

**ENHANCED AUTOMATIC SHORT ANSWER
GRADING SYSTEM USING BERT MODEL
AND BI-LSTM NETWORK**

A PROJECT REPORT

Submitted by

HEMALATHA E (422420104304)

MONISHA S (422420104022)

SNEHA S (422420104035)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



UNIVERSITY COLLEGE OF ENGINEERING TINDIVANAM

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report “**ENHANCED AUTOMATIC SHORT ANSWER GRADING SYSTEM USING BERT MODEL AND BI-LSTM NETWORK**” is the bonafide work of “**HEMALATHA E (422420104304), MONISHA S (422420104022), SNEHA S (422420104035)**” who carried out the project work under my supervision.

SIGNATURE

Dr. L. Jegatha Deborah., M.E, Ph.D.,

HEAD OF THE DEPARTMENT,

Assistant Professor,

Department of CSE,

University College of Engineering

Tindivanam,

Melpakkam – 604 001.

SIGNATURE

Dr. L. Jegatha Deborah., M.E, Ph.D.,

SUPERVISOR,

Assistant Professor,

Department of CSE,

University College of Engineering

Tindivanam,

Melpakkam – 604 001.

Submitted for the University Examination held on _____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

At the outset, we like to express our gratitude to the God almighty. We express our thanks to our respected Dean **Dr. P. Thamizhazhagan, M.E., Ph.D.**, for his constant inspiration and encouragement throughout the project.

For mostly, we pay our grateful acknowledgement and extend our sincere gratitude to our head of the department **Dr. L. Jegatha Deborah, M.E., Ph.D.**, project internal guide for our better guidance and constant encouragement in completing this project work successfully. We appreciate her thoroughness, tolerance and ability to share her knowledge with us. We thank her for being easily approachable and quite thoughtful. We owe her harnessing our potential and bringing out the best in us. Without her immense support through every step of the way, we could never have it to this extent.

We extend our thanks to our project coordinator **Ms. P. Sathya, M.Tech.**, for her continual support during the entire project schedule. Without her immense support through every step of the way, we could never have it to this extent.

We thank all our teaching and non-teaching faculty members of the department and also our fellow friends for helping us in providing valuable suggestions and timely ideas for the successful completion of the project and also, last but not least we extend our immense thanks to our parents and siblings who have been a great source of inspiration and rendered us great support during the course of this project work. We sincerely thank all of them.

ABSTRACT

Automatic short-answer grading (ASAG) plays a vital role in intelligent tutoring systems (ITS) by assessing students' responses to questions. Deep learning has emerged as a sophisticated technique for comprehensively understanding textual content. However, applying deep learning to ASAG poses challenges, particularly in achieving high-precision scoring due to the complexity of evaluating short answers. Moreover, the limited size of ASAG's corpus restricts the availability of sufficient training data for effective deep learning models. A novel approach using a BERT-based deep neural network for ASAG is introduced to address these challenges. The Bidirectional Encoder Representations from Transformers (BERT) model is leveraged to enhance understanding of short-answer texts. To further enhance the ASAG system's performance, a specialized semantic refinement layer called Bidirectional Long short-term memory (Bi-LSTM) is integrated to generate high precision scores. Overall, by introducing a BERT and Bi-LSTM in the ASAG system can overcome challenges related to understanding short answers, handling limited training data, and improving grading accuracy.

Keywords: Automatic short-answer grading (ASAG), Bidirectional Encoder Representations from Transformers (BERT), Bi-LSTM network, Intelligent tutoring systems (ITSs), Textual entailment, Deep learning, Deep neural network.

திட்டப் பணிச்சுருக்கம்

கேள்விகளுக்கான மாணவர்களின் பதில்களை மதிப்பிடுவதன் மூலம் அறிவார்ந்த பயிற்சி அமைப்புகளில் (ஐ. டி. எஸ்) தானியங்கி குறுகிய-பதில் தரப்படுத்தல் (ஏ. எஸ். ஏ. ஜி) முக்கிய பங்கு வகிக்கிறது. ஆழமான கற்றல் உரை உள்ளடக்கத்தை விரிவாகப் புரிந்துகொள்வதற்கான ஒரு அதிநவீன நுட்பமாக உருவெடுத்துள்ளது. இருப்பினும் (ஏ. எஸ். ஏ. ஜி)க்கு ஆழமான கற்றலைப் பயன்படுத்துவது சவால்களை முன்வைக்கிறது. குறிப்பாக குறுகிய பதில்களை மதிப்பிடுவதில் உள்ள சிக்கல் காரணமாக அதிக துல்லியமான மதிப்பெண்களை அடைவதில். மேலும் (ஏ. எஸ். ஏ. ஜி) இன் கார்பஸின் வரையறுக்கப்பட்ட அளவு பயனுள்ள ஆழமான கற்றல் மாதிரிகளுக்கு போதுமான பயிற்சி தரவு கிடைப்பதை கட்டுப்படுத்துகிறது. இந்த சவால்களை எதிர்கொள்ள (ஏ. எஸ். ஏ. ஜி)க்கான பி. இ. ஆர். டி-அடிப்படையிலான ஆழமான நரம்பியல் வலையமைப்பைப் பயன்படுத்தி ஒரு புதிய அணுகுமுறை அறிமுகப்படுத்தப்பட்டுள்ளது. குறுகிய பதில்

உரைகளின் புரிதலை மேம்படுத்துவதற்காக இருதரப்பு என்கோடர் பிரதிநிதித்துவங்கள் டிரான்ஸ்ஃபார்மர்கள் (பி. இ. ஆர். டி) மாதிரி பயன்படுத்தப்படுகிறது. முன் பயிற்சி பெற்ற மற்றும் நேர்த்தியான (பி. இ. ஆர். டி) மாதிரியைப் பயன்படுத்துவதன் மூலம் அமைப்பு பதில் உரையை மாறும் வகையில் குறியாக்கம் செய்கிறது இது பதில்களின் நுணுக்கமான பகுப்பாய்வை செயல்படுத்துகிறது. ஏ. எஸ். ஏ. ஜி அமைப்பின் செயல்திறனை மேலும் மேம்படுத்தி சொற்பொருள் சுத்திகரிப்பு அடுக்கு என்று அழைக்கப்படும் ஒரு சிறப்பு அடுக்கு (பி. இ. ஆர். டி) வெளியீடுகளைச் செம்மைப்படுத்த ஒருங்கிணைக்கப்படுகிறது. இந்த சொற்பொருள் சுத்திகரிப்பு அடுக்கு இருதிசை-நீண்ட குறுகிய கால நினைவகத்தை (பிஐ-எல்எஸ்டிஎம்) கொண்டுள்ளது இது அதிக துல்லியமான மதிப்பெண்களை உருவாக்குகிறது. ஒட்டுமொத்தமாக சொற்பொருள் சுத்திகரிப்பு அடுக்குடன் (பி. இ. ஆர். டி) அடிப்படையிலான ஆழமான நரம்பியல் நெட்வொர்க் கட்டமைப்பை அறிமுகப்படுத்துவதன் மூலம் ஏ.

எஸ். ஏ. ஜி அமைப்பு குறுகிய பதில்களைப் புரிந்துகொள்வது வரையறுக்கப்பட்ட பயிற்சி தரவைக் கையாளுதல் மற்றும் தரப்படுத்தல் துல்லியத்தை மேம்படுத்துதல் தொடர்பான சவால்களை சமாளிக்க முடியும்.

முக்கிய வார்த்தைகள்: தானியங்கி குறுகிய பதில் தரப்படுத்தல் (ஏ. எஸ். ஏ. ஜி) இருதரப்பு என்கோடர் பிரதிநிதித்துவங்கள் டிரான்ஸ்ஃபார்மர்கள் (பி. இ. ஆர். டி) இருதிசை நீண்ட குறுகிய கால நினைவகம் நெட்வொர்க் (பிஐ-எல்எஸ்டிஎம்) நுண்ணறிவு பயிற்சி அமைப்புகள் உரை ஈடுபாடு ஆழமான கற்றல் ஆழமான நரம்பியல் நெட்வொர்க்.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT – ENGLISH	iv
	ABSTRACT – TAMIL	v
	TABLE OF CONTENTS	viii
	LIST OF FIGURES	xii
	LIST OF TABLES	xiii
	LIST OF ABBREVIATIONS	xiv
1	INTRODUCTION	01
2	LITERATURE SURVEY	04
3	PROBLEM STATEMENT	11
	3.1 Objectives	11
	3.2 Scope of the project	11
	3.3 System Description	12
4	REQUIREMENT ANALYSIS	13
	4.1 Functional Requirements	13
	4.1.1 Hardware Requirements	13
	4.1.2 Software Requirements	13

	4.2 Non-Functional Requirements	14
5	SYSTEM DESIGN	15
	5.1 System Architecture	15
	5.2 Module Description	16
	5.2.1 BERT encoding	16
	5.2.2 Bi-LSTM network	16
	5.2.3 Attention mechanism	17
	5.2.4 Max-pooling	17
	5.3 Algorithm Description	19
	5.3.1 BERT encoding	19
	5.3.2 Bi-LSTM network	19
	5.3.3 Attention mechanism	20
	5.3.4 Max-pooling	21
6	UML DIAGRAMS	22
	6.1 Use Case Diagram	22
	6.2 Class Diagram	23
	6.3 Sequence Diagram	24
	6.4 Collaboration Diagram	25
	6.5 State Chart Diagram	26

	6.6 Activity Diagram	27
	6.7 Deployment Diagram	28
	6.8 Component Diagram	29
	6.9 Package Diagram	30
	6.10 Data Flow Diagram	31
	6.10.1 DFD LEVEL-0	31
	6.10.2 DFD LEVEL-1	32
	6.10.3 DFD LEVEL-2	32
7	IMPLEMENTATION	34
	7.1 Datasets	34
	7.2 Technology used	35
8	TESTING	39
	8.1 System Testing	39
	8.1.1 Types of Testing	39
	8.1.1.1 Unit Testing	39
	8.1.1.2 Integration Testing	40
	8.1.1.3 Functional Testing	40
	8.1.1.4 Whitebox Testing	41
	8.1.1.5 Blackbox Testing	41

9	RESULTS AND DISCUSSIONS	42
	9.1 Evaluation measures	42
	9.1.1 Cohen’s Kappa	42
	9.1.2 Confusion matrix	44
	9.2 Performance analysis	46
10	CONCLUSION AND FUTURE WORKS	48
	10.1 Conclusion	48
	10.2 Future Works	48
	REFERENCES	49
	APPENDIX	53

LIST OF FIGURES

FIG.NO	FIGURE NAME	PAGE NO
5.1	System Architecture	15
6.1	Use Case Diagram	22
6.2	Class Diagram	23
6.3	Sequence Diagram	24
6.4	Collaboration Diagram	25
6.5	State Chart Diagram	26
6.6	Activity Diagram	27
6.7	Deployment Diagram	28
6.8	Component Diagram	29
6.9	Package Diagram	30
6.10.1	DFD LEVEL-0	31
6.10.2	DFD LEVEL-1	32
6.10.3	DFD LEVEL-2	32
7.1.1	Dataset	34
7.2.1	Token embedding	35
7.2.2	Tokenization	36
7.2.3	Bi-LSTM	36
7.2.4	Implementing	37

7.2.5	Maxpooling	37
7.2.6	Questions	38
7.2.7	Grading	38
9.1.2.1	Comparison on Precision and Recall	44
9.1.2.2	Confusion matrix	44
9.1.2.3	ASAG Accuracy	45

LIST OF TABLES

TABLE NO	TABLE NAME	PAGE NO
9.1.1.1	Cohen's kappa description	41
9.1.1.2	Comparison of grading	42
9.1.1.3	Values of consistency	42
9.1.2.1	Prediction type	43
9.1.2.2	Definition of parameters	43
9.2.1	Accuracy obtained from different models	46
9.2.2	Comparison on different models	46

LIST OF ABBREVIATIONS

S.NO	ABBREVIATIONS	DEFINITIONS
1.	ITS	Intelligent Tutoring System
2.	ASAG	Automatic Short Answer Grading
3.	BERT	Bidirectional Encoder Representations from Transformers
4.	LSTM	Long Short-Term Memory
5.	NLP	Natural Language Processing
6.	CNN	Convolutional Neural Networks
7.	DNN	Deep Neural networks
8.	ELMo	Embedding from language models
9.	GPT	Generative Pretrained Transformer
10.	UA	Unseen answers
11.	UQ	Unseen questions
12.	UD	Unseen domain

CHAPTER 1

INTRODUCTION

Automatic Short-Answer Grading using BERT model and Bi-LSTM network discusses the importance of adaptive testing and assessment in intelligent tutoring systems (ITSs) for personalized learning routes. It highlights the significance of capturing students' cognitive levels accurately to tailor educational experiences effectively. This work emphasizes the role of automatic short-answer grading (ASAG) in evaluating student responses efficiently within ITSs. An Intelligent Tutoring System (ITS) is a computer-based instructional system that provides personalized and adaptive instruction to learners, mimicking the guidance and feedback provided by a human tutor. Both students' answers and reference answers appear in the form of natural language, ASAG is regarded as an application of recognizing textual entailment in educational technology

The existing methods for ASAG still remain challenging mainly for two reasons. First, students usually use different free texts to answer the same question, in which students' answers may have significant differences in sentence structure, language style, and text length. Therefore, it is necessary to utilize advanced learning techniques to combine different deep neural networks in the ASAG task in order to achieve a deeper semantic understanding of students' answers. Second, the deep learning method for ASAG is fully supervised machine learning, the training corpus for ASAG is usually small in size, which is usually only a few thousand instead of the tens of thousands in conventional deep learning tasks. Therefore, training a stable and effective deep neural network model on a small corpus is a major challenge faced by the deep learning methods of ASAG.

