International Journal of Electrical and Computer Engineering (IJECE)
Vol. 15, No. 3, June 2025, pp. 3346~3361
ISSN: 2088-8708, DOI: 10.11591/ijece.v15i3.pp3346-3361

Eval$_{BERT}$: a novel framework for assisted descriptive answers and C programming evaluation

Prakruthi Sondekere Thippeswamy[1], Manjunathswamy Byranahalli Eraiah[2], Salma Jabeen[1]
[1]Department of Computer Science and Engineering, Don Bosco Institute of Technology Research Centre, Visvesvaraya Technological University, Belagavi, India
[2] Department of Computer Science and Engineering, Guru Ghasidas Vishwavidyalaya, A Central University, Bilaspur, India

Article Info

Article history:

Received Jun 14, 2024
Revised Jan 20, 2025
Accepted Mar 3, 2025

Keywords:

Bidirectional encoder representations from transformers
Bi-directional long short-term memory
Graphical processing unit
Multinomial Naïve bayes
Natural language programming

ABSTRACT

Manual assessment of descriptive answers is often time-consuming, error-prone, and subject to bias. While artificial intelligence (AI) has made significant strides, current automated evaluation methods typically rely on simplistic metrics like word counts or predefined terms, which lack a deeper understanding of the content and are highly dependent on curated datasets. As demand for automated grading systems increases, there is a growing need to evaluate not only descriptive answers but also code-based responses. This study addresses these challenges by applying natural language processing (NLP) and deep learning (DL) techniques, testing three baseline models: multinomial Naïve bayes (MNB), bidirectional long short-term memory (Bi-LSTM), and bidirectional encoder representations from transformers (BERT). We propose Eval$_{BERT}$, a BERT-based model fine-tuned with domain-specific academic corpora using computer processing unit (CPU) acceleration. EvalBERT automates grading for both descriptive and C programming exams, offering features like readability statistics and error detection. Experimental results show that Eval$_{BERT}$ achieves 94.86% accuracy, outperforming other models by 1.22 percentage points, with training time reduced by half. Additionally, Eval$_{BERT}$ is the first model pre-trained with academic corpora for this purpose. An interactive user interface, E-Pariksha, was also developed for administering and taking exams online. Eval$_{BERT}$ provides precise assessments, enabling educators to better evaluate student performance and offer more detailed feedback.

Corresponding Author:

Prakruthi Sondekere Thippeswamy
Department of Computer Science and Engineering, Don Bosco Institute of Technology Research Centre, Visvesvaraya Technological University
Machhe, Belagavi, Karnataka 590018, India
Email: st.prakruthi@gmail.com

1. INTRODUCTION

Academic achievements in India are typically gauged through summative assessments such as standardized tests and examinations. The evaluation of answer scripts plays a crucial role in determining students' understanding of a subject. However, there is a lack of standardized systems for assessing exam papers nationwide. The current evaluation process involves multiple stages, including junior and senior evaluators, grade calculators, and moderators, making it quite time-consuming. Ensuring consistency among evaluators over several days of assessment can be challenging, leading to discrepancies in grading. Various factors may hinder evaluators from accurately assessing descriptive answers and determining the students' intelligence level. Even when presented with similar responses, different evaluators may assign significantly

different grades, influenced by their individual assessment approaches and levels of experience. Hence, there is a need for a more standardized and objective marking methodology [1].

Subjective questions and their answers offer a means to evaluate a student's performance and understanding in a more open-ended manner. These responses aren't restricted by specific guidelines, allowing students to express themselves based on their individual perspectives and comprehension of the topic. However, subjective answers differ significantly from objective ones in several key aspects. Firstly, they tend to be lengthier compared to objective responses. Secondly, writing subjective answers typically requires more time. Additionally, evaluating subjective answers demands a greater level of attention, as they often contain more context and require a teacher's objectivity and concentration. It is simple and useful to use machines to evaluate objective responses. It is possible to provide a program question and one-word answers so that it can map student responses effectively. At this point, automated marking technology for objective questions is widely used and has matured to a high degree. For subjective questions, a few auto assessment systems exist, but none of them quite live up to expectations. This is mostly because natural language is difficult to understand and has many ambiguities. To accurately score subjective responses, the evaluator must carefully evaluate each word in the response. Variables including the evaluator's weariness, mental condition, and objectivity have a big impact on the outcome. Moreover, individuals often use synonyms and convenient abbreviations, further complicating the evaluation process. As a result, subjective question marking still has many shortcomings. Various preprocessing steps, such as data cleaning and tokenization, are necessary before analysis can begin. Post this, a variety of methods, including idea graphs, ontologies, latent semantic structures, and document similarity, can be used to compare textual data. The final score can then be calculated by considering elements like language usage, structure, similarity, and the existence of keywords [2], [3]. Although this issue has been addressed in the past, there is still need for improvement [4]–[6].

Additionally, C programming-based exams are often perceived as more daunting by both students and teachers. This is because C programming involves intricate syntax rules and semantics. Automatic natural language processing (NLP) systems may struggle to accurately interpret and understand the nuances of C code, especially in cases where context is crucial for determining correctness. C programs can range in complexity from simple algorithms to advanced data structures and algorithms. Automatically assessing the correctness and efficiency of such code requires sophisticated analysis beyond basic NLP techniques. Also, C code may contain ambiguous constructs or multiple valid solutions. NLP models may struggle to handle such ambiguity and accurately determine the correctness of the code without additional context or domain-specific knowledge. Moreover, identifying errors in C code, such as logical errors, syntax errors, or runtime errors, requires more than just linguistic analysis. It involves understanding the logic and behavior of the code, which may be challenging for NLP models without specialized programming knowledge. Thus, human intervention and specialized tools designed for code analysis are often necessary to overcome these challenges effectively.

In this paper, we investigate three approaches for evaluating subjective answers and C programming answers using machine learning (ML), deep learning (DL) and natural language processing (NLP) techniques. Our approach utilizes various NLP pre-processing methods, including tokenization and lemmatization, word embeddings like word2Vec. We also employ similarity measurement techniques such as cosine similarity and word mover distance (WMD), along with classification methods like multinomial naive Bayes, Bi- LSTM and BERT. Overall, it employs a two-step process to address this challenge. Initially, the answers are assessed by comparing them to the provided solution and keywords using different similarity-based techniques as mentioned above. Subsequently, the outcomes from this initial step are utilized to train the classification models capable of evaluating answers independently, without relying on predefined solutions and keywords. To assess the effectiveness of different models, we employ accuracy as the main metrics, comparing their performance and against other models. Therefore, the key contributions from this paper are:

- $Eval_{BERT}$: A novel descriptive and C-code evaluation model is proposed by training BERTBASE with domain-specific corpora. It outperforms the baseline models with an accuracy of 94.86% and is 1.22 pp higher in accuracy than the next best existing model compared.
- By using new hardware architectures like GPU and fitting its memory with all mini-batch training data, we reduce the $Eval_{BERT}$ training time by half compared to training it on a CPU-only system.
- An interactive frontend web-interface called E- Pariksha is developed for faculty to administer the exam and students to take online exams, with EvalBERT assisting evaluation at the backend.

The rest of the paper is structured as follows: next section introduces the background of the problem and reviews relevant literature. This is followed by proposed methodology. Next, we discuss the experimental analysis and present the results. Finally, concluding remarks are mentioned for the paper.

