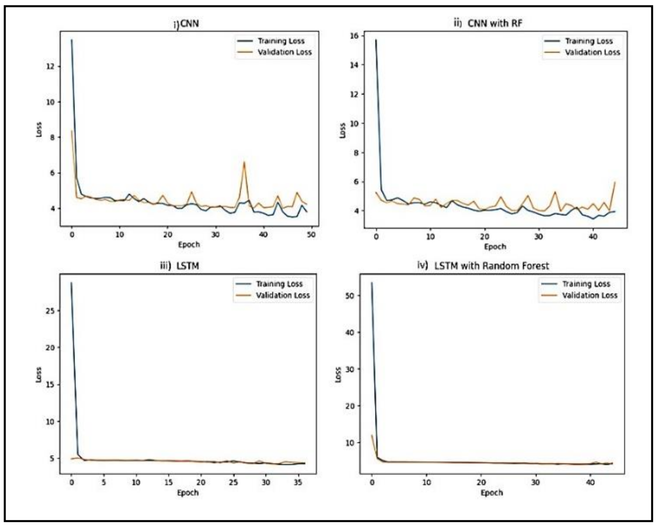

Fig. 3. Loss curves for various models implemented

V. RESULTS AND ANALYSIS

In this section, the analysis of the statistical models as well as the deep learning models discussed in Section IV is exhibited and a comparative analysis of the performance metrics are put forward. The models were evaluated based on the following metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE) and Coefficient of Determination (R2) as shown in Table II.

The statistical models as discussed in the previous section were tuned using hyperparameter tuning and then evaluated using the R2 score and MAPE of which are demonstrated in the same Table II. Although random forest

TABLE II. PERFORMANCE METRICS FOR ALL THE MODELS

Models	**Training set**				**Testing set**			
	RMSE	**MAE**	**R2**	**MAPE**	**RMSE**	**MAE**	**R2**	**MAPE**
Random forest	0.71	0.56	0.89	9.76	1.89	1.52	0.27	26.12
Ridge regressor	1.52	1.24	0.51	20.73	1.94	1.62	0.22	27.15
XGBoost	0.92	1.08	0.82	8.91	2.18	1.73	0.12	25.79
KNN regressor	1.75	1.41	0.42	21.76	2.39	1.94	0.16	25.80
CNN	1.87	1.15	0.25	35.24	2.09	1.71	0.11	35.40
LSTM	1.98	1.63	0.15	34.49	2.13	1.76	0.07	34.75
CNN with random forest	1.72	0.54	0.89	9.37	2.16	1.66	0.05	27.78
LSTM with random forest	0.74	0.58	0.88	10.26	1.91	1.53	0.20	26.72

showed overfitting as depicted in the high variation of test and train R2 score, the MAE and RMSE of Random Forest Regressor was the least, indicating better accuracy in comparison to the other models. The other models did not show overfitting, however had higher test RMSE value. The combination of LSTM with random forest also gave satisfying results with respect to RMSE due to the sequential nature of LSTM

For the random forest regressor that gave the best results out of the other statistical regressors, the combination of hyperparameters suggested that the best performing model preferred smaller values for the minimum samples needed to divide an internal node and the minimum samples needed to be at the node, indicating that themodel was able to learn fine-grained details of the training data. The best performing model did not require any limit onthe maximum depth of the tree. The model, which employed100 trees, a typical default value, provided an appropriate balance between calculation time and performance.

All the statistical models (machine learning regressors) shown in Fig. 4. were overfitting the data except for the ridge regressor. However, the RMSE of the models were taken into consideration. Since, the RMSE for the ridge regressor being 1.94 exceeds that of the random forest regressor having a value of 1.89, the random forest regressor was an appropriate model for the data.