The proposed framework that leverages Bidirectional Encoder Representations from Transformers (BERT)-based deep neural networks to enhance the grading accuracy of short answers. Bidirectional Encoder Representations from Transformers, is a natural language processing (NLP) model developed by Google. It utilizes a transformer architecture and is pre-trained on large amounts of text data to understand context and relationships between words. BERT is often used for various NLP tasks, such as text classification, question answering, and named entity recognition.

Additionally, a semantic refinement layer is constructed, incorporating a bidirectional Long Short-Term Memory (LSTM) network with position information to refine the semantics of BERT outputs. Long Short-Term Memory is a type of recurrent neural network architecture designed to address the vanishing gradient problem in traditional RNNs. LSTM has memory cells with gates that control the flow of information, allowing them to capture and store long-term dependencies in sequential data. This makes LSTMs particularly effective for tasks involving sequences, such as natural language processing, time series prediction, and speech recognition. The main contributions of this article can be summarized as follows.

We extend the application of the pretrained BERT model in ASAG tasks from a fine-tuning approach to a combination with bidirectional LSTM (Bi-LSTM). On top of the fine-tuned BERT model, we construct a semantic refinement layer to refine the semantics of BERT outputs, which consists of a Bi-LSTM network with position information. Overall, by introducing a BERT-based deep neural network framework with a semantic refinement layer, the ASAG system can overcome challenges related to understanding short answers, handling limited training data, and improving grading accuracy.

The primary motivation behind this research is to leverage advanced deep learning techniques, particularly BERT-based models, to automate the grading of short answers efficiently and accurately. By automating this process, the paper aims to improve the efficiency of grading tasks, ensure consistency in evaluation, scale grading capabilities to accommodate larger volumes of student responses, and enhance the overall accuracy of assessment. The motivation behind the work lies in addressing the challenges faced in grading short answers manually, which can be time-consuming, subjective, and prone to inconsistencies. By leveraging advanced deep learning techniques, such as BERT-based models, the paper aims to automate the process of grading short answers in educational settings, MOOCs, and online exam platforms.

The main contribution of the work involves developing and evaluating a novel BERT-based deep neural network model specifically tailored for automatic short-answer grading. The incorporation of a semantic refinement layer in the BERT-based model to deepen the understanding of answer texts and improve feature generalization abilities within the domain. This enhancement aims to promote a deeper semantic understanding of student responses and enhance the model's ability to generalize features for accurate grading. This model is designed to address the challenges associated with manual grading by providing a more objective and scalable solution for evaluating student responses in educational settings, MOOCs, and online exam platforms. Through the implementation and fine-tuning of the BERT model, the work aims to demonstrate the potential of deep learning techniques in revolutionizing the assessment process and advancing intelligent tutoring systems and automated grading mechanisms in educational environments.

CHAPTER 2

LITERATURE SURVEY

2.1. D. Alikaniotis, H. Yannakoudakis, and M. Rei, “Automatic text scoring using neural networks,” in Proc. 54th Annu. Meeting Assoc. Comput Linguistics, vol. 1, pp. 715–725, 2016.[24]

Automatic text scoring using neural networks involves leveraging artificial neural network architectures to assess the quality or score of written text. Here's a simplified overview of the process. Collect and preprocess a labeled dataset containing examples of text along with corresponding scores or grades. Convert the textual data into a suitable format for neural network input. This often involves techniques like tokenization and embedding to represent words or phrases numerically. Design a neural network architecture that can effectively capture the features and patterns in the input text data. This may include recurrent neural networks (RNNs), long short-term memory networks (LSTMs), or more advanced models like BERT. Integrate the trained neural network into the text scoring system, allowing it to automatically evaluate new or unseen text based on the learned patterns during training.

Merits: Neural networks can capture complex, non-linear relationships within the text, allowing them to model intricate patterns and dependencies that may be challenging for traditional scoring methods.

Demerits: Neural networks are often considered black-box models, making it difficult to interpret how the model arrives at a particular score. Lack of interpretability can be a concern in educational settings where transparency is crucial.

2.2. J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., pp. 4171–4186, 2019.[15]

The paper "BERT: Pre-training of deep bidirectional transformers for language understanding" by Jacob Devlin and his colleagues, published in 2018, introduces BERT (Bidirectional Encoder Representations from Transformers), a groundbreaking natural language processing (NLP) model. BERT is designed to capture bidirectional context for each word in a sentence, unlike previous models that only considered left-to-right or right to-left context. This bidirectional approach allows BERT to understand the relationships and dependencies between words in both directions. BERT is pre-trained on a massive amount of unlabeled text data. The pre-training objective involves predicting missing words within sentences, both masked words and randomly selected words. This unsupervised pre-training helps BERT learn contextual representations. This encourages the model to understand the context and relationships between the words.

Merits: BERT's bidirectional architecture captures context from both left and right directions, allowing it to better understand the nuances and relationships between words in a sentence. BERT generates contextualized word representations. The Transformer architecture used in BERT facilitates parallelization during training, making it computationally efficient and scalable.

Demerits: Training and fine-tuning BERT can be computationally intensive, requiring substantial resources. This may pose challenges for researchers or organizations with limited access to high-performance computing infrastructure.

2.3. C. N. Tulu, O. Ozkaya, and U. Orhan, “Automatic short answer grading with SemSpace sense vectors and MaLSTM,” IEEE Access, vol. 9, pp. 19270–19280, 2021.[4]

In Automatic short answer grading using semantic spaces and MaLSTM the "SemSpace" typically refers to a semantic space, which is a mathematical representation where words or phrases are placed based on their semantic similarity. Sense vectors may indicate that the model considers different senses or meanings of words. Utilizing semantic space and sense vectors helps capture the semantic relationships between words, enhancing the understanding of the content. MaLSTM is a model architecture that combines Matching Networks and Long Short-Term Memory (LSTM) networks. LSTM is effective for handling sequential data, and in the context of short answer grading, MaLSTM likely aims to capture the similarity or matching degree between the student's answer and a reference answer. The overall goal of the paper seems to be improving the automatic grading of short answers. Using SemSpace sense vectors suggests a focus on semantic understanding, enabling the model to recognize not only surface-level similarities but also semantic nuances in the responses.

Merits: This model captures the nuances and multiple senses of words in short answers. MaLSTM combines Matching Networks with LSTM, enabling the model to perform contextual matching. Improved Relevance. The use of semantic vectors and MaLSTM likely contributes to a more nuanced relevance assessment.

Demerits: Utilizing semantic vectors and MaLSTM may require significant computational resources, making the approach computationally intensive, especially during training and fine-tuning. Success in automatic grading often depends on the availability of large and diverse datasets for training.

2.4. M. Uto and Y. Uchida, “Automated short-answer grading using deep neural networks and item response theory,” in Proc. Int. Conf. Artif. Intell. Educ.,pp. 334–339, 2020.[7]

The combination of Deep Neural Networks (DNNs) and Item Response Theory (IRT) in Automatic Short Answer Grading (ASAG) represents a hybrid approach that leverages the strengths of both techniques. Here's an explanation of how these elements are typically integrated. DNNs are used for their capability to learn complex patterns and representations from data. In ASAG, DNNs can be employed to automatically extract features and representations from short answers. These neural networks are trained on a large dataset of short answers and corresponding grades. The initial layers of the DNN are responsible for feature extraction. These layers capture various aspects of the input short answers, learning to represent them in a high-dimensional feature space. This process is essential for capturing the nuanced information present in language.

Merits: Deep Neural Networks (DNNs) contribute to the model's ability to understand the semantic content of short answers, capturing contextual information and relationships between words. DNNs excel at automatically extracting relevant features from input data. In the context of Automatic Short Answer Grading (ASAG), this capability allows the model to discern important patterns and characteristics in student responses.

Demerits: The integration of Deep Neural Networks (DNNs) and Item Response Theory (IRT) can result in a computationally complex model, requiring substantial computational resources for both training and inference. The use of DNNs can make the model less interpretable. Tuning parameters for both the DNN and IRT components can be challenging.

2.5. H. Tan, C. Wang, Q. Duan, Y. Lu, and R. Li, “Automatic short answer grading by encoding student responses via a graph convolutional network,” Interact. Learn. Environ., vol.pp. 1–15, 2020.[8]

In this work each word or phrase in the student response becomes a node in the graph. The relationships between these nodes, such as co-occurrence or semantic connections, are modeled as edges. A Graph Convolutional Network employs an embedding layer to convert the nodes (words or phrases) into high-dimensional vectors, capturing their semantic meaning. Convolutional Operations: GCN applies convolutional operations on the graph to aggregate information from neighboring nodes, allowing the model to understand the contextual relationships between words. Scoring and Classification: After the convolutional operations, the model generates a representation of the student response. This representation is then used for scoring or classification tasks, determining the correctness or quality of the answer. Training on Labeled Data: The GCN model is trained on a dataset with labeled examples, where the correct grades or classifications are provided for various student responses. This enables the model to learn the patterns and features associated with correct or incorrect answers.

Merits: GCNs excel at capturing contextual relationships within student responses, allowing for a more nuanced comprehension of the meaning and coherence in answers.

Demerits: GCNs heavily rely on labeled data for training. In cases where there's limited or biased training data, the model's performance may be hindered. Implementing and training GCNs can be computationally expensive and resource-intensive, particularly when dealing with large datasets or complex graph structures.

2.6. N. Seuzen, A. N. Gorban, J. Levesley, and E. M. Mirkes, “Automatic short answer grading and feedback using text mining methods,” *Procedia Comput. Sci.*, vol. 169, no.pp. 726–743, 2020.[9]

Automatic short answer grading and feedback using text mining methods involves the application of computational techniques to analyze and assess students' written responses. The student's written response is subjected to text mining techniques, including preprocessing steps like removing stop words, stemming, or lemmatization, to prepare the text for analysis. Relevant features, such as word frequencies, n-grams, or other linguistic patterns, are extracted from the processed text. These features serve as input for the grading model. The trained model is applied to grade new student responses. It classifies the answers into categories indicating correctness or quality, providing an automated grading score. Based on the grading results, tailored feedback is generated for the student. Text mining identifies specific strengths and weaknesses in the response, allowing for personalized and constructive feedback.

Merits: Automation through text mining allows for swift grading of numerous short answers, significantly reducing the time and effort required by educators. Text mining methods provide a consistent and standardized approach to grading, subjective variations in assessment across different graders. The automated system can efficiently handle a large volume of student responses.

Demerits: Text mining methods may struggle with handling ambiguous language or varied interpretations, leading to challenges in accurately grading answers that lack clarity. Grading short answers requires a nuanced understanding of natural language, and text mining models may struggle with the complexity of language nuances, idioms, or context.

2.7 CONSOLIDATED SURVEY

The paper “Automatic text scoring using neural networks [2.1]” by D. Alikaniotis, H. Yannakoudakis, and M. Rei can capture complex, non-linear relationships within the text, but there is a lack of interpretability. The paper "BERT: Pre-training of deep bidirectional transformers for language understanding [2.2]" by Jacob Devlin and his colleagues, published in 2018, introduces BERT, It generates contextualized word representations, enabling it to capture the varying meanings of words based on their context in a sentence. The paper published by C. N. Tulu, O. Ozkaya, and U. Orhan, “Automatic short answer grading with SemSpace sense vectors and MaLSTM [2.3]” is computationally intensive, especially during training and fine-tuning. The paper by M. Uto and Y. Uchida, “Automated short-answer grading using deep neural networks and item response theory[2.4]” is can be challenging for tuning parameters for both the DNN and IRT components. The paper by H. Tan, C. Wang, Q. Duan, Y. Lu, and R. Li, “Automatic short answer grading by encoding student responses via a graph convolutional network[2.5]” there is limited or biased training data, the model's performance may be hindered. The paper by N. Seuzen, A. N. Gorban, J. Levesley, and E. M. Mirkes, “Automatic short answer grading and feedback using text mining methods[2.6]” may struggle with handling ambiguous language or varied interpretations, leading to challenges in accurately grading answers that lack clarity. The above-mentioned deep learning methods achieve grading and scoring in an end-to-end manner, but they require a large amount of labeled corpus for training their model, which is what most ASAG corpora lack. To solve this problem, various pretrained transfer learning models, such as embeddings from language models (ELMo), BERT, generative pretrained transformer (GPT), and GPT-2, are applied to the ASAG task. Among them, BERT is especially outstanding and has achieved state-of-the-art grading results.