2. LITERATURE REVIEW

Raut *et al.* [1] discussed various similarity measures such as Cosine similarity and word mover distances between students' answers and model answers. The weight value for each similarity measure is manually assigned for assessment. However, text summarization is not performed which is a limitation in their work. Nandini and Maheswari [7] utilize question-answer classification and feature extraction for automatic answer evaluation. They incorporate matching keywords and similarity measures to determine the final score, achieving a system performance with a precision, recall and sensitivity value of 95%, 94%, and 94.5% respectively. Bashir *et al.* [8] discuss the use of WMD and responsibility measures, along with word2vec for semantic similarity. They also present two score prediction algorithms that achieve up to 88% accuracy. Patil and Ali [9] discuss various approaches for automating and assisting evaluators in grading answer scripts. They highlight the requirement for large training datasets and utilize techniques such as LSA, machine learning, and statistical methods. However, existing systems often lack accuracy when tested on standard and common datasets, suggesting that assistance to evaluators could enhance accuracy. Dharma Tetali *et al.* [10] propose the TADACO ReportLab, tested with a sample of 120 students through tool for evaluating descriptive answers. They develop a model using Python modules like Pyuic, Platypus, and semi-automated and manual evaluation. Results show that the semi-automated mode yields better results compared to the completely automated mode. Vij *et al.* [11] discuss both auto and manual evaluations, noting minimal differences in random forest principle and was trained random forest principle and was trained on 530 samples to automatically evaluate descriptive answers. Combéfis [12] reviews and classifies techniques and tools used for automated code assessment in educational settings, exploring their advantages, limitations, and potential improvements. The study concludes that while current tools provide valuable support in evaluating student code, there remains significant room for enhancing feedback quality and adapting assessments to more complex, real-world programming scenarios. Dubey and Makwana [13] aims to develop a computer-assisted system for evaluating descriptive answers using Weka's random forest classification technique. The results demonstrate that the proposed method effectively classifies descriptive responses with high accuracy, offering a viable automated solution for evaluating open-ended questions in educational settings. Vinothina and Prathap [14] proposed EVaClassifier designed for the automated assessment of descriptive answers, employing support vector machines (SVMs). The system's performance is assessed based on the accuracy of grading provided by the supervised machine learning algorithm it employs. Table 1 compares the performance of various automatic evaluation techniques in literature. It is observed that most of the techniques use similarity measures and grammatical mistakes for evaluation. We find very few models based on deep learning and none using latest transformer models to extract decisive evaluation parameters like text summarization and C code error log generation for programming-based answers.

Table 1. Comparison of various automatic evaluation techniques and their accuracies

Sl. No.	Topic/model	Technique	Accuracy	Remarks
1	Intelligent essay evaluator [15]	Clustering method, latent semantic analysis technique	60-90%	Most of the auto-evaluation techniques have considered similarity measures and grammatical mistakes for evaluation. None of them have used BERT to extract decisive parameters for evaluation like text summarization, C code error log generation for programming-based answers.
2	Automated essay scoring using Bayes theorem [16]	Classification method, Bayes theorem	76%	
3	Descriptive answer assessment in online examination [7]	Naïve Bayes classification algo. and relation-based feature extraction technique	95% Prec, 94% recall, 94.5% sensitivity	
4	Automatic answer script evaluation using NLP and OCR [17]	OCR technique, NLP and deep learning analysis	Average 80%	
5	Evaluation of short answers on C-rater system [18]	NLP technique	80%	
6	C-Rater: automatic assessment of students free-text answers [19]	BLEU and LSA	50%	
7	TADACO-A semi-automatic assessment tool [10]	Matching keywords and phrases	Good results vs. manual evaluation	
8	Eklavya-AI proctored online examination system [20]	Machine learning algorithms	90% accuracy	
9	Deep automated text scoring model Based on memory network [21]	Deep learning LSTM network	83.37%	

3. MATERIALS AND METHOD

A high-level system architecture is illustrated in Figure 1. A web interface named ‘E-Pariksha’ is designed for faculty log-in to add descriptive questions and display them using Django. The faculty must log-in and upload the descriptive course name and questions, along with the marks for each question into the

database. Any student taking the examination should log-in through the student login page which redirects to the page where the questionnaire is displayed. Students must answer the questions for which the answers and keywords are stored in the database. The answers are available to the faculty for evaluation after they log-in for assessment. The administrator of the evaluation portal has to login to view courses, for conduction of exam, oversee faculty and student activities and publish exam results. The ML or transformer-based models work at the backend in evaluation of both the descriptive and C-coding questions

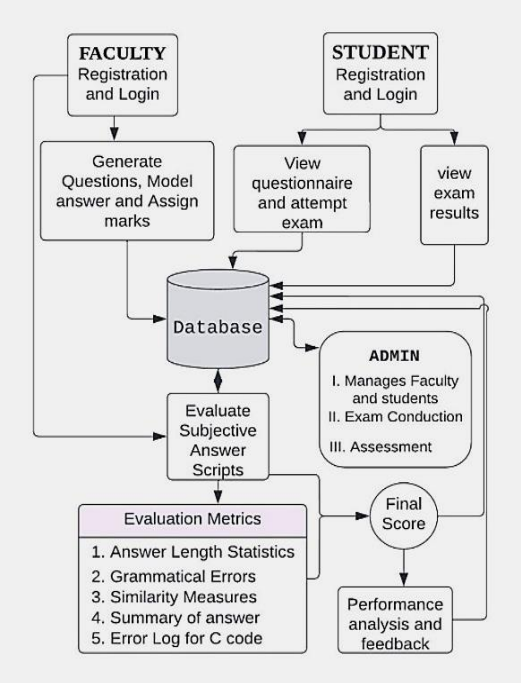


Figure 1. System architecture of the proposed model

4. PROPOSED METHODOLOGY

The proposed method consists of a web interface. Each component of the interface is briefly explained below.

a. Registration phase: A web interface frontend is designed where student and faculty must register themselves to log-in to the webpage. Admin has all the controls of login and database maintenance. Faculty can frame the question paper which is displayed to the student and the student must answer and upload their response.
b. Student's questionnaire uploading and answer retrieval: Faculty must log in and upload question ID, Question description and marks allotted to each question into the database. The questionnaire is displayed to the students. Students must answer the questions and answers uploaded are stored in the database. The same are reflected in the faculty login for assessment.
c. Feedback phase: A feedback module is used to provide performance analysis and feedback to the student.

4.1. Evaluation metrics

Answer evaluation metrics module is used for generating answer length, grammatical errors, similarity measure between model and student's answer, summary of descriptive answer, finding similarity with model answer and syntactic error log for C-code based answers which are briefly explained below.

a. Answer length (evaluation parameter 1): In many descriptive exams, the length of the answer is expected to be limited to 150 words, 200 words and so on. So, it is very important to know the length of the answer for evaluation including other statistics like line and word count. Answer length statistics includes number of letters, words and lines present in the student's answer. This score (*S1*) contributes 10% to the overall score.
b. English and grammatical errors (evaluation parameter 2): Functions of NLTK library are used for language modeling to find out English and grammatical errors which need to be avoided in descriptive answers, for better assessment [22]. This score (S2) contributes 10% to the total score.

c. Similarity measure (evaluation parameter 3): Efficient similarity techniques are implemented for comparing model answers and student's answers. Similarity techniques like WMD, cosine, Jaccard and bigram similarity can be used. Thepercentage of similarity shall be displayed for the evaluator'sreference. WMD and cosine similarity measure are used in this research [23], [24] which are explained below.

Word mover's distance (WMD) is a measure of similarity between two text documents. It leverages the word embeddings of words in the documents to compute the minimal "cost" of transforming one document into the other. Word embeddings represent words as vectors in a continuous vector space. Let v_i denote the embedding of the *i-th* word in the vocabulary.