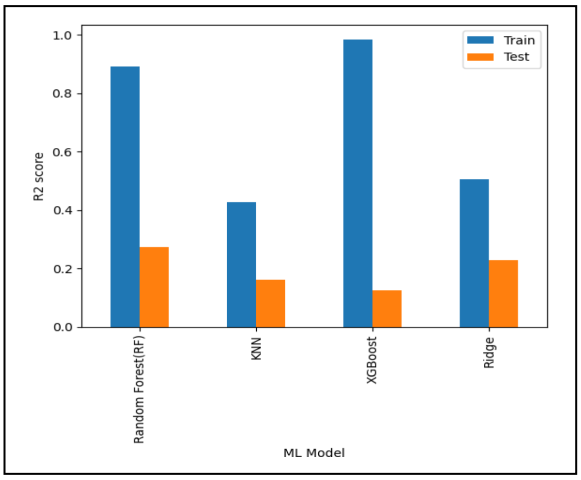

Fig. 4. Coefficient of Determination (R2) of Statistical models for test and train

Hence, the random forest regressor was chosen despite having overfit the data and the experiment with the combination of the deep learning approach with that of the statistical approach was implemented. While hyperparameter tuning was performed on the machine learning models, the deep learning models were also trained with a regularization technique called the early stopping method which was based on validation loss for 50 epochs with a batch size of 64.

Fig. 5. shows that the RMSE for the random forest regressor and LSTM with random forest were found to be similar and better in comparison to the other deep learning models. Both of these models also showed a similar performance for the R2 score as they were able to explain 89% and 88% of the training set and 26% and 20% respectively on the test set. However, both the models were overfitting the data but the LSTM with random forest regressor was showing a greater overfitting compared to that of the random forest regressor alone.

The statistical model also achieved a greater performance in a lesser training time compared to the other models.

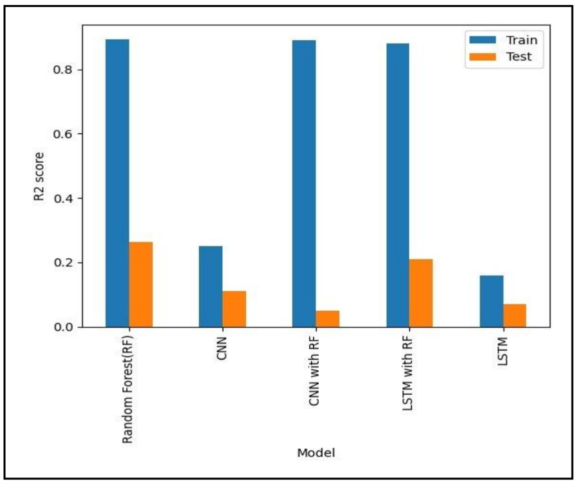

Fig. 5. Coefficient of Determination (R2) of Deep learning models for test and train

The research carried out in [17] also used CodeBERT where a RMSE value of 1.56 using the random forest was achieved. However, since the data used in this paper was a superset of the data in [17] and with more variety and different code snippets, the resulting RMSE achieved in this paper's work was higher than the latter. The data used for this work had inconsistencies and was almost twice as large as the data used in [17] leading to more overfitting.

VI. Conclusion and Future scope

The research objective of correctly forecasting the grades given for the relevant C programming codes was accomplished using the statistics and deep learning methodologies.

The Random Forest regressor was determined to be the model with the best performance when using a statistical method, whereas the LSTM model, which performs well due to its sequential nature, produced the best results when using a deep learning technique. The experiments conducted throughout the paper, however, revealed that the transformations from the text space to the R-n space and back to the R space resulted in a loss of information, which prevented the deep learning models from being able to explain much of a variance. In contrast, the Random Forest regressor was found to be the best of both approaches because it was seen to have a better RMSE and was able to explain more of the training data as well. This was due to the fact that the statistical models did not result in the prediction losing information, making the statistical method the best strategy as expected. The deep learning models overdid and were unable to perform well in comparison to the statistical models, which typically do not require a large dataset, because the dataset that was obtained was rather small.

The work of this paper could be extended and improvised to overcome the limitations caused due to the dataset being small by expanding and refining the dataset. This way the overfitting situation could also be solved. The future scope of this work would be to employ more code-based transformer word embedding models such as CodeT5, CuBERT and code2vec and further perform a comparative analysis and thus arrive at the best autograding system for C programs. However, this could further be extended to other programming languages like Java, python and C++ as well.

ACKNOWLEDGMENT

Amrita Vishwa Vidyapeetham provided the necessary infrastructure and support for the conduct of this research activity, as well as for the production of the publication, for which the authors are grateful. The professors of the freshmen and sophomore year students helped the writers collect the necessary data, and for that the authors would also want to express their sincere gratitude.