CHAPTER 3

PROBLEM STATEMENT

The existing Automatic Short Answer Grading models may suffer from a lack of deep understanding because of data scarcity. Additionally, such models may not give high precision scores.

3.1 OBJECTIVES

- To overcome the problem of data scarcity, the Bi-directional Encoder Representations from Transformers (BERT) model is incorporated, which understands the context of the sentences effectively.
- To generate high precision scores, the semantics of BERT output needs to be refined, which is done by incorporating the Bidirectional Long Short-term Memory (Bi-LSTM) network.

3.2 SCOPE OF THE PROJECT

- **Online Learning platforms:** This BERT-based model automates short-answer grading, provides immediate feedback, and enables personalized learning experiences, enhancing engagement and learning outcomes for students on online platforms.
- **Competitive Examinations:** The model ensures fair evaluation by automating grading processes, reducing manual effort and biases, while its accuracy and scalability efficiently handle large exam volumes, maintaining assessment integrity and delivering prompt results.

- **Educational Institutions:** Institutions benefit from streamlined grading tasks, saving educators time to provide targeted student support, while the model's data-driven insights help track student progress, identify learning gaps, and tailor instructional strategies for improved academic performance.
- **Language Learning Apps:** Leveraging the model, language apps assess proficiency levels, offer detailed feedback, and provide personalized learning recommendations, enhancing language comprehension and supporting learners in mastering linguistic skills through interactive exercises.

3.3 SYSTEM DESCRIPTION

The system model proposed in this work “Automatic Short-Answer Grading” integrates a BERT-based deep neural network framework with bidirectional LSTM (Bi-LSTM) network to automate and enhance the process of short-answer grading. By dynamically encoding answer texts using the BERT model and refining semantics through the Bi-LSTM network, the system achieves a deep understanding of student responses, leading to accurate and context-aware grading. Leveraging transfer learning and pre-trained models, the system overcomes the limitations of small labeled corpora in ASAG tasks and demonstrates state-of-the-art performance on benchmark datasets. Designed for end-to-end grading, the system streamlines the evaluation process in educational environments, offering a scalable and efficient solution for evaluating student responses in intelligent tutoring systems, MOOCs, and online exam platforms.

CHAPTER 4

REQUIREMENT ANALYSIS

Requirements are the basic needs or constraints which are required to develop a system. Requirements are broadly classified as user requirements and the system requirements. These requirements can be collected while designing the system. The two major classifications of requirements are software and the hardware requirements.

1. Functional Requirements

2. Non-Functional Requirements

4.1 FUNCTIONAL REQUIREMENTS

Functional requirements specify which output file should be produced from the given file then describe the relationship between the input and output of the system, for each functional requirement a detailed description of all data inputs is their source and the range of valid inputs must be specified.

4.1.1 HARDWARE REQUIREMENTS

- Processor - Intel core i5 or AMD x86-64 processor
- Memory - 32 GB
- Hard disk - 8 GB

4.1.2 SOFTWARE REQUIREMENTS

- Framework - TensorFlow and Django
- Libraries - Hugging Face Transformers
- Operating System - Windows 10 or above

4.2 NON-FUNCTIONAL REQUIREMENTS

Non-Functional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands) or accuracy (i.e., how precise are the systems numerical answers). It describe user-visible aspects of the system that are not directly related with the functional behavior of the system.

Performance

- The system should demonstrate high performance in terms of speed and accuracy in grading short answers, providing timely feedback to users.

Scalability

- The model should be able to handle a large volume of short-answer responses efficiently, ensuring scalability to accommodate varying levels of usage.

Reliability

- The model should be reliable and consistent in its grading capabilities, minimizing errors and ensuring trustworthy assessment results.

Usability

- The system should be user-friendly, with an intuitive interface for educators and students to interact with the grading platform easily.

Compatibility

- The system should be compatible with various learning management systems, online platforms, and devices to ensure widespread accessibility.

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

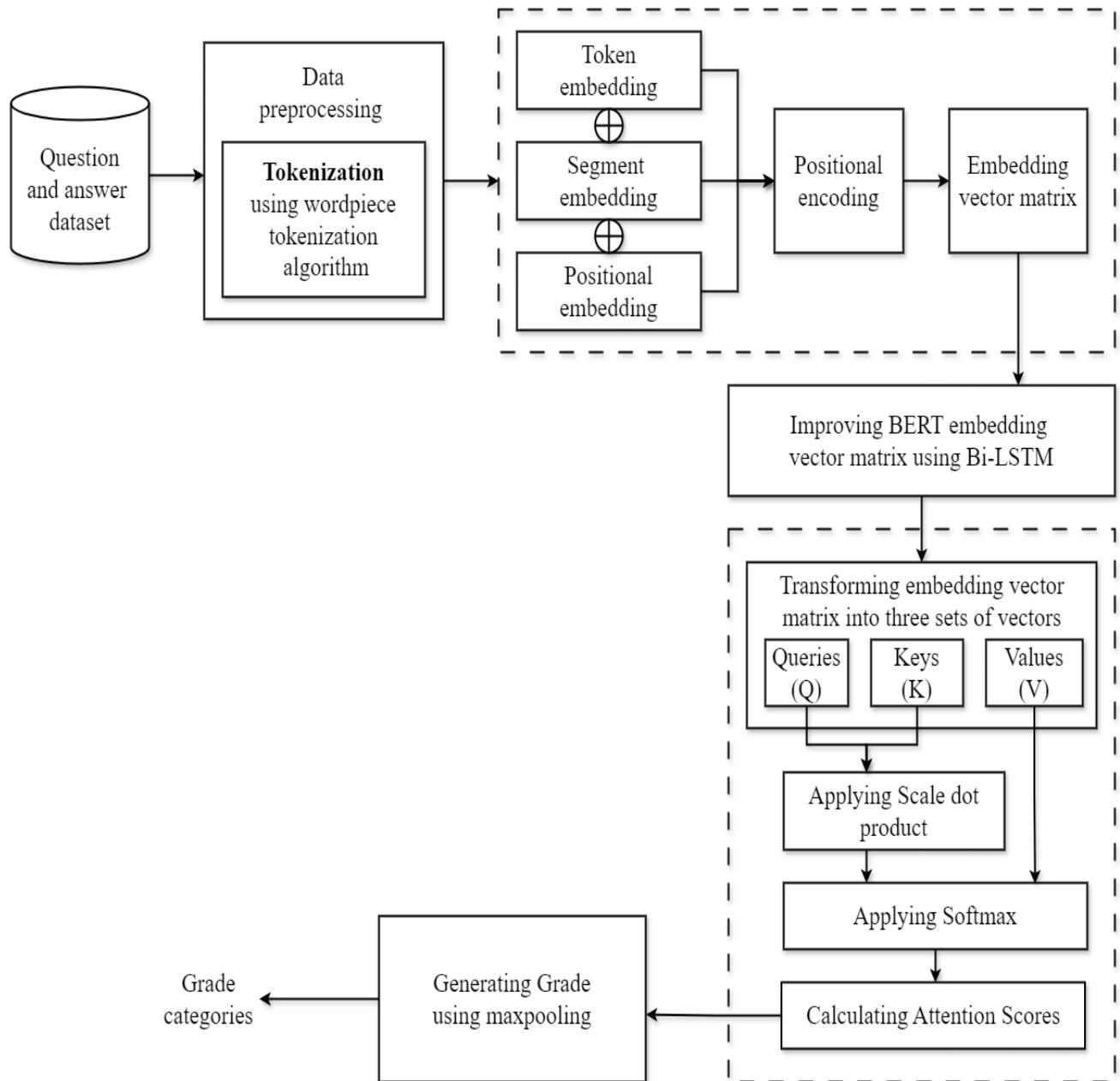


Fig. 5.1 System Architecture

5.2 MODULE DESCRIPTION

- BERT encoding
- Bi-LSTM network
- Attention mechanism
- Max-pooling

5.2.1 BERT ENCODING

The process involves initializing all parameters of the BERT model and then jointly fine-tuning these parameters along with other layers in the model. The input sequence for the fine-tuned BERT layer consists of a sentence pair comprising the student's answer and the reference answer. The input to a BERT model is tokenized text, which is converted into embeddings which include token embeddings, segment embedding, positional embedding and positional encoding. Token embeddings represent the meaning of individual words or sub words as a vector. Positional encodings encode the position of tokens in the sequence using sinusoidal functions to enable understanding of sequential order. It learns to represent a token's position in a sequence as a vector which is known as a hidden state. For example, the position of the first token in a sequence is represented by a (learned) vector $\mathbf{Wp}[:, 1]$, the position of the second token is represented by another (learned) vector $\mathbf{Wp}[:, 2]$, etc. The purpose of the positional embedding is to allow a Transformer to make sense of word ordering; in its absence the representation would be permutation invariant and the model would perceive sequences as “bags of words” instead.

Input: Tokenized texts

Output: Embedding vector matrix

5.2.2 Bi-LSTM NETWORK

In the semantic refinement layer of the ASAG system, a bidirectional Long Short Term Memory (LSTM) network with position information is employed to refine the semantics of the BERT outputs. The Bi-LSTM network extracts fine global context for the BERT outputs to the hidden states of the BERT model. By combining these components and utilizing complex gate structures in the Bi-LSTM network improves the hidden states of the BERT output. The aim is to enhance the system's ability to understand and evaluate short answers accurately. At the end of each sub layer, layer normalization is applied to normalize the activations.

Input: Embedding vector matrix.

Output: Improved vector matrix.

5.2.3 ATTENTION MECHANISM

After receiving the output from the Bi-LSTM network, the attention layer in a model like BERT processes this output to capture important contextual information and relationships within the sequence. This process is achieved by using a multi-head attention mechanism, The contributions of these contexts to the fused semantics are automatically adjusted by a weight matrix. After this combination process, the fused semantic representation is processed by layer normalization before being fed into the max-pooling layer for further evaluation. The attention layer plays a crucial role in integrating different levels of contextual information to enhance the system's ability to evaluate short answers accurately.

Input: Improved vector representations of primary and context sequence.

Output: Updated representations of tokens.

5.2.4 MAX-POOLING

In the max-pooling layer of the ASAG system, a max-pooling operation is performed on the semantic representation generated by the attention mechanism of the BERT model. During the max-pooling operation in the ASAG system, the semantic representation is processed to extract the most relevant and significant features that are crucial for evaluating the student's answer. In the Prediction Layer, a max-pooling operation is performed on the updated representation of tokens to obtain the final semantic representation Z for the answer pair (q, p) . The max-pooling operation selects the maximum value from the fused semantic representation across a specific dimension or window. By selecting the maximum value from the set of values in the representation, the max-pooling operation helps in capturing the most salient information that contributes to the overall understanding and assessment of the answer pair (question, student's answer). To calculate the probability of each grade category, Z is fed into a linear transformation followed by a softmax function. The output of the max-pooling operation is the final semantic representation Z . The max-pooling operation helps in capturing the most relevant and significant features from the fused semantic representation for further processing and grading of the short answer pair. The result of the max pooling operation is a final semantic representation. Finally, grade categories are generated for student's answer using a regression task.

Input: Updated representation of tokens.

Output: Predicted probability of grade category.

5.3 ALGORITHM DESCRIPTION

5.3.1 BERT POSITIONAL ENCODING ALGORITHM

Input: $l \in [l_{\max}]$, position of a token in the sequence.

Step 1: Extract the positional encoding of a token's position in a sequence.

Step 2: Takes a position l and a positional encoding W_p as inputs.

$$W_p[2i-1, t] = \sin(t/l_{\max}^{2i/d_e}) \quad (1)$$

$$W_p[2i, t] = \cos(t/l_{\max}^{2i/d_e}), \quad (2)$$

Step 3: Extracting the column vector corresponding to the given position l .

Step 4: Return e_p as the output, representing the positional embedding vector matrix for the token at position l .

Output: $e_p \in \mathbb{R}^{d_e}$, the embedding vector-matrix representation of the position.

5.3.2 BIDIRECTIONAL-LSTM ALGORITHM

Input: e_p as Embedding vector matrix

Step 1: Calculate the forget gate activation vector:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (3)$$

Step 2: Calculate the input gate activation vector:

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i) \quad (4)$$

Step 3: Calculate the candidate cell state vector:

$$\tilde{q}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (5)$$

Step 4: Update the cell state:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{q}_t \quad (6)$$

Step 5: Calculate the output gate activation vector:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (7)$$

Step 6: Calculate the new hidden state vector

$$h_t = o_t \odot \tanh(c_t) \quad (8)$$

Output: h_t as Improved vector matrix.

5.3.3 ATTENTION MECHANISM ALGORITHM

Input: $X \in \mathbb{R}^{d_x \times l_x}$, $Z \in \mathbb{R}^{d_z \times l_z}$, vector representations of primary and context sequence.