Consider the two documents, model answers as D1 and students answer as D2, represented as weighted bags of words. Let x_i and y_j denote the words in D1 and D2, respectively. The distance between words x_i and y_j is typically measured using the Euclidean distance between their embeddings as (1):

$$d(x_i, y_j) = \left\| v_{x_i} - v_{y_j} \right\|_2 \quad (1)$$

WMD frames the similarity measure as an optimal transportation problem. It finds the minimum cumulative cost required to move the distribution of word embeddings of one document to match the other. Let T_{ij} be the flow between word x_i in D1 and word y_j in D2. The optimization problem can be formulated as (2):

$$minimize \sum_{i,j} T_{ij} \cdot d(x_i, y_j) \; subject \; to \sum_j Tij = wi, \forall_1 ; \sum_i T_{ij} = w_j, \forall_j ; \; Tij \geq 0, \forall i, j \quad (2)$$

where w_i and w_j are the weights of word x_i in D_1 and word y_j in D_2, respectively. Typically, these weights can be the term frequencies or normalized term frequencies (so that they sum to 1). In the above equation, Flow Matrix T represents how much of the word x_i from D_1 is transported to word y_j in D_2. And Cost $d(x_i, y_j)$ represents the "cost" or "effort" required to transform x_i into y_j.

The objective is to find the transportation plan T that minimizes the total cost, which directly corresponds to the dissimilarity between the two documents. The score *(S3_WMD)* from this evaluation contributes 80% to the total score. Cosine similarity is a measure of similarity between two non- zero vectors of an inner product space that measures the cosine of the angle between them. This measure is particularly used in high-dimensional positive spaces, such as in text analysis, where each term or word is a dimension. Given two vectors A and B, the cosine similarity $sim\ (A, B)$ is defined as (3):

$$sim(A, B) = \frac{A.B}{\|A\|\|B\|} \quad (3)$$

where, $(A.B)$ is the sum of the products of the correspondingentries of the two sequences of numbers. norm (or length) of the vector, calculated as the square root of the sum of the squares of its components.

The cosine similarity ranges from -1 to 1. 1 indicates that thevectors are identical. 0 indicates that the vectors are orthogonal (no similarity) and -1 indicates that the vectors arediametrically opposed. In this study, considering model answers as D_1 and students answer as D_2 represented by TF-IDF vectors A and B:

$$A = [a_1, a_2, a_3, \ldots., a_n] ; \; B = [b_1, b_2, b_3, \ldots., b_n]$$

The cosine similarity is computed as (4):

$$sim\ (a.b) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}} \quad (4)$$

This measure helps in determining how similar two documents are based on their term distributions, providing faculty a simple and intuitive metric for text similarity. The score (*S3_cosine*) from this evaluation contributes 80% to the total score.

d. Summary of student's answer (Evaluation parameter 4): The purpose is to create a cogent and fluent summary having only the important points outlined in the document. The summarized text of every answer of student is displayed for the evaluator's assistance in assessment. Extractive summarization, an efficient summarization technique is used for generating a summary of student's answers. The extractive summarization technique uses the same text as in the original. It is less complex than the abstractive method. It tends to be more accurate than the abstractive method, as it simply picks out the sentences from the original text itself. This method is suitable for domains where there is less variation in the language.

e. Syntactic error log for C code-based answers (evaluation parameter 5): The error log is manually generated for the student's C code. Parameters like missed semicolons, matching brackets and parentheses, wrongly spelled keywords, undefined variables, improper data types, incorrect function arguments and preprocessor directives are considered for checking errors in C code-based answers. This score (S5) contributes 100% to the total score for this metrics. St is the actual total score by adding S1 to S4 metrics as shown in Table 2. And the proposed algorithm for evaluation is mentioned in Algorithm 1.

Table 2. Weightage of individual evaluation metric for descriptive answers

Evaluation metric	Weightage
For descriptive answers	
Answer length (*S1*)	5%
English and Grammatical errors (*S2*)	5%
Similarity index (*S3_cosine or S3_WMD*)	80%
Summary of student's answer (*S4*)	10%
Total score (*St*)	100%
For C Programming answers	
Syntactic error log (S5)	100%

4.2. Corpus details

Since there are no publicly available labeled descriptive and C code question answers, we curate a labeled corpus by crawling previous ten years examination question papers from various university websites, their solutions, student answers and data belonging to computer science and general knowledge from various websites. The corpus contains necessary keywords in questions which were extracted from the solution. Keywords are terms that are unique to a question and are necessary to provide an answer. These keywords, which just need to have the most important terms in lower case, have a big impact on how well the similarity assessment module in the model scores. Students' answers are mapped using the solution, which is a subjective response including synonym words to descriptive questions or parameters like semicolons, matching brackets and parentheses, correctly spelled keywords, defined variables, proper data types, correct function arguments and preprocessor directives in C code-based answers. Every term and context covered in the answers must be included in this solution in different lines or paragraphs. The solution to a question is prepared by the faculty/evaluator.

For training and testing the model, we need a large corpus. The crawled data is annotated as it is unlabeled. Both the descriptive and C code question answers are annotated, and a best score based on the answers given by students is given. The annotated corpus consists of over 655 descriptive and 345 C code related questions, each with a correct answer (solution) and annotated 10 student answers to each of the questions. The corpus thus contains a total of 10,002 answers, keywords for descriptive questions and C code parameters extracted from the solutions and data from various computer science knowledge-based websites. Table 3 shows the distribution of scores in answers sheets used in the corpus.

Table 3. Score distribution in answer sheets

Questions	Score											
	100	90	80	70	60	50	40	30	20	10	0	sum
Descriptive	111	251	478	830	1076	1296	1086	836	380	167	40	6551
C-code	144	468	656	487	657	356	287	114	102	156	24	3451

4.3. Models used

The proposed research uses three machine learning or deep learning models to measure semantic similarity in text, namely multinomial Naïve bayes, Bi-LSTM and BERTBASE and BERT pretrained on domain specific corpora proposed as EvalBERT. The models are discussed briefly in the section below.

a. Multinomial naïve Bayes

Multinomial naive bayes (MNB) model, a probabilistic classifier is commonly employed in text classification tasks in NLP, with applications ranging from information retrieval to question answering systems. The MNB model assumes that features are generated from a multinomial distribution, making it suitable for text classification tasks where the features represent word frequencies. To adapt MNB for measuring semantic similarity, we treat documents or sentences as bags of words, ignoring word order and considering only the frequency of each word. Mathematically, given two text samples X and Y, represented

by their word frequency vectors x and y respectively, the semantic similarity S between them can be estimated using MNB as (5):

$$S(X,Y) = \frac{\sum_{i=1}^{n} \min(x_i, y_i)}{\sum_{i=1}^{n} \max(x_i, y_i)} \quad (5)$$

where x_i and y_i represent the frequency of word i in samples X and Y respectively, and n is the total number of unique words in both samples. By comparing the word frequencies of two text samples using MNB, we obtain a similarity score ranging from 0 to 1, where 0 indicates no similarity and 1 indicates identical text. This approach provides a simple yet effective method for measuring semantic similarity in answers.

b. Bi-directional long-short term memory network

Bi-LSTM networks, as shown in Figure 2 is a variant of recurrent neural networks (RNNs) used for capturing contextual information and semantic relationships within text sequences in answers. It can be employed to encode input text sequences into fixed-dimensional vectors, capturing both forward and backward contextual information. These vector representations can then be compared to determine the similarity between two text inputs. Mathematically, given two input text sequences X and Y, represented as sequences of word embeddings x = (x1, x2, ..., xn) and y = (y1, y2, ..., ym) respectively, the semantic similarity S between them can be computed using Bi-LSTM as (6):

$$(X,Y) = cosine\_similarity(Bi - LSTM(x), Bi - LSTM(y)) \quad (6)$$

where $Bi - LSTM(x)$ represents the output vector representation of input sequence x obtained from the Bi-LSTM network, and $cosine\_similarity$ computes the similarity between the two output vectors.

c. Bidirectional encoder representations from transformers

As shown in Figure 3, BERTBASE (12 layers, 110M parameters) and BERTLARGE (24 layers, 330M parameters) versions exhibit different levels of architectural complexity. An encoder stack built on the Transformer architecture is the standard component of BERT. It has an embedding layer that, like the typical encoder in the Transformer model, receives a string of words as input and passes it on to the next encoder unit. Every encoder layer self-attenuates, and the results are distributed via a feed-forward network [24].