REFERENCES

[1] V. Aggarwal, S. Srikant, and V. Shashidhar, "Principles for using Machine Learning in the Assessment of Open Response Items: Programming Assessment as a Case Study," in Proceedings of the NIPS Workshop on Data Driven Education, 2013.

[2] S. Srikant and V. Aggarwal, "Automatic Grading of Computer Programs: A Machine Learning Approach," in Proceedings of the 12th International Conference on Machine Learning Applications (ICMLA), 2013.

[3] S. Srikant and V. Aggarwal, "A system to grade computer programming skills using machine learning," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 1887-1896.

[4] Rai, Kaushal Kumar, Bhavya Gupta, Pardeep Shokeen, and Pinaki Chakraborty. "Question independent automated code analysis and grading using bag of words and machine learning." In 2019 International Conference on Computing, Power and Communication Technologies (GUCON), pp. 93-98. IEEE, 2019.

[5] Efendi, Toni, Fetty Fitriyanti Lubis, Atina Putri, Dana Waskita, Tri Sulistyaningtyas, Yusep Rosmansyah, and Jaka Sembiring. "A Bibliometrics-Based Systematic Review on Automated Essay Scoring in Education." In 2022 International Conference on Information Technology Systems and Innovation (ICITSI), pp. 275-282. IEEE, 2022.

[6] Rocha, Thiago Marques, and André Luiz DaCosta Carvalho. "Siameseqat: A semantic context-based duplicate bug report detection using replicated cluster information." IEEE Access 9 (2021): 44610-44630.

[7] Khaleel, Maysa Faroun, Mohamed Abu Sharkh, and Mohamad Kalil. "A cloud-based architecture for automated grading of computer-aided design student work using deep learning." In 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 1-5. IEEE, 2020.

[8] Maji, Subhadip, Anudeep Appe, Raghav Bali, Arijit Ghosh Chowdhury, Veera Chikka Raghavendra, and Vamsi M. Bhandaru. "An interpretable deep learning system for automatically scoring request for proposals." In 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 851-855. IEEE, 2021.

[9] Ardakani, Saeid Pourroostaei, Can Zhou, Xuting Wu, Yingrui Ma, and Jizhou Che. "A Data-driven Affective Text Classification Analysis." In 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 199-204. IEEE, 2021.

[10] Ibrahim, Samira Said, Essameldean F. Elfakharany, and Essam Hamed. "Improved Automated Essay Grading System Via Natural Language Processing and Deep Learning." In 2022 International Conference on Engineering and Emerging Technologies (ICEET), pp. 1-7. IEEE, 2022.

[11] N. G. Resmi and Dr. S. K. P., "Multiresolution analysis of source code using discrete wavelet transform," International Journal of Applied Engineering Research, vol. 9, pp. 13341-13360, 2014.

[12] P. Seshagiri, Vazhayil, Ab, and P. Sriram, "AMA: Static Code Analysis of Web Page for the Detection of Malicious Scripts," in Proceedings of the 6th International Conference on Advances in Computing & Communications, 2016, Procedia Computer Science, vol. 93, pp. 768-773, Elsevier.

[13] M. G. Thushara and K. Somasundaram, "Static Analysis on Floating Point Programs Dealing with Division Operations," International Journal of Advanced Computer Science and Applications, vol. 10. 2019.

[14] K. R. Chandrika and J. Amudha, "A fuzzy inference system to recommend skills for source code review using eye movement data," Journal of Intelligent & Fuzzy Systems, vol. 34, no. 3, pp. 1743-1754, 2018.

[15] P. Nedungadi and H. Raj, "Unsupervised word sense disambiguation for automatic essay scoring," Smart Innovation, Systems and Technologies, vol. 27, pp. 437-443, 2014.

[16] R. Islam Rabin, A. Alipour, "ProgramTransformer: A tool for generating semantically equivalent transformed programs," Software Impacts, vol. 14, 2022, pp. 1-29, doi: 10.1016/j.simpa.2022.100429.

[17] N. Nakka & P. B. Pati, “Autograding of Programming Skills”, 2023 IEEE International Conference for Convergence in Technology (I2CT), INDIA.