Parameters: W_{qkv} consisting of:

$$W_q \in \mathbb{R}^{d_{attn} \times d_x}, b_q \in \mathbb{R}^{d_{attn}}$$

$$W_k \in \mathbb{R}^{d_{attn} \times d_z}, b_k \in \mathbb{R}^{d_{attn}}$$

$$W_v \in \mathbb{R}^{d_{out} \times d_z}, b_v \in \mathbb{R}^{d_{out}},$$

Hyperparameters: $\text{Mask} \in \{0,1\}^{l_z \times l_x \uparrow (3)}$

Step 1: Compute Query, Key, and Value Matrices such as Q, K and V.

$$Q \leftarrow W_q X + b_q \mathbf{1}^T \quad [\text{Query} \in \mathbb{R}^{d_{attn} \times l_x}] \quad (9)$$

$$K \leftarrow W_k Z + b_k \mathbf{1}^T \quad [\text{Key} \in \mathbb{R}^{d_{attn} \times l_z}] \quad (10)$$

$$V \leftarrow W_v Z + b_v \mathbf{1}^T \quad [\text{Value} \in \mathbb{R}^{d_{out} \times l_z}] \quad (11)$$

Step 2: Compute the scale dot product of K and the transpose of Q to get the attention scores matrix S, representing the similarity between queries and keys.

$$S \leftarrow K^T Q \quad [\text{Score} \in \mathbb{R}^{l_z \times l_x}] \quad (12)$$

Step 3: Apply the softmax function to the rows of S scaled by attention to compute the attention weights.

Step 4 : To compute the output multiply the attention weights by V to get the output matrix which aggregates information from the context sequence based on attention weights.

$$\mathbf{X}^{(h)} = V \cdot \text{softmax}(S/\sqrt{d_{attn}}) \quad (13)$$

Output: $\mathbf{X}^{(h)} \in \mathbb{R}^{d_{out} \times l_x}$, updated representations of tokens in \mathbf{X} , folding in information from tokens in \mathbf{Z} .

5.3.4 MAX-POOLING ALGORITHM

Input: $\mathbf{X}^{(h)}$ as updated representation of tokens.

Step 1: Perform a max-pooling operation on the $\mathbf{X}^{(h)}$ attention scores.

$$\mathbf{Z} = \text{Max-pooling}(\mathbf{X}^{(h)}) \quad (14)$$

Step 2: To calculate the probability of each grade category, \mathbf{Z} is fed into a linear transformation followed by a softmax function.

$$\mathbf{G} = \mathbf{MZ} + \mathbf{b} \quad (15)$$

$$P(y/Z) = \frac{\exp(G_y)}{\sum_i \exp(G_i)} \quad (16)$$

where \mathbf{M} is the representation matrix of grade categories, \mathbf{b} is a bias vector, G_y is the number of grade categories, \mathbf{G} is the vector of scores.

Output: $P(y/Z)$ denotes the predicted probability of grade category y .

CHAPTER 6

UML DIAGRAMS

6.1 USE CASE DIAGRAM

Use case diagrams depict the functionality of a system from the user's perspective by showing the different use cases or tasks that the system can perform and the actors that interact with the system to achieve those tasks. They help to ensure that all of the requirements of the system are captured and understood.

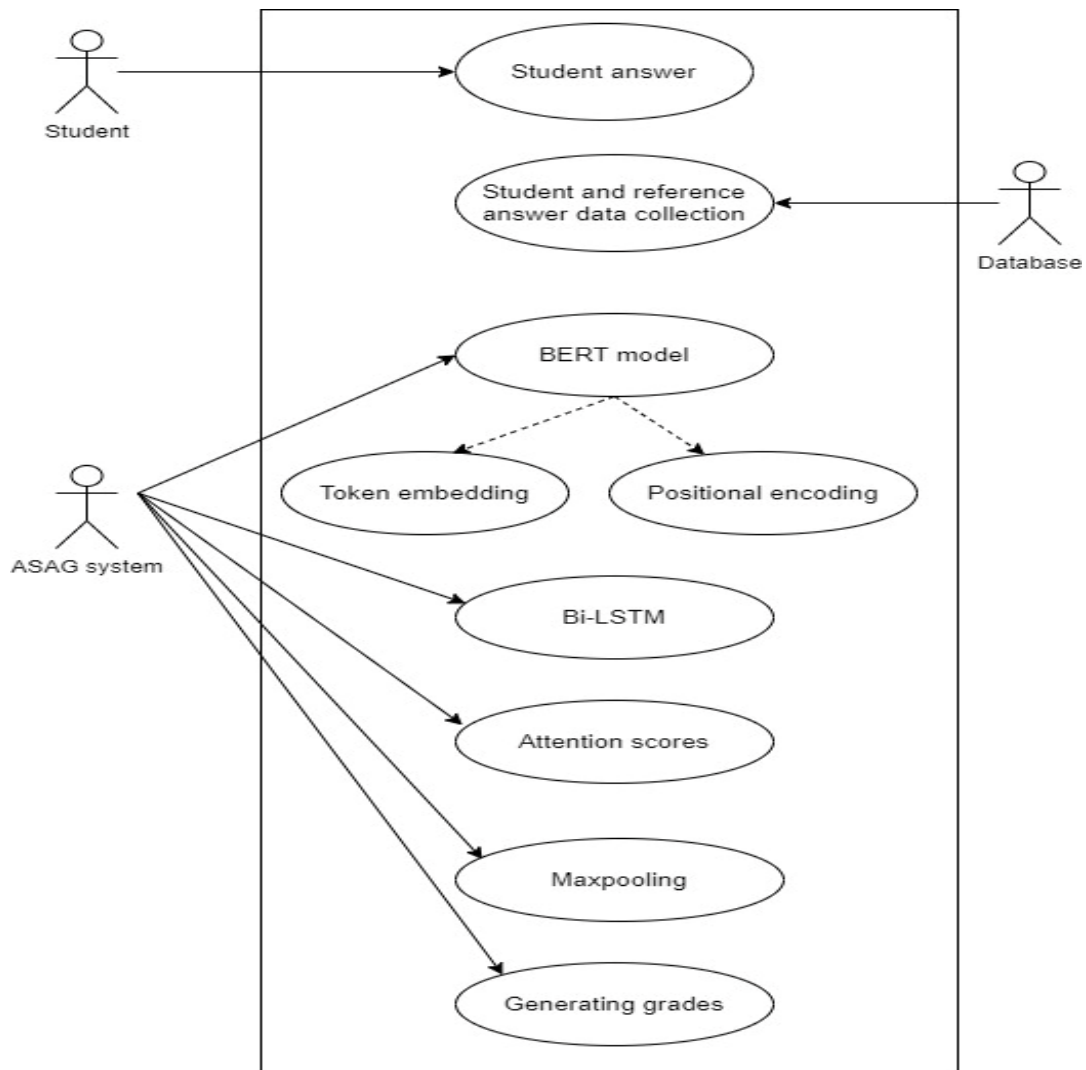


Fig. 6.1 Use Case Diagram

6.2 CLASS DIAGRAM:

Class diagrams depict the structure of a system by showing the classes, their attributes, and the relationships between them. They can help developers to understand the relationships between different objects in the system and how data flows between them.

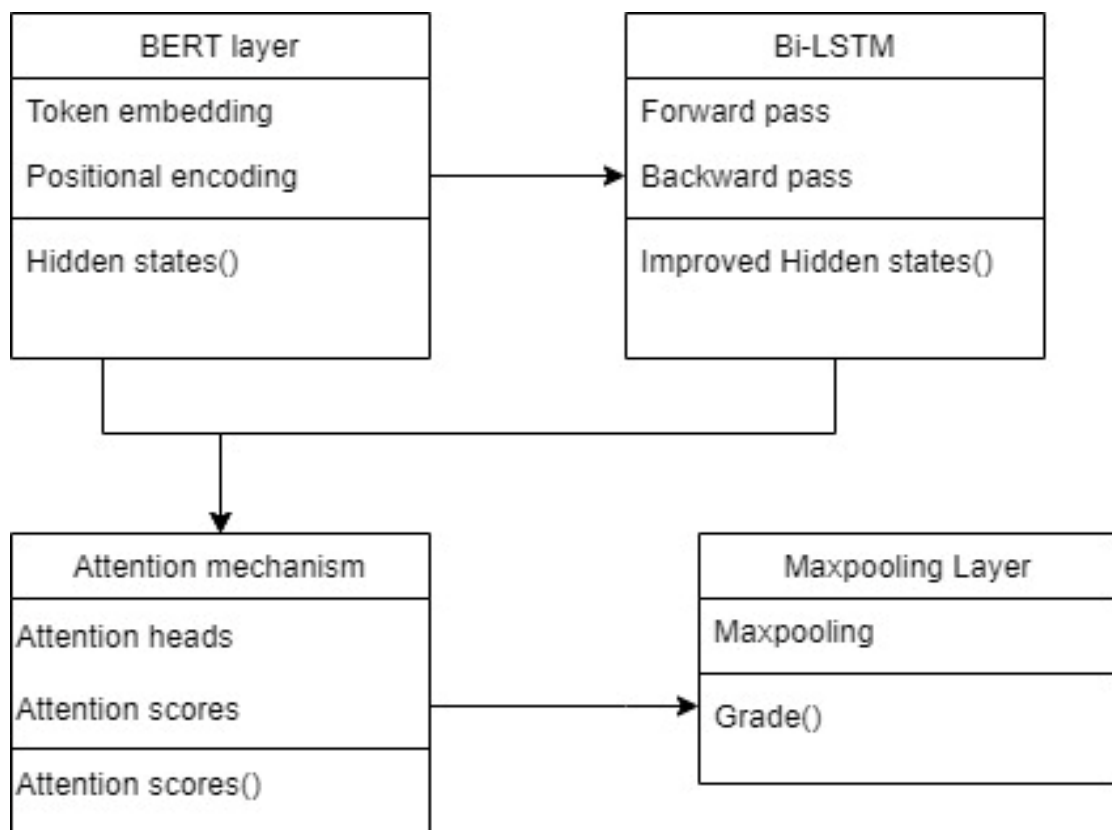


Fig. 6.2 Class Diagram

6.3 SEQUENCE DIAGRAM:

Sequence diagrams depict the interactions between objects in a system over time, showing the order in which messages are sent and received. They can help developers to understand the behavior of the system during different scenarios and identify potential issues.

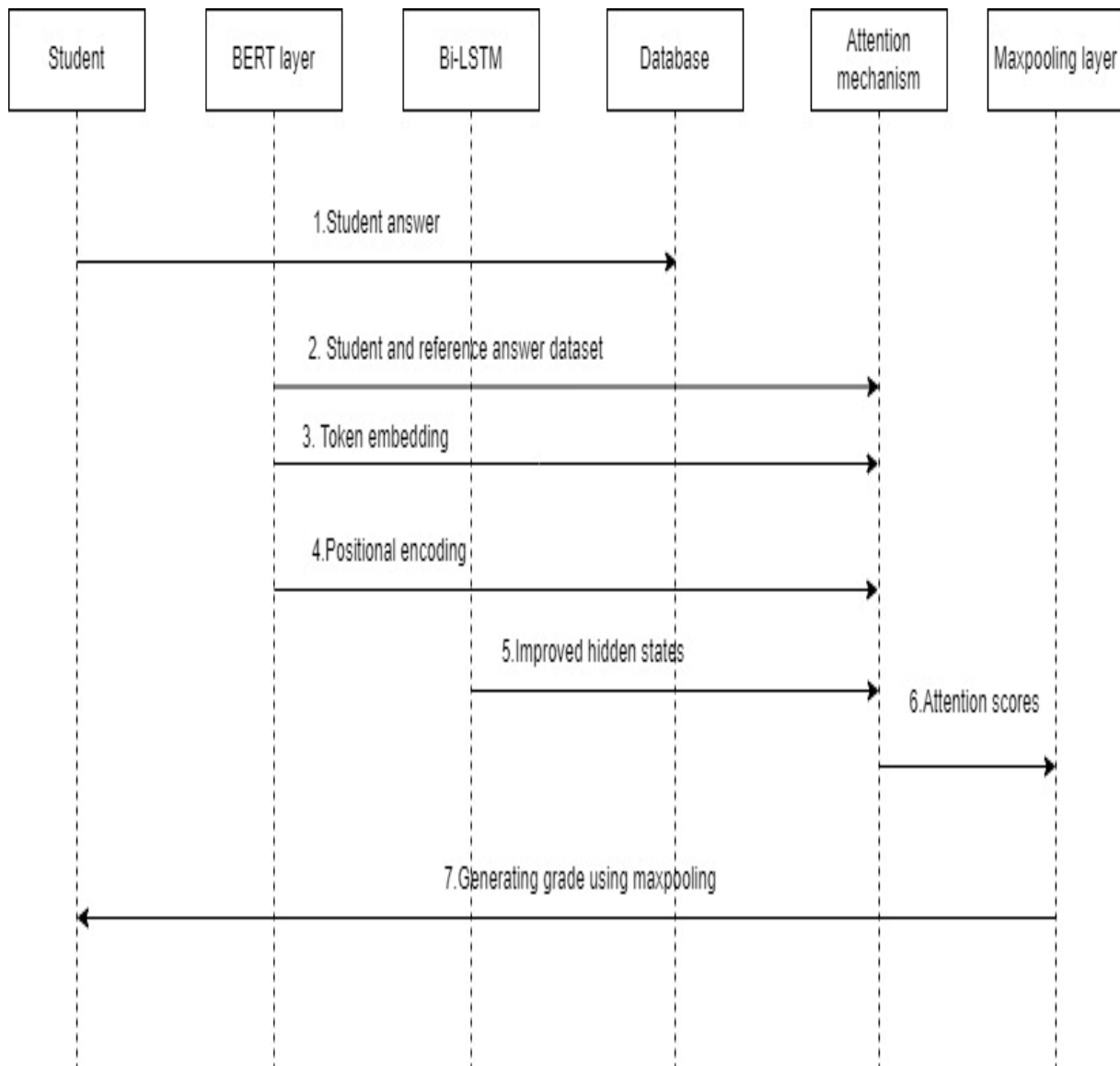


Fig. 6.3 Sequence Diagram

6.4 COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is a type of UML (Unified Modeling Language) diagram that shows the interactions between objects or parts of a system to achieve a specific task or function. It is used to model the dynamic behavior of a system by illustrating the messages exchanged among objects or parts during the execution of a use case scenario.