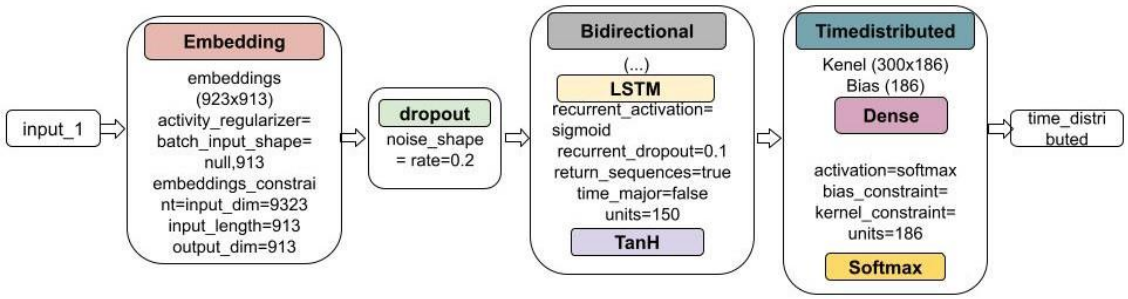


Figure 2. Proposed Bi-LSTM for semantic similarity in answers

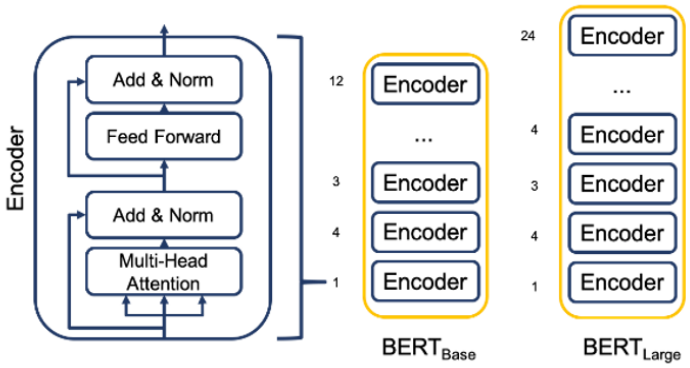


Figure 3. BERT model architecture

The following encoder layer receives the feedforward network's output. For different tasks, BERT uses a fine-tuning strategy without requiring particular changes. A pre-trained BERT model can reach state-of-the-art performance by adding an extra layer. Using training set data, the BERT architecture is optimized while maintaining the necessary format for data organization. BERT layers receive three input arrays:

- *input_ids*: These are integers corresponding to each word
- In the input sequence.
- *attention_mask*: It is shown as either 1 or 0. It indicates which elements of the *input_ids* array should be attended to.
- *token_type_ids*: Used to distinguish responses, it requires special tokens like [CLS] and [SEP] to separate distinct responses within the *input_ids*.

The *encode_plus* function from the tokenizer class tokenizes the raw input. A [CLS] indicator was affixed at the start of the text token, followed by the addition of a [SEP] token at the end. Subsequently, each token was sequentially assigned an index, and the length of each sentence was assessed against the maximum length, with padding applied to those falling short of it. To ensure proper formatting of raw data for input into the BERT model, a helper function is utilized. Following this, an attention mask was created to enhance learning efficiency. In this study, a maximum length of 364 was set for descriptive answers. $Eval_{BERT}$: As shown in Figure 4, $BERT_{BASE}$ is extended and pre-trained with additional corpora consisting of descriptive and C programming questions, answer keys, and answer scripts from various websites, university portals, class exams and fine-tuned.

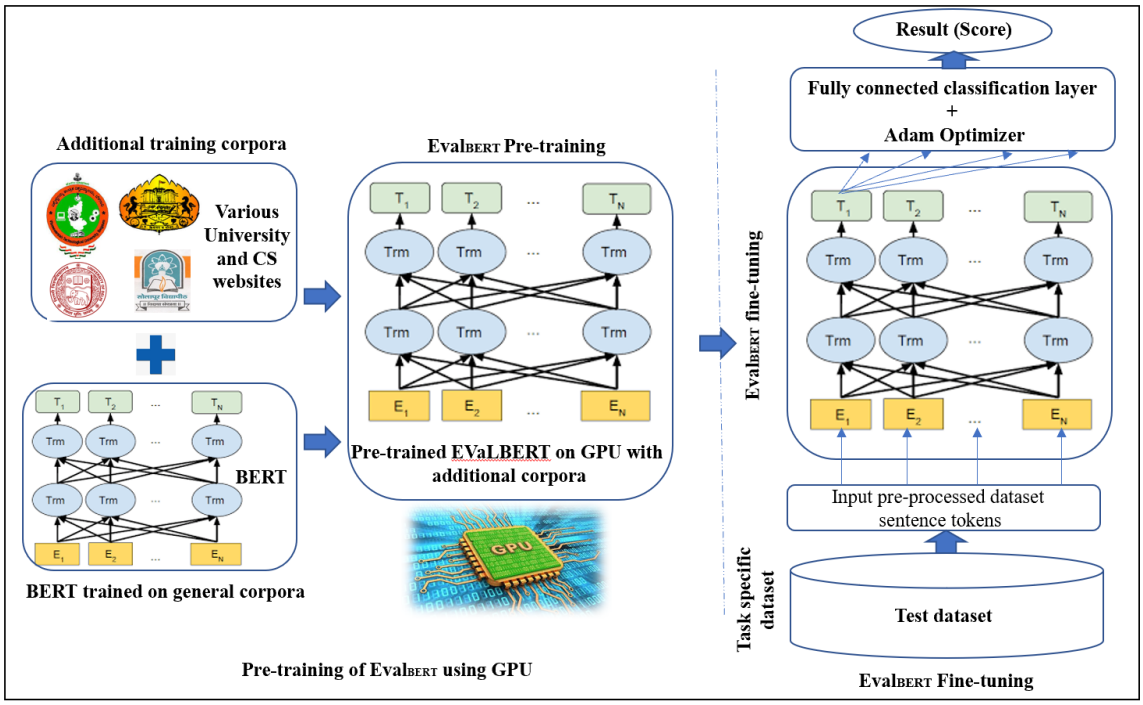


Figure 4. Pre-training and fine-tuning in proposed EVaLBERT model

Pre-training: Though plain $BERT_{BASE}$ is good for general corpora, its performance drops on task-specific data. Therefore, we use large amount of annotated data for unsupervised pre-training. The data was divided into 70% training and 20% for testing. We used 10% data at validation stage. To feed the training data as per input format of BERT, we used sentence splitter library to separate the content into sentence units, followed by separating the score labels. To tokenize the content, we used BERT base-cased which converts sentences into numeric indices, calculates the maximum sequence length of the input token, and fills any missing data with padded zeroes. The attention mask is next initialized and separated into training and validation. The batch size is set and to avoid overfitting and longer training time, we limited the epochs to 20, using Nvidia GPU which accelerates the training. The various model layers are

- Input layers (*input_ids and attention_mask*): input_ids layer is responsible for receiving the tokenized input sequence. Each word or subword is represented as an integer from the tokenizer. The *attention_mask* layer specifies which tokens are meaningful and which are padding tokens. It helps the model focus only on the actual input sequence and ignore the padded portions.
- BERT model layer (*tf_bert_model*): This is the pre-trained BERT model. In this case, it's using the *TFBaseModelOutput* architecture from TensorFlow (TF) which includes last_hidden_state, the output embedding representation for each token in the sequence, and pooler_output, a vector representation for the entire sequence (for classification). The output size is (None, 768), where 768 is the hidden layer size. This layer has over 108 million parameters, making it the core of the model.
- Global max pooling layer (*global_max_pooling1d_7*): This layer applies a max pooling operation over the output from the BERT model. It reduces the dimensionality by taking the maximum value across the tokens for each feature, resulting in a single vector of size (768) per input sequence.
- Dense layer 1 (*dense_21*): This fully connected layer further processes the pooled output by passing it through 768 neurons. It adds more capacity to the model for better feature extraction. This layer has 590,592 parameters.
- Dropout layer 1 (*dropout_47*): A dropout layer is applied to prevent overfitting by randomly setting a fraction of the input units to zero at each update during training. No trainable parameters here.
- Dense layer 2 (*dense_22*): The output from the previous layer is passed through a smaller fully connected layer with 128 neurons. This step allows the model to compress the feature representation further and reduce the dimensionality. This layer has 98,432 parameters.
- Dropout layer 2 (*dropout_48*): Another dropout layer is applied to further prevent overfitting before passing the data to the next dense layer.
- Dense layer 3 (*dense_23*): Another dense layer with 32 neurons is applied. This layer focuses on learning more compressed representations of the input data. It has 4,128 parameters.
- Output layer (*dense_24*): The final output layer consists of 2 neurons, likely for a binary classification task. It outputs the prediction probabilities. The layer has 66 parameters.

Fine-tuning: An extra fully connected classification layer with Adam optimizer is added that fine-tunes the learning and leverages the pre-trained backbone weights as shown in Figure 5. Even for this task, a single GPU was used. Assessment of various models is carried out using standard relative performance parameter like accuracy.

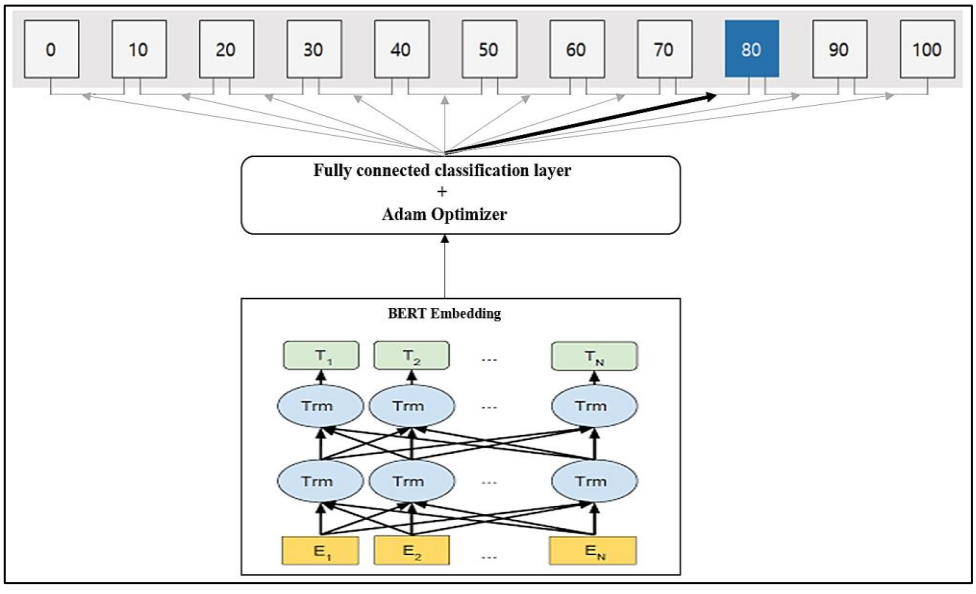


Figure 5. Fine tuning EVaL$_{BERT}$ model

Algorithm 1. Proposed algorithm for evaluation

```
1: Input student's answers digitally through the web interface and store them in the
   database.
```

2: Preprocess the answer removing stop words, tokenize each sentence and use CountVectorizer to represent them in TF-IDF form
3: Exercise decisive evaluation metrics for evaluation of answers viz. Answer length (*S1*), English and Grammatical errors (*S2*), Similarity measure (*S3*), summary of student's answer (*S4*) and Syntactic error log for C code-based answers (*S5*)
4: Calculate score prediction using similarity measure viz. Word distance mover (*S3_WDM*) and Cosine similarity (*S3_cosine*)
5: Word movers distance method:
6: Calculate distance between words using Euclidean distance $d(x_i, y_j) = \left\| v_{x_i} - v_{y_j} \right\|_2$
7: Frame the similarity measure as an optimal transportation problem to minimize $\sum_{i,j} T_{ij} \cdot d(x_i, y_j)$ subject to $\sum_j Tij = wi, \forall_I$, $\sum_i T_{ij} = w_j, \forall_j$ and $Tij \geq 0$, then $\forall i, j$
8: Cosine similarity method:
9: Calculate cosine similarity $sim(A,B) = \frac{A.B}{\|A\|\|B\|}$ where $A.B = \sum_{i=1}^{n} A_i B_i$; and norm $\|A\| = \sqrt{\sum_{i=1}^{n} A_i^2}$; $\|B\| = \sqrt{\sum_{i=1}^{n} B_i^2}$.
10: **if** *sim (A,B)* = 1,
then *S3*= 80% (the vectors are identical)
else if *sim (A,B)*~ 0,
then *S3*<10% (the vectors are orthogonal with less or no similarity)
11: Calculate overall score by combining *S1,S2,S3* for descriptive answers and *S4* for C programming answers
12: Assess, award marks, and accomplish performance analysis and feedback to students.
13: End

5. RESULTS AND DISCUSSION

The results are tabulated with and without using the MNB model and the two similarity measures, WMD and Cosine similarity. The assessment was carried out by a faculty (evaluator score) which is compared with the score predicted using one of the similarity measures. Table 4 shows the score predictions for five answer scripts and the error using cosine similarity and without MNB model support. Table 5 shows the scores comparison for the first five answers used for training purposes using cosine similarity and with model support. It is seen that average error decreases from 7.2 marks to 4.2 marks when using model suggestions. Table 6 shows the scores comparison for the first five answers used for training purposes using WDM similarity and without MNB model support. Table 7 shows the score predictions of five answer scripts with the error using WDM similarity with MNB model support.

Table 4. Score prediction using cosine similarity measure without MNB support

Evaluator score	Actual score (S_t) Cosine	Error
56	61	5
43	52	9
0	07	7
78	84	6
49	58	9
Average error		7.2

Table 5. Score prediction using cosine similarity measure with MNB support

Evaluator score	Actual score (St) Cosine	Error
56	59	3
43	48	5
0	03	3
78	83	5
49	54	5
Average error		4.2

Table 6. Score prediction using WDM similarity measure without MNB support

Evaluator score	Actual score (S_t) Cosine	Error
56	60	4
43	49	6
0	06	6
78	83	5
49	56	7
Average error		5.6

Table 7. Score prediction using WDM similarity measure with MNB support

Evaluator score	Actual score (S_t) Cosine	Error
56	59	3
43	47	4
0	04	4
78	81	3
49	55	6
Average error		4.0

It is seen that average error decreases from 5.6 marks to 4.0 marks when using MNB model suggestions. Figure 6 shows the accuracy of both the approaches with and without the machine learning model support. It is seen that accuracy is 84.71% and 87.39% for WMD and cosine similarity respectively without using the ML model suggestion. The accuracy decreased to 82.54% and 80.08% for WMD and cosine similarity along with the classification model used. It is because the results of cosine similarity measure are semantically weaker which prevents the model from getting trained on the right data as in the case of WDM. Figure 7 shows the weights and other hyperparameters of $Eval_{BERT}$ model.