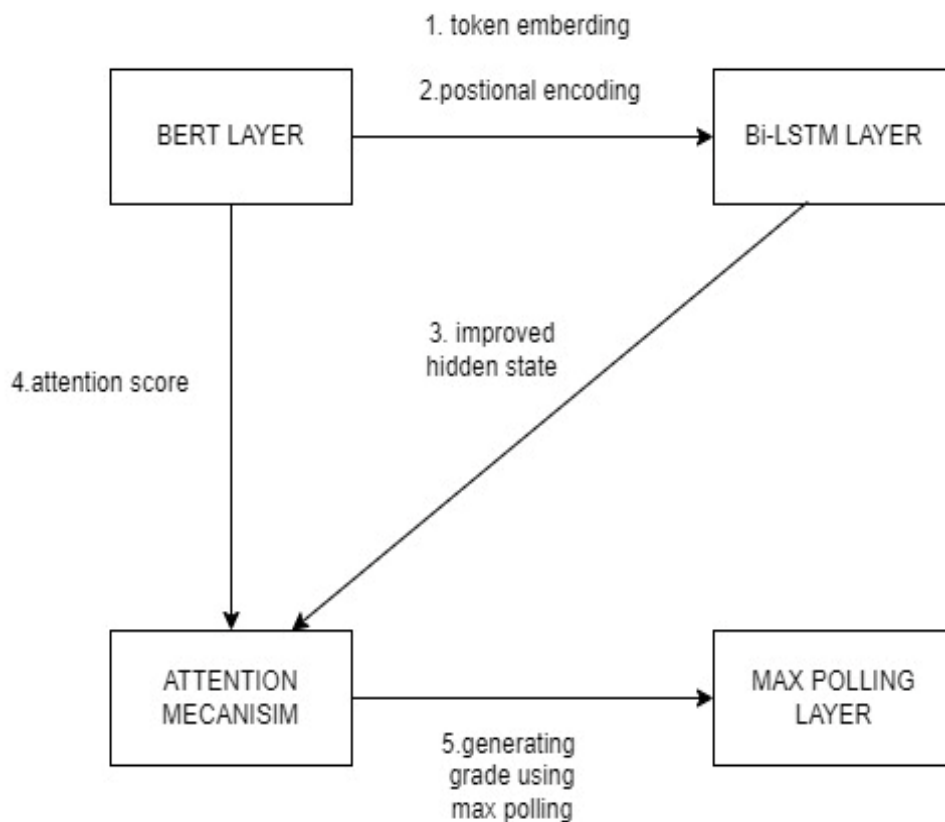


Fig. 6.4 Collaboration Diagram

6.5 STATECHART DIAGRAM

State diagrams depict the states and transitions of objects in a system, showing how objects behave in response to events. They can help developers to understand the behavior of the system during different states and identify potential issues.

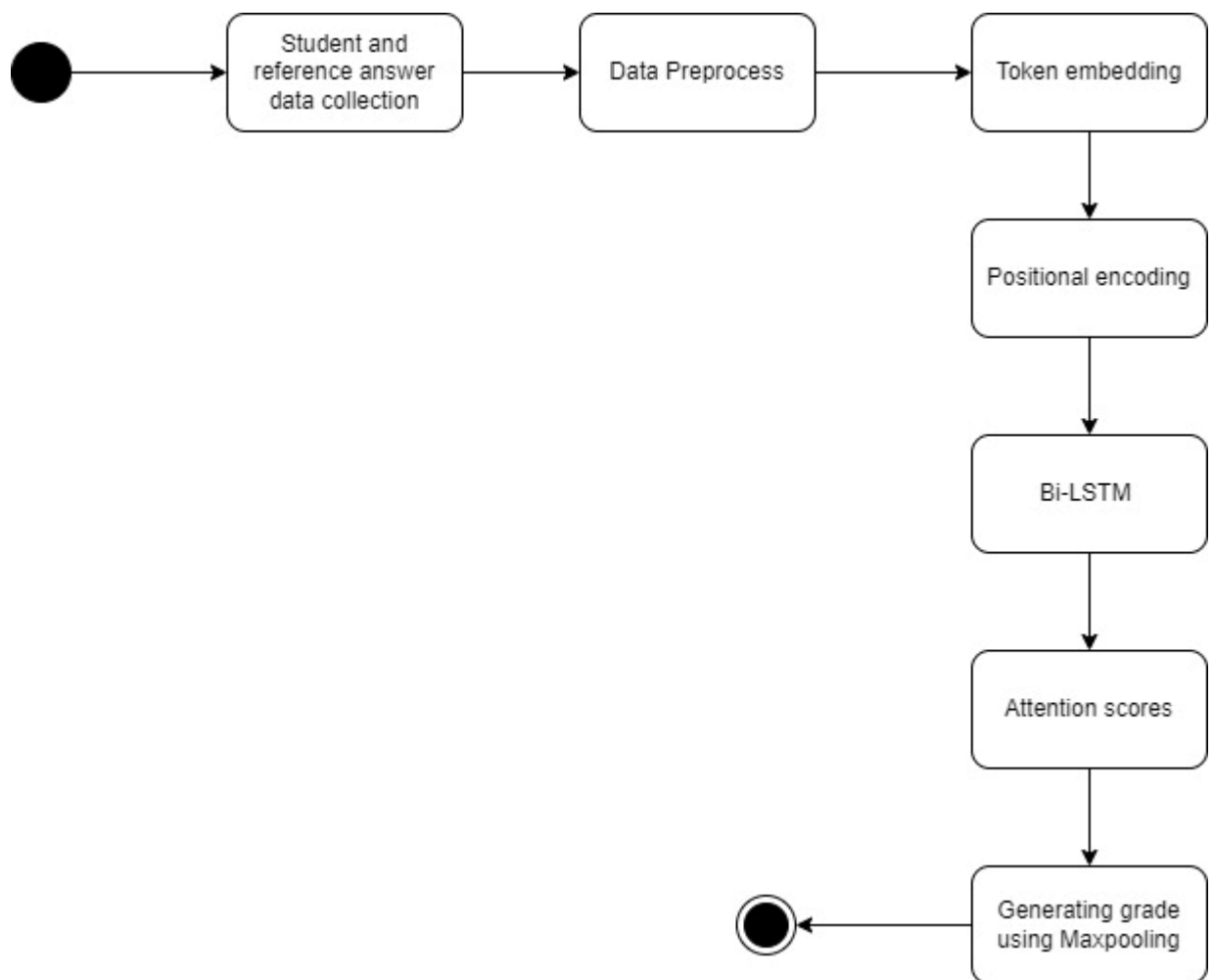


Fig. 6.5 State Chart Diagram

6.6 ACTIVITY DIAGRAM

Activity diagrams depict the flow of control in a system by showing the activities and decisions involved in a process. They can help developers to understand the various steps involved in a process and identify potential bottlenecks or inefficiencies.

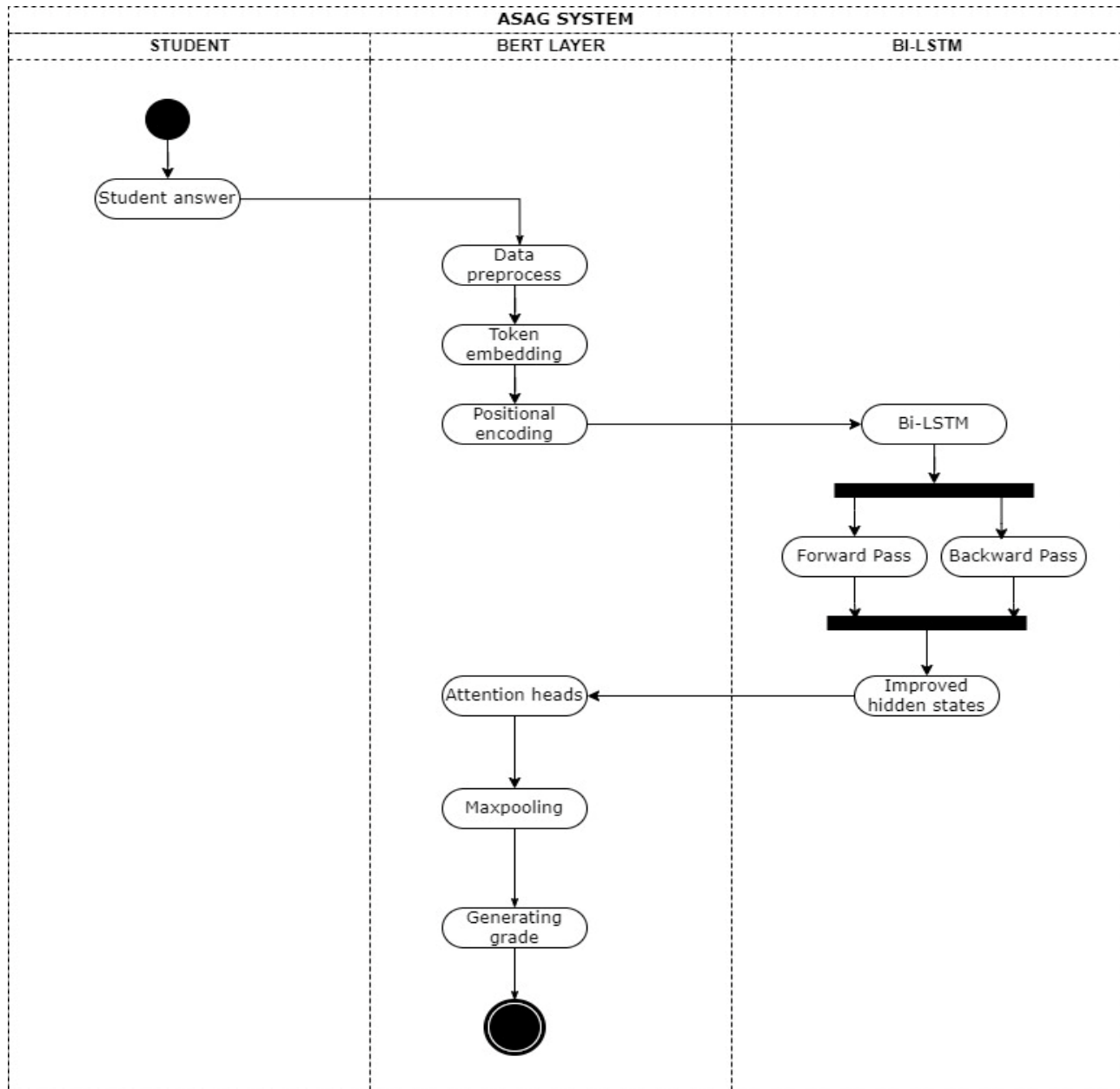


Fig. 6.6 Activity Diagram

6.7 DEPLOYMENT DIAGRAM

Deployment diagrams depict the physical deployment of a system's components on hardware or software platforms. They can help developers to understand how the system is physically deployed and identify potential issues with hardware or software configurations.