Table 8 shows the accuracy comparison of all models implemented. Machine learning models along with similarity measure could maximum attain an accuracy of 82.54%. Deep learning provides a good option for NLP tasks and both Bi-LSTM and BERT perform better than ML models with an accuracy of 87.89% and 88.74% respectively. However, $Eval_{BERT}$ pre-trained on domain-specific academic corpora outperforms all models with an accuracy of 94.86%.

Table 9 shows the accuracy comparison of $Eval_{BERT}$ with existing state-of-the-art models. Bashir *et al.* [8] use MNB along with WMD and cosine similarity and attain a maximum accuracy of 87%. Lee *et al.* [25] use a fine-tunes BERT model for Korean test database. They attain a maximum accuracy of 93.64% with 20 epochs, similar to the no. of epochs used in this study. However, $Eval_{BERT}$ outperforms it with an accuracy of 94.86%. Figure 8 shows the training and validation accuracy charts of the Bi-LSTM model. Figure 8(a) shows a good training and validation accuracy for the model. The model fits both new data and training data well, as seen by the validation loss plot and the training loss plot in Figure 8(b), respectively.

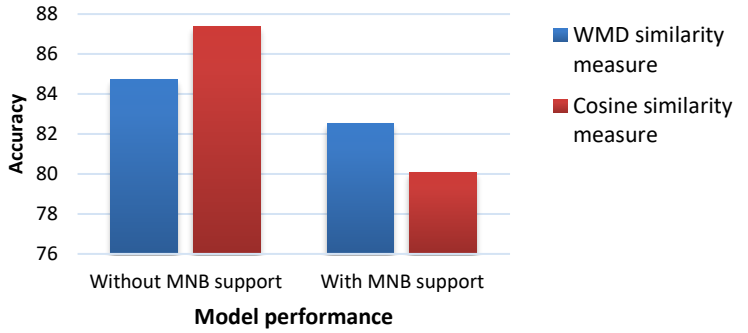


Figure 6. Accuracy of both approaches with and without using ML model

```
Model: "model_6"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_ids (InputLayer)      [(None, 70)]              0           []

 attention_mask (InputLayer) [(None, 70)]              0           []

 tf_bert_model (TFBertModel) TFBaseModelOutputWi       108310272   ['input_ids[0][0]',
                             thPoolingAndCrossAt                    'attention_mask[0][0]']
                             tentions(last_hidde
                             n_state=(None, 70,
                             768),
                              pooler_output=(Non
                             e, 768),
                              past_key_values=No
                             ne, hidden_states=N
                             one, attentions=Non
                             e, cross_attentions
                             =None)

 global_max_pooling1d_7 (Global  (None, 768)           0           ['tf_bert_model[6][0]']
 MaxPooling1D)

 dense_21 (Dense)            (None, 768)               590592      ['global_max_pooling1d_7[0][0]']

 dropout_47 (Dropout)        (None, 768)               0           ['dense_21[0][0]']

 dense_22 (Dense)            (None, 128)               98432       ['dropout_47[0][0]']

 dropout_48 (Dropout)        (None, 128)               0           ['dense_22[0][0]']

 dense_23 (Dense)            (None, 32)                4128        ['dropout_48[0][0]']

 dense_24 (Dense)            (None, 2)                 66          ['dense_23[0][0]']

=================================================================================================
Total params: 109,003,490
Trainable params: 109,003,490
Non-trainable params: 0
```

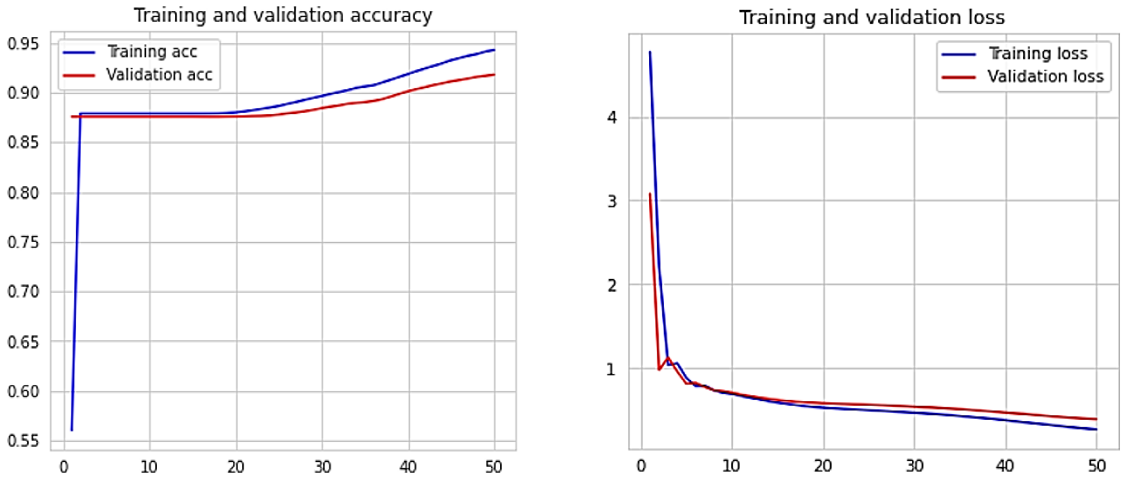
Figure 7. Weights and other hyper-parameters in EvalBERT model

Table 8. Comparison of accuracy results of various models implemented

Model	Word Embedding/ Similarity measure	Acc (%)
Multinomial Naïve Bayes	Cosine similarity	80.08
	Word mover distance	82.54
Bi-directional LSTM	Cosine similarity	87.89
BERTBASE	General corpora	88.74
Proposed $Eval_{BERT}$	General+academic corpora	94.86

Table 9. Comparison with existing models

Reference	Model	Acc (%)
Bashir *et al.* [8]	MNB+WMD	87.00
Jung *et al.* [25]	BERT	93.64
Proposed	$Eval_{BERT}$	94.86

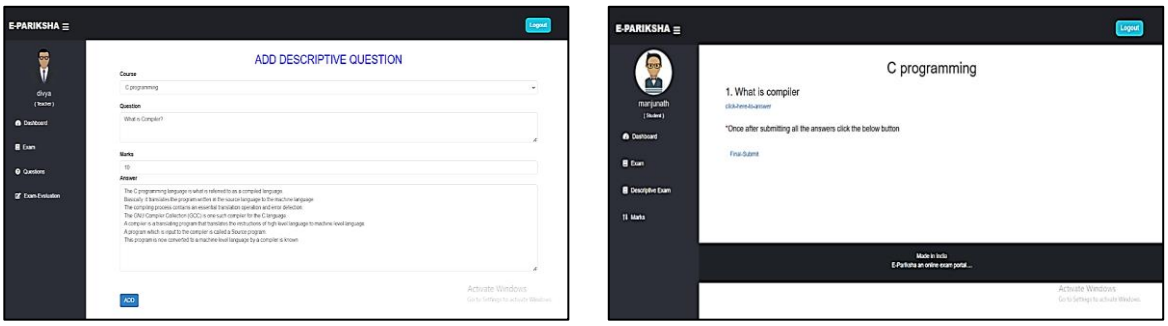


(a) (b)

Figure 8. Learning curves of the proposed model (a) accuracy and (b) loss

E-Pariksha: A web interface named 'E-Pariksha' is designed using Django for faculty and student login as shown in Figure 9. The faculty must log-in and upload descriptive course name and descriptive questions as shown in Figure 9(a) along with marks for each question into the database. Any student taking an examination needs to log-in through the student login page which redirects to the page where the questionnaire is displayed as shown in Figure 9(b). Students must answer the questions and answers are stored in the database. The same is reflected in the faculty login for assessment. Table 10 displays the results of evaluation metrics for the question "What is a compiler?" and a C program-based question "Write a program to read and print the value of variable" to assist the evaluator and provide feedback. Similarly, the administrator of the evaluation portal must log-in to view courses, faculty, and student activities.