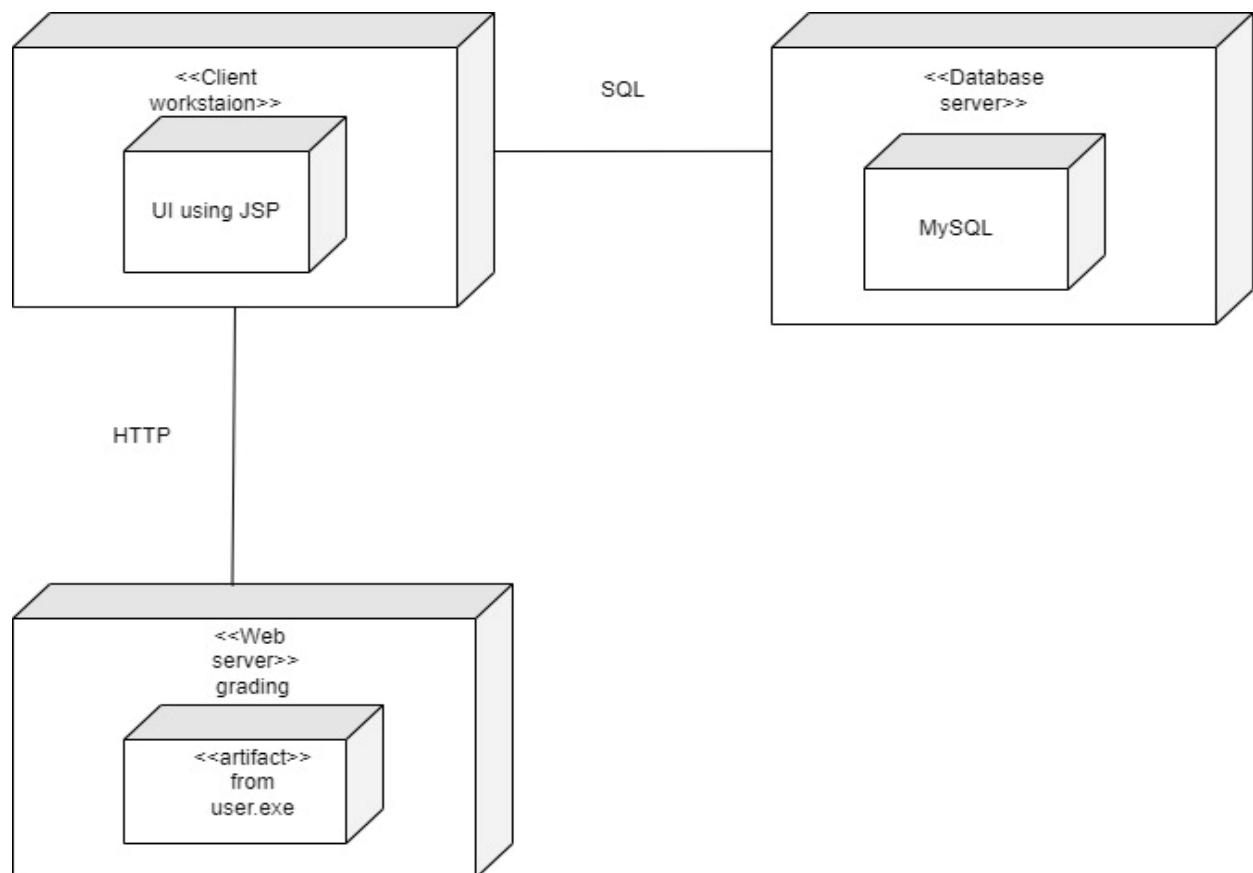


Fig. 6.7 Deployment Diagram

6.8 COMPONENT DIAGRAM

Component diagrams depict the physical and logical components of a system and the relationships between them. They can help developers to understand how the various components of the system work together and how changes to one component may impact others.

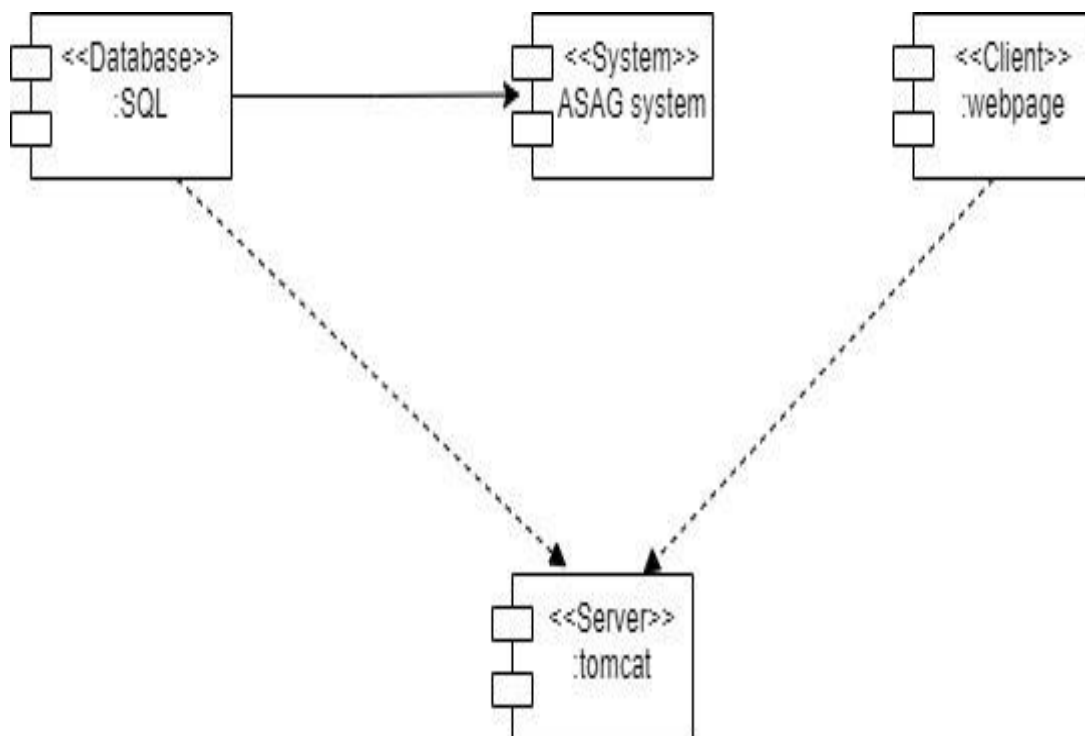


Fig. 6.8 Component Diagram

6.9 PACKAGE DIAGRAM

A package diagram is a type of UML (Unified Modeling Language) diagram that shows the organization and dependencies among packages in a system. A package is a container for related classes, interfaces, and other packages, and is used to organize and structure the software system.

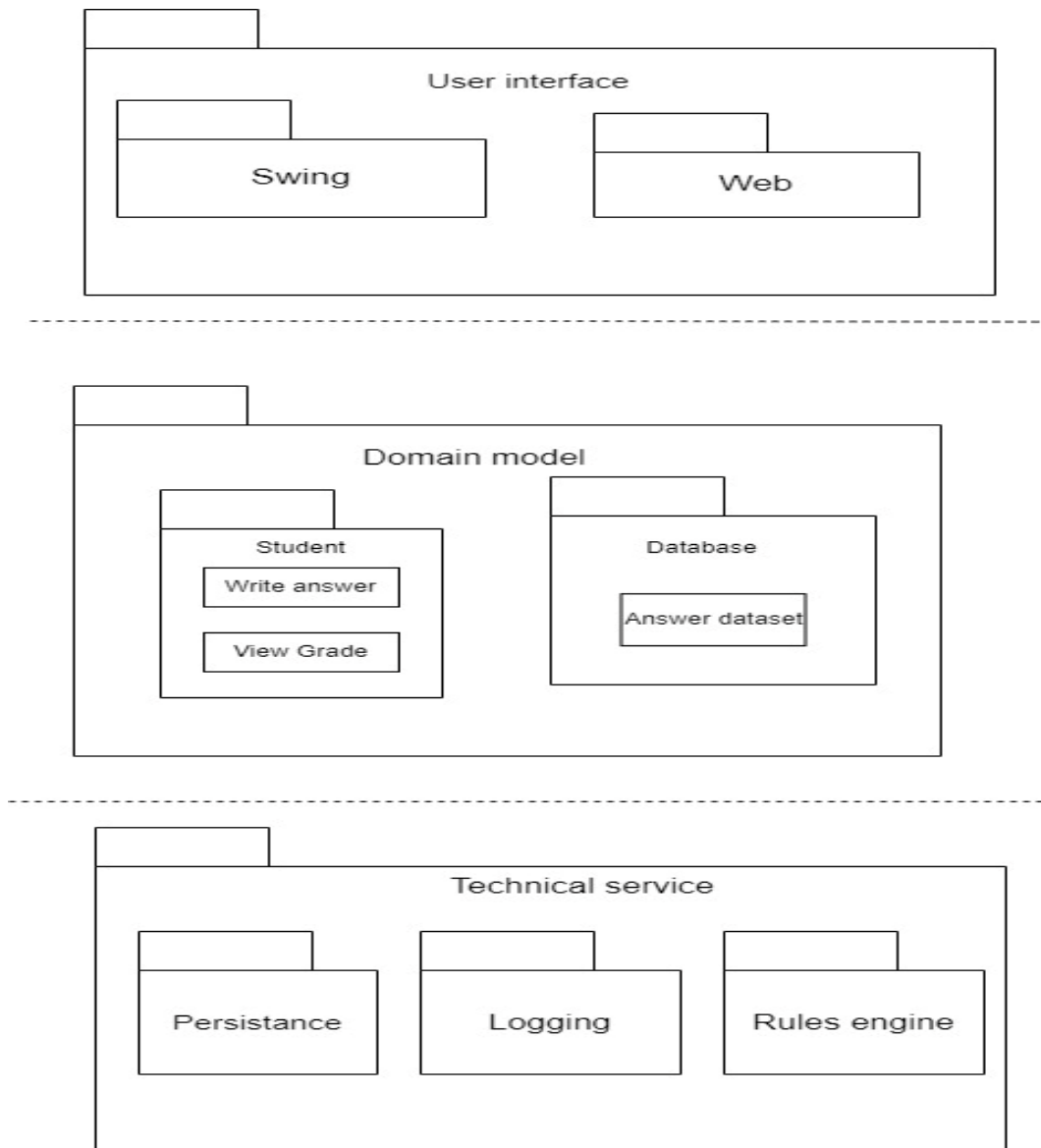


Fig. 6.9 Package Diagram

6.10 DATA FLOW DIAGRAM (DFD)

A data flow diagram (DFD) is a graphical representation of the ‘flow’ of a data through an information system. It differs from the flowchart as it shows the data flow instead of the control flow of the program. A data flow diagram can also be used for the visualization of data processing. DFD is designed to show how a system is divided into smaller portions and to highlight the flow of data between those parts. DFD is an important technique for modeling a system's high level detail by showing how input data is transformed to output results through a sequence of functional transformations.

6.10.1 DATA LEVEL-0

A context diagram is a top level (also known as “Level 0”) data flow diagram. It only contains one process node (“Process 0”) that generalizes the function of the entire system in relationship to external entities. There is only one process in the system and all the data flows either into or out of this process. Context level DFD demonstrates the interactions between the process and external entities. They do not contain data stores. When drawing context level DFD we must first identify the process, all the external entities and all the data flows. DFD-Level 0 shows the overview of the ontology-based reasoning engine module. It takes contextual information or contextual attributes as an input. This module uses semantic-based information retrieval model algorithm to produce the default actions as output.

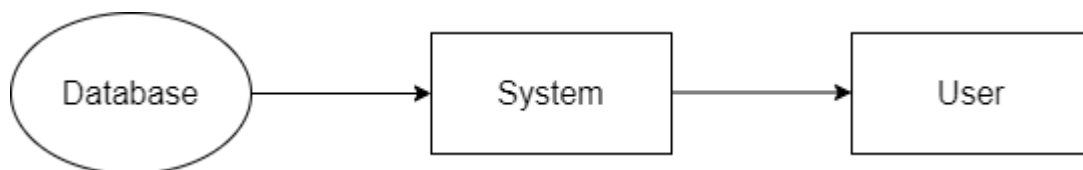


Fig. 6.10.1 DFD LEVEL - 0

6.10.2 DATA LEVEL-1

A DFD-level 1 is more detailed than a DFD-level 0 but not as detailed as a DFD-level 2. DFD-level 1 takes recommended default actions as an input by using semantic-based matching algorithm to produce the APIs of the operational systems/Web services of the relevant response agencies and resource providers based on their jurisdictions, rules, policies.

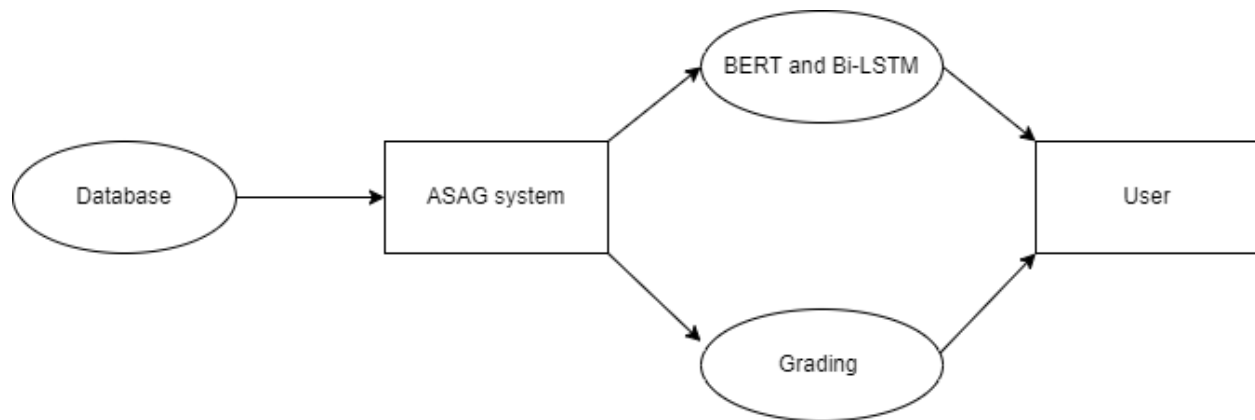


Fig. 6.10.2 DFD LEVEL-1

6.10.3 DATA LEVEL-2

A DFD-level 2 offers a more detailed look at the processes that make up an information system than a DFD-level 1 does. It can be used to plan or record the specific makeup of a system. DFD-level 2 gives more detailed look about the module. It takes recommended relevant web services of the APIs of response 36 organizations as input. This module uses the find service composition and reachability analysis algorithm to generate workflow of the response process. Then it shows the resource availability status to the user.

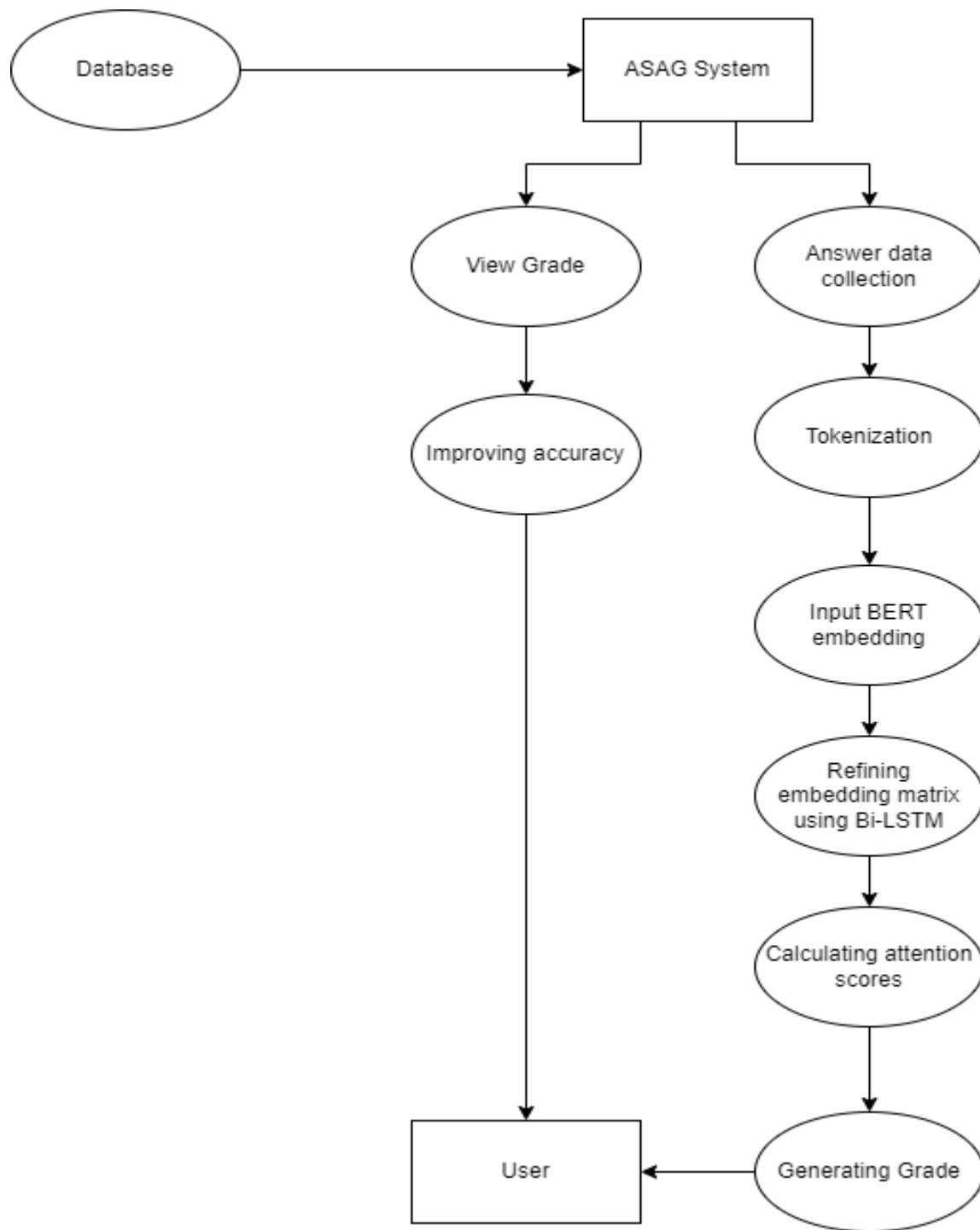


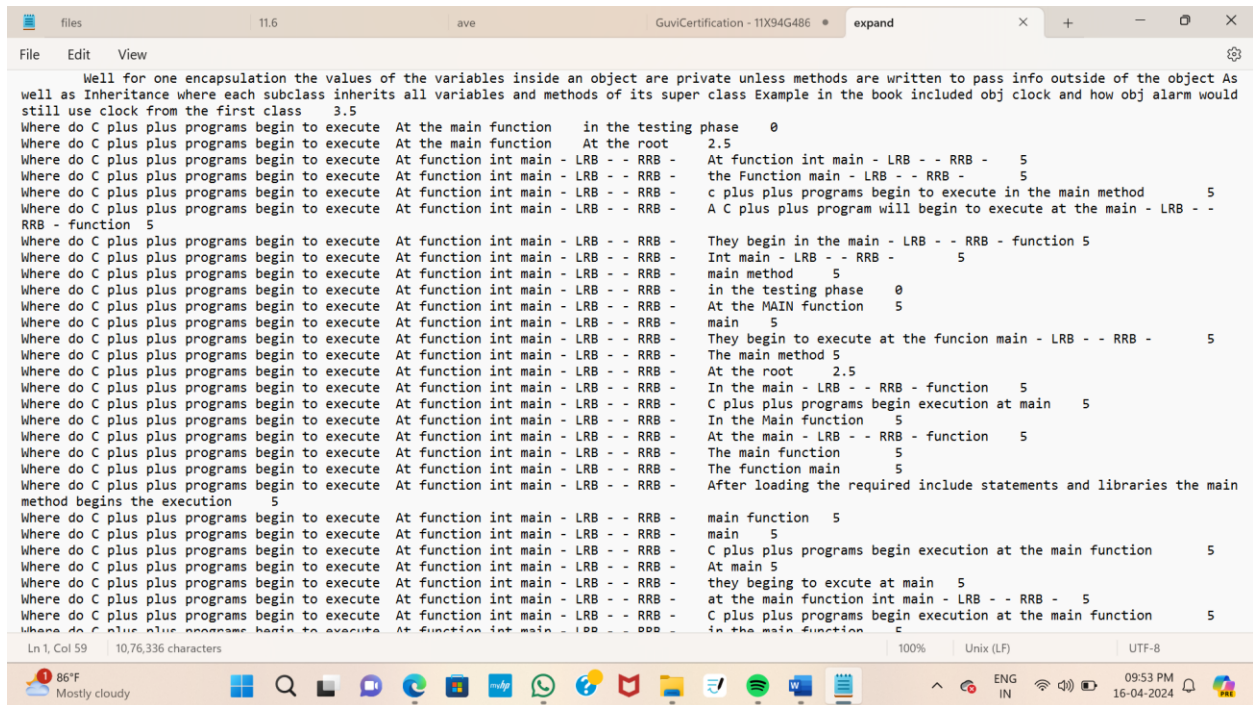
Fig. 6.10.3 DFD LEVEL-2

CHAPTER 7

IMPLEMENTATION

7.1 Datasets

The dataset we used in our experiments was the Short Answer Scoring SemEval-2013 dataset and Mohler dataset (2011). We used the SciEntsBank corpus in the SemEval-2013 dataset. SciEntsBank corpus contains approximately 10 000 answers to 197 assessment questions in 15 different science domains. This corpus is a benchmark for the ASAG classification task.



The screenshot shows a code editor window with a file named 'expand'. The text inside is a log of C++ program execution, detailing the flow from the main function to various sub-functions and methods. The log includes timestamps and function names, such as 'At the main function', 'At the root', 'At function int main', 'At the MAIN function', 'main', 'The main method', 'In the testing phase', 'At the function main', 'C plus plus programs begin execution at the main function', 'At main', 'they beging to excute at main', 'at the main function int main', and 'C plus plus programs begin execution at the main function'. The log is organized into columns, with the first column containing the function name and the second column containing the timestamp. The text is repeated multiple times, suggesting a sequence of events or a loop in the execution process.

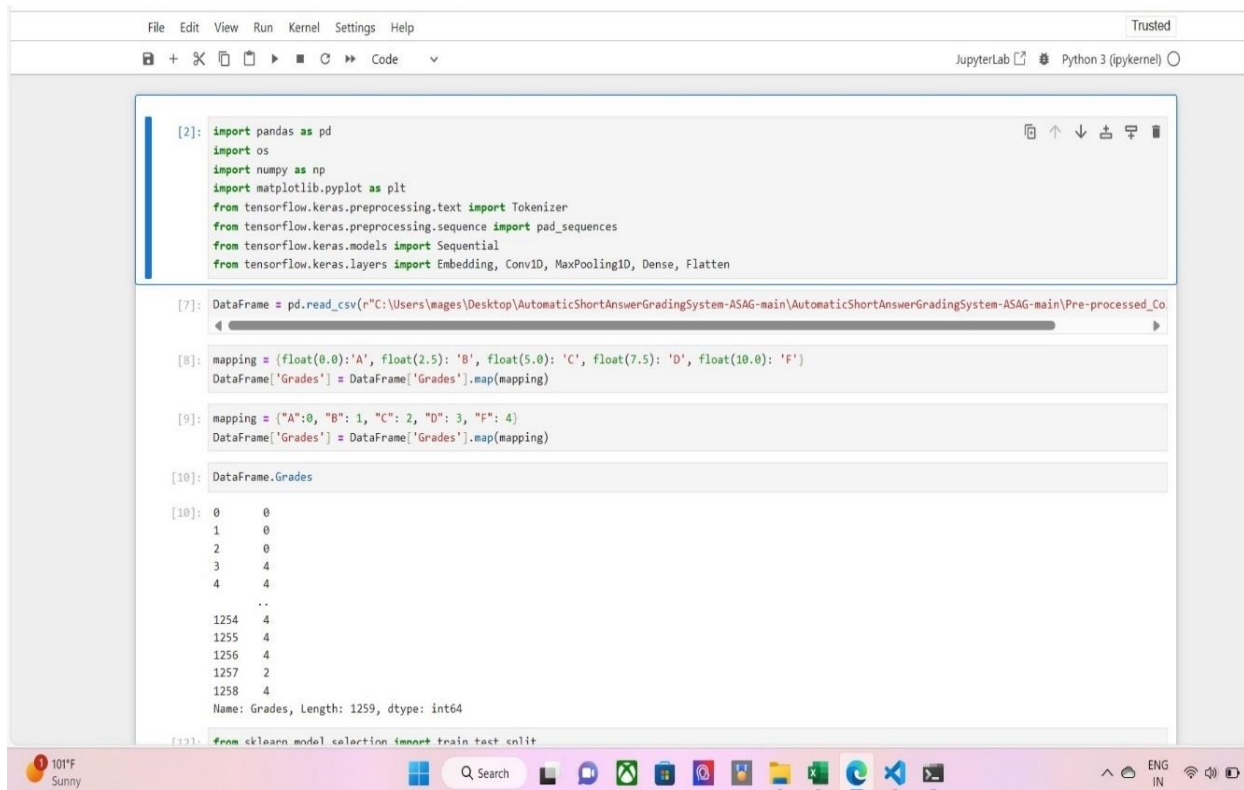
Fig. 7.1.1 Dataset

A computer science short-answer dataset were created by Mohler et al. from ten assignments and two exams of an introductory computer science class at the University of North Texas.

It contains 80 questions with 2273 students' answers. Each student's answer was scored by two teachers on an integer ranging from 0 to 5. We took the average of two labeled scores as the true score of the students' answer, resulting in 11 scoring grades ranging from 0 to 5 with 0.5 intervals. The questions, the reference answers, the students' answer to the questions, and their respective grades form the main components of the dataset.