The study's findings demonstrate that $Eval_{BERT}$, when trained on domain-specific academic corpora, significantly improves the accuracy of automated grading for descriptive answers and C programming responses, achieving 94.86% accuracy. This result is supported by the fact that $Eval_{BERT}$ outperforms baseline models, including MNB and Bi-LSTM, with a 1.22 percentage point improvement over the nearest competitor. The integration of GPU for training also halved the model's training time, demonstrating its efficiency for large-scale academic evaluation tasks.



(a) (b)

Figure 9. E-Pariksha frontend (a) adding descriptive questions to a specific course and (b) student answering a question in the 'C Programming' course

Compared to previous studies that primarily relied on simpler models or keyword-based approaches, this research demonstrates the clear advantage of transformer-based models like BERT for more nuanced and accurate assessments. Previous studies struggled with limitations in dataset quality, scalability, and performance accuracy, which our study addresses with specialized training on academic data. A strength of our study lies in its ability to evaluate both descriptive and code-based responses, but its limitations include the focus on English and C programming, leaving room for broader language and programming support. Unexpectedly, GPU acceleration not only reduced training time but also enhanced the overall performance, suggesting the importance of computational resources in model refinement.

Table 10. Display instance of evaluation metrics to assist the evaluator

Sl. No	Questions (marks allotted) and student answers	Evaluation metrics	Evaluators marks	Performance and feedback
1	*What is compiler? (10)* A Compiler is a software that typically takes a high-level language code as input and converts the input to a lower-level language at once. compiler translates the program written in the source language to the machine language. A compiler is a translating program that translates the instructions of high-level language to machine level language. The compiling process contains an essential translation operation and error detection. The C programming language is what is referred to as a compiled language. GCC is a compiler in C.	*a. Model Answer:* The C programming language is what is referred to as a compiled language. Basically, it translates the program written in the source language to the machine language. The compiling process contains an essential translation operation and error detection. The GNU Compiler Collection (GCC) is one such compiler for the C language. A compiler is a translating program that translates the instructions of high-level language to machine-level language. A program which is input to the compiler is called a Source program. This program is now converted to a machine-level language by a compiler is known as the Object code. *b. Answer Length statistics* Output: Word count: 86; Letter count: 439; Line count: 5 *c. Number of grammatical mistakes* No grammatical errors found in text2.txt. *d.* Percentage of similarity with the original answer The similarity between the two texts: 89.34805233826893 % *e. Summary of students' answer* A Compiler is a software that typically takes a high-level language code as input and converts the input to a lower-level language at once. The C programming language is what is referred to as a compiled language. The process contains an essential translation operation and error detection. *f. Syntactic error assistance for C programming-based answers* NA	8	Satisfactory Elaborate on compiler's functionality

2	*Write a program to read and print the value of variable (4)*
	
	#include <stdio.h>
	int main(){
	int a
	printf("Enter value: ");
	scanf("%d",&a);
	printf("%d",a);
	}
		a. Model Answer
		#include <stdio.h>
		int main(){
		int a;
		printf("Enter: ");
		scanf("%d",&var);
		printf("%d",var);
		}
		b. Answer Length statistics- NA; *c. Number of grammatical mistakes*- NA; *d. Percentage of similarity with the model answer*-NA; *e. Summary of students' answer*-NA
		f. Syntactic error assistance for C programming-based answers
		Error Log: /content/code.c: In function 'main':; /content/code.c:5:5: error: expected '=', ',', ';', 'asm' or '__attribute__' before 'printf'; 5 | printf("Enter: "); | ^; /content/code.c:6:17: error: 'a' undeclared (first use in this function; 6 | scanf("%d",&a); | ^ /content/code.c:6:17: note: each undeclared identifier is reported only once for each function it appears in.
			3	Good
				
				Avoid Syntax errors

This study aims to create an automated evaluation framework that can assist educators by improving the accuracy and efficiency of grading. The significance of this research lies in its potential to reshape automated academic assessments, reducing manual grading time while providing consistent, detailed feedback. However, unanswered questions remain, such as how $Eval_{BERT}$ could handle non-English languages, handwritten scripts, or complex responses like mathematical equations. Future research could explore expanding $Eval_{BERT}$'s capabilities to cover these areas and further enhance its practical applications in diverse educational settings.

The study's limitations include its current focus on evaluating only English-language descriptive answers and C programming responses, which restricts its applicability to other languages and programming languages. Additionally, while $Eval_{BERT}$ shows high accuracy, it depends heavily on the quality and relevance of the domain-specific academic corpora used for training. The system also requires significant computational resources, particularly for GPU-based training. Another limitation is its inability to handle handwritten answer scripts from offline exams, as the current framework is tailored for online exams. Expanding $Eval_{BERT}$ to recognize complex mathematical formulas, diagrams, or voice inputs for handicapped students is also a challenge that remains unaddressed in the current version.

6. CONCLUSION

In this research, we introduced $Eval_{BERT}$, a cutting-edge framework designed to assist evaluators in assessing descriptive answers and C programming responses. The framework leverages state-of-the-art models such as MNB, Bi-LSTM, and BERT, utilizing advanced techniques like WMD and cosine similarity for measuring response similarity. Through additional training of $BERT_{BASE}$ on domain-specific academic corpora using GPU acceleration, $Eval_{BERT}$ achieved a significant accuracy rate of 94.86%, outperforming the nearest competing model by 1.22 percentage points. Furthermore, GPU integration helped reduce training time by half, making the system more efficient for real-world applications. The development of the E-Pariksha interface further enhances the practical utility of $Eval_{BERT}$ by providing a seamless platform for administering and taking exams online. Faculty can now efficiently evaluate both descriptive and code-based responses, reducing manual intervention while offering prompt, detailed feedback to students.

Our findings have important implications for the research field and the academic community. $Eval_{BERT}$ represents a key advancement in the automation of complex assessment tasks, particularly in contexts where qualitative feedback is essential. By providing a precise, scalable, and fast solution, it addresses the growing demand for automated evaluation systems in education. The system not only reduces evaluation time but also ensures consistency and reduces the potential for human bias in grading. Looking forward, $Eval_{BERT}$ has the potential to be extended in several ways. Future work could expand its language support to handle non-English responses and extend its capabilities to other programming languages beyond C. Additionally, integrating tools for evaluating handwritten answer scripts from offline exams, voice recognition for handicapped students, and handling mathematical formulas and diagrams would greatly broaden its applicability. These enhancements would further solidify $Eval_{BERT}$'s role as a comprehensive tool for modern academic evaluation.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Prakruthi Sondekere Thippeswamy	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Manjunathswamy Byranahalli Eraiah		✓		✓		✓		✓		✓	✓	✓		
Salma Jabeen		✓		✓		✓				✓		✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

INFORMED CONSENT

We have obtained informed consent from all individuals included in this study.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author, P S T, upon reasonable request.

REFERENCES

[1] S. P. Raut, S. D. Chaudhari, V. B. Waghole, P. U. Jadhav, and A. B. Saste, "Automatic evaluation of descriptive answers using NLP and machine learning," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 735–745, Mar. 2022, doi: 10.48175/IJARSCT-3030.