7.2 Technology used

Python is a popular high-level, interpreted programming language known for its simplicity, readability, and versatility. The codes are all executed in the python programming language.



```
[2]: import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Dense, Flatten

[7]: DataFrame = pd.read_csv(r"C:\Users\mages\Desktop\AutomaticShortAnswerGradingSystem-ASAG-main\AutomaticShortAnswerGradingSystem-ASAG-main\Pre-processed_Co

[8]: mapping = {float(0.0): 'A', float(2.5): 'B', float(5.0): 'C', float(7.5): 'D', float(10.0): 'F'}
DataFrame['Grades'] = DataFrame['Grades'].map(mapping)

[9]: mapping = {"A":0, "B": 1, "C": 2, "D": 3, "F": 4}
DataFrame['Grades'] = DataFrame['Grades'].map(mapping)

[10]: DataFrame.Grades

[10]: 0      0
1      0
2      0
3      4
4      4
...
1254   4
1255   4
1256   4
1257   2
1258   4
Name: Grades, Length: 1259, dtype: int64

[11]: from sklearn.model_selection import train_test_split
```

Fig 7.2.1 Token embedding

```

[12]: tokenizer.word_index

[12]: {'null': 1,
      'the': 2,
      'is': 3,
      'in': 4,
      'of': 5,
      'a': 6,
      'and': 7,
      'you': 8,
      'to': 9,
      'for': 10,
      'how': 11,
      'data': 12,
      'class': 13,
      'are': 14,
      'what': 15,
      '2': 16,
      'trees': 17,
      'that': 18,
      'one': 19,
      'missing': 20,
      'information': 21,
      'vectors': 22,
      'fill': 23,
      'when': 24,
      'do': 25,
      'classifier': 26,
      'two': 27,
      'decision': 28,
      'name': 29,
      'used': 30,
      'explain': 31,
      ...

[17]: train_sequences = tokenizer.texts_to_sequences(train_text_data)
      test_sequences = tokenizer.texts_to_sequences(test_text_data)

```

Fig 7.2.2 Tokenizer

```

1  #!/usr/bin/env python
2  import os
3  import sys
4
5  if __name__ == '__main__':
6      os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
7      try:
8          from django.core.management import execute_from_command_line
9      except ImportError as exc:
10         raise ImportError(
11             "Couldn't import Django. Are you sure it's installed and "
12             "available on your PYTHONPATH environment variable? Did you "
13             "forget to activate a virtual environment?"
14         ) from exc
15     execute_from_command_line(sys.argv)
16

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Please use 'rate' instead of 'keep_prob'. Rate should be set to 'rate = 1 - keep_prob'.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 1, 300)	721200
lstm_2 (LSTM)	(None, 64)	93440
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 814,705
Trainable params: 814,705
Non-trainable params: 0

2024-05-03 11:40:24.425375: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not

Fig 7.2.3 Bi-LSTM

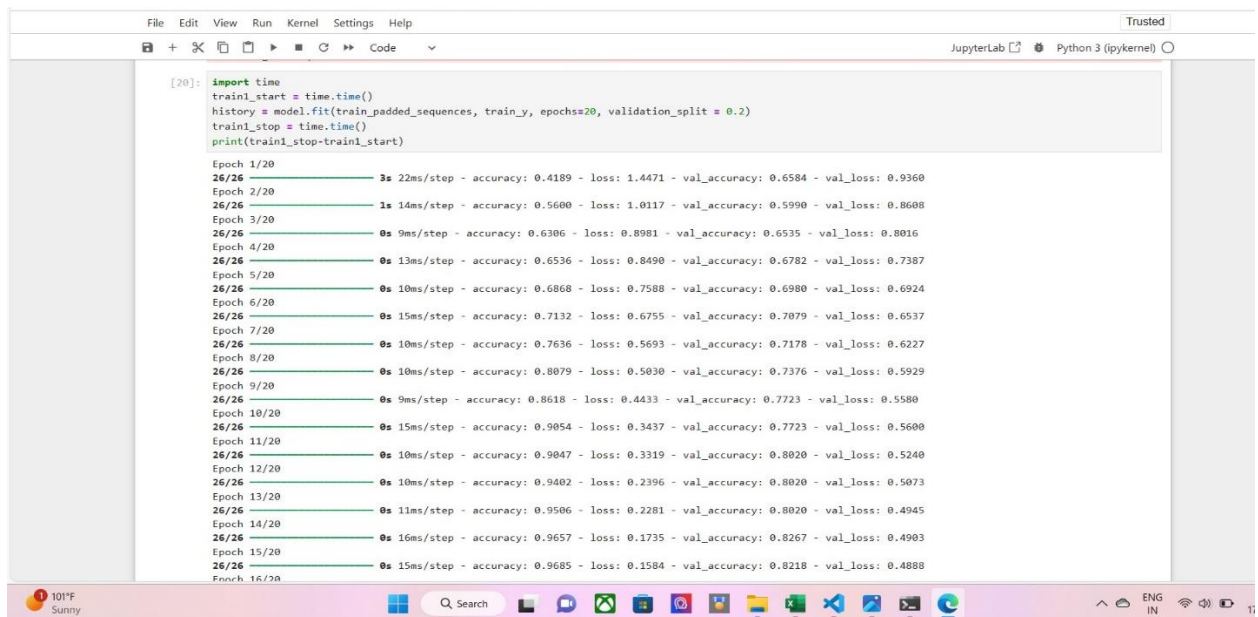


Fig 7.2.4 Implementing

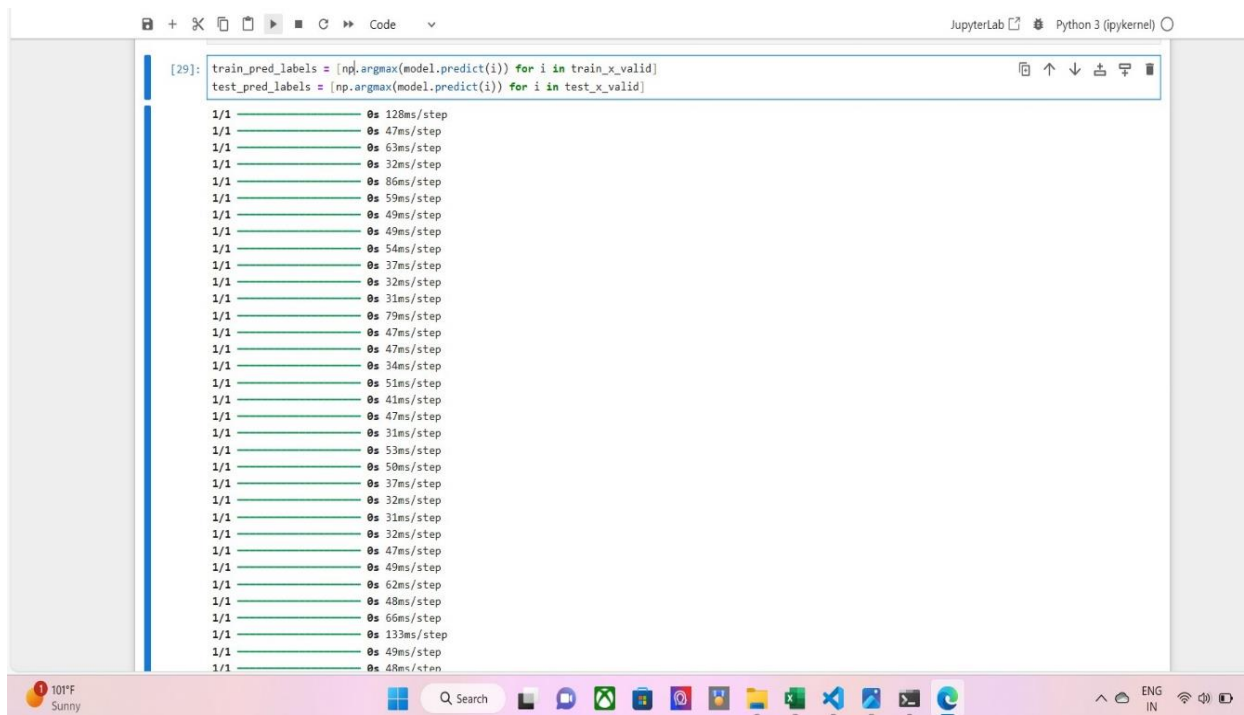


Fig 7.2.5 Maxpooling

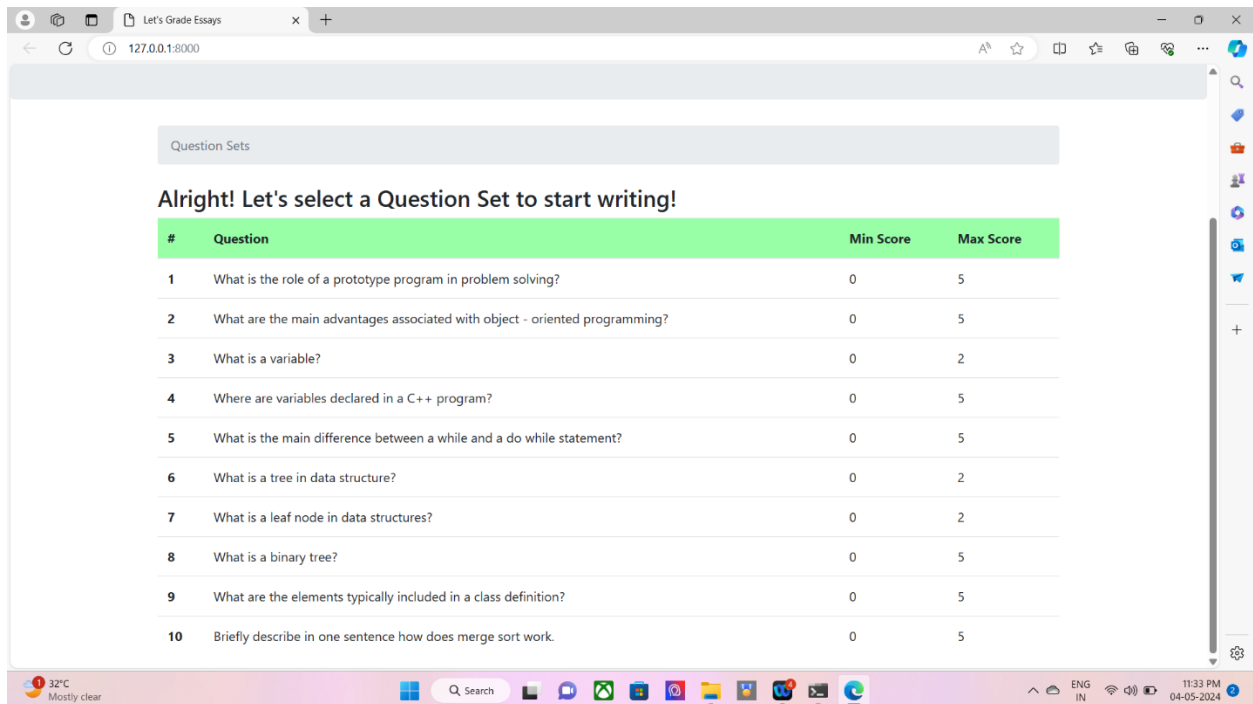


Fig 7.2.6 Questions

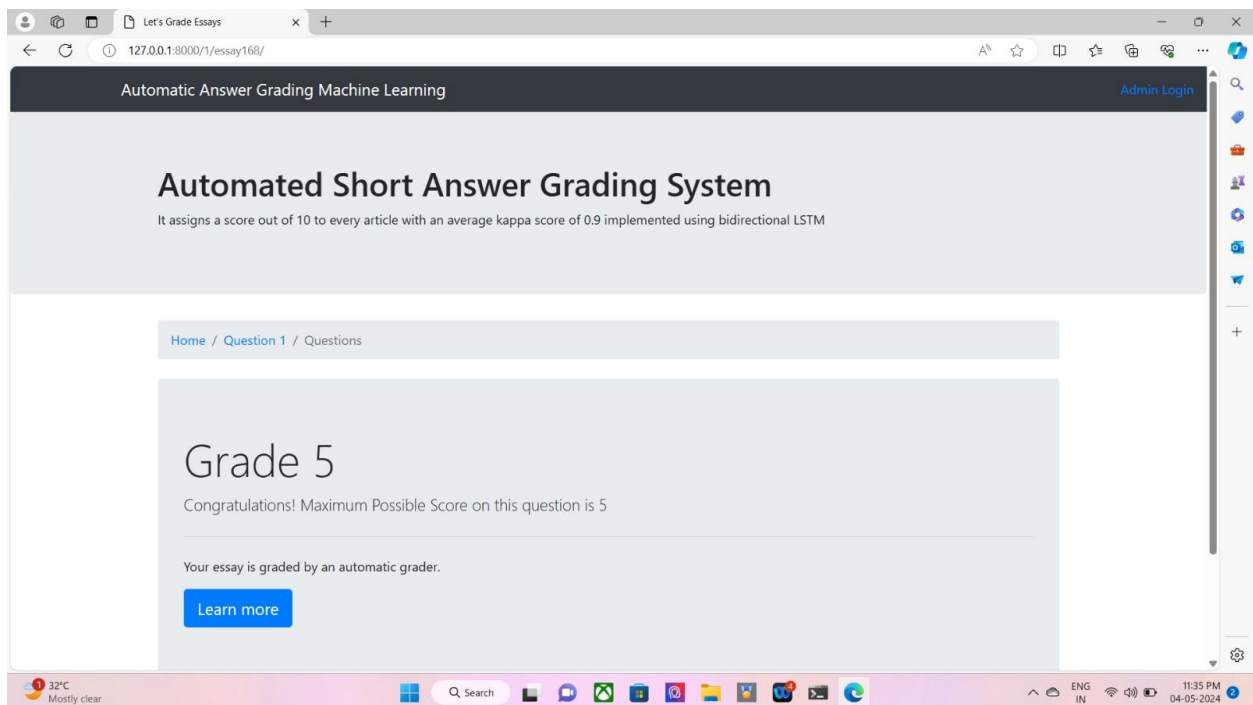


Fig 7.2.7 Grading

CHAPTER 8

TESTING

8.1 System Testing

System testing for the Enhanced Automatic Short Answer Grading System is a comprehensive phase that focuses on evaluating the entire system's functionality and performance. This phase involves the following key activities:

- Verifying that the system can accurately interpret and grade a diverse set of short answers.
- Testing the system's response to various input scenarios to ensure consistent and reliable