[2] J. Wang and Y. Dong, "Measurement of text similarity: a survey," *Information*, vol. 11, no. 9, p. 421, Aug. 2020, doi: 10.3390/info11090421.

[3] M. Han, X. Zhang, X. Yuan, J. Jiang, W. Yun, and C. Gao, "A survey on the techniques, applications, and performance of short text semantic similarity," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 5, 2021, doi: 10.1002/cpe.5971.

[4] S. Patil and S. Patil, "Evaluating student descriptive answers using natural language processing," *International Journal of Engineering Research & Technology*, vol. 3, no. 3, pp. 1716–1718, 2014.

[5] P. Patil, S. Patil, V. Miniyar, and A. Bandal, "Subjective answer evaluation using machine learning," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 24, pp. 1–13, 2018.

[6] J. Muangprathub, S. Kajornkasirat, and A. Wanichsombat, "Document plagiarism detection using a new concept similarity in formal concept analysis," *Journal of Applied Mathematics*, vol. 2021, 2021, doi: 10.1155/2021/6662984.

[7] V. Nandini and P. U. Maheswari, "Automatic assessment of descriptive answers in online examination system using semantic relational features," *The Journal of Supercomputing*, vol. 76, no. 6, pp. 4430–4448, 2018.

[8] M. F. Bashir, H. Arshad, A. R. Javed, N. Kryvinska, and S. S. Band, "Subjective answers evaluation using machine learning and natural language processing," *IEEE Access*, vol. 9, pp. 158972–158983, 2021, doi: 10.1109/ACCESS.2021.3130902.

[9] R. G. Patil and S. Z. Ali, "Approaches for automation in assisting evaluator for grading of answer scripts: a survey," in *2018 4th International Conference on Computing Communication and Automation, ICCCA 2018*, 2018, pp. 1–6. doi: 10.1109/CCAA.2018.8777664.

[10] D. R. Tetali, G. Kiran Kumar, and L. Ramana, "A python tool for evaluation of subjective answers (aptesa)," *International Journal of Mechanical Engineering and Technology*, vol. 8, no. 7, pp. 247–255, 2017.

[11] S. Vij, D. Tayal, and A. Jain, "A machine learning approach for automated evaluation of short answers using text similarity based on WordNet graphs," *Wireless Personal Communications*, vol. 111, no. 2, pp. 1271–1282, 2020, doi: 10.1007/s11277-019-06913-x.

[12] S. Combéfis, "Automated code assessment for education: review, classification and perspectives on techniques and tools," *Software*, vol. 1, no. 1, pp. 3–30, Feb. 2022, doi: 10.3390/software1010002.

[13] R. Dubey and R. R. S. Makwana, "Computer-assisted valuation of descriptive answers using weka with randomforest classification," *Lecture Notes in Electrical Engineering*, vol. 476, pp. 359–366, 2019, doi: 10.1007/978-981-10-8234-4_31.

[14] V. Vinothina and G. Prathap, "EVaClassifier using linear SVM machine learning algorithm," *Advances in Intelligent Systems and Computing*, vol. 1034, pp. 503–509, 2020, doi: 10.1007/978-981-15-1084-7_48.

[15] T. K. Landauer, D. Laham, and P. Folt, "Automatic essay assessment," *Assessment in Education: Principles, Policy and Practice*, vol. 10, no. 3, pp. 295–308, 2003, doi: 10.1080/0969594032000148154.

[16] L. M. Rudner and T. Liang, "Automated essay scoring using Bayes' theorem," *Journal of Technology, Learning, and Assessment*, vol. 1, no. 2, pp. 1–22, 2002.

[17] P. Deepak, R. Rohan, R. Rohith, and R. R., "NLP and OCR based automatic answer script evaluation system," *International Journal of Computer Applications*, vol. 186, no. 42, pp. 22–27, Sep. 2024, doi: 10.5120/ijca2024924038.

[18] J. Z. Sukkarieh and S. Stoyanchev, "Automating model building in c-rater," in *TextInfer 2009 - 2009 Workshop on Applied Textual Inference at the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, ACL-IJCNLP 2009 - Proceedings*, 2009, pp. 61–69. doi: 10.3115/1708141.1708153.

[19] D. Pérez, A. Gliozzo, C. Strapparava, E. Alfonseca, P. Rodríguez, and B. Magnini, "Automatic assessment of students' free-text answers underpinned by the combination of a BLEU-inspired algorithm and latent semantic analysis," in *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2005 - Recent Advances in Artifical Intelligence*, 2005, pp. 358–362.

[20] "Artificial intelligence proctoring for secure assessment," *Ekalavya*, 2023. https://www.eklavvya.com/remote-proctoring/ (accessed Mar. 20, 2024).

[21] S. Yang, "Deep automated text scoring model based on memory network," in *Proceedings - 2020 International Conference on Computer Vision, Image and Deep Learning, CVIDL 2020*, 2020, pp. 480–484. doi: 10.1109/CVIDL51233.2020.00-46.

[22] M. Agarwal, R. Kalia, V. Bahel, and A. Thomas, "AutoEval: a NLP approach for automatic test evaluation system," in *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)*, Sep. 2021, pp. 1–6. doi: 10.1109/GUCON50781.2021.9573769.

[23] M. F. Bashir, H. Arshad, A. R. Javed, N. Kryvinska, and S. S. Band, "Subjective answers evaluation using machine learning and natural language processing," *IEEE Access*, vol. 9, pp. 158972–158983, 2021, doi: 10.1109/ACCESS.2021.3130902.

[24] M. Bali and A. S. Pichandi, "NeRBERT- a biomedical named entity recognition tagger," *Revue d'Intelligence Artificielle*, vol. 37, no. 1, pp. 239–247, 2023, doi: 10.18280/ria.370130.

[25] J. H. Lee, J. S. Park, and J. G. Shon, "A BERT- based automatic scoring model of Korean language learners' essay," *Journal of Information Processing Systems*, vol. 18, no. 2, pp. 282–291, 2022, doi: 10.3745/JIPS.04.0239.

BIOGRAPHIES OF AUTHORS

Prakruthi Sondekere Thippeswamy received the B.E. degree in computer science and engineering from VTU, Belgaum, Karnataka in 2009, and master's degree in computer science from Sri Siddhartha Academy of Higher Education in 2011. She has 12 years of teaching experience at various engineering colleges in Bangalore. She is currently a research scholar, pursuing Ph.D. at DBIT research Center, Bangalore. She has published 6 papers in various national and international conferences and journals. Her current research interests include machine learning, natural language processing and emerging technologies. She can be contacted at email: st.prakruthi@gmail.com.

Manjunathswamy Byranahalli Eraiah received B.E. degree in Telecommunication Engineering from VTU, Belgaum, Karnataka in 2005, M.E. (information technology) from Bangalore University in 2008. He is awarded with Ph.D. degree in CSE from Bangalore University, Karnataka in 2016. He has 16+ Years of teaching experience and 5+ years of research experience. His areas of interest include cognitive networks, image processing and data science and machine learning. He published 30+ papers in various national and international journals. He has filed 5+ patents. He is a member of ISTE, CSI, IAENG and WAIRCO. He can be contacted at email: manjube2412@gmail.com.

Salma Jabeen received the B.E. degree in CSE, M.Tech. degree in software engineering and Ph.D. in CSE from VTU Belgaum, Karnataka. She has published 10 research papers in referred international journals, 17 research papers in the proceedings of various international/national conferences and 1 patent. She has published book on advance artificial intelligence. Her areas of research include artificial intelligence, machine learning, data science, database, system software, unix system programing. She has membership in IEI, IAENG. She can be contacted at email: salmakhayum@gmail.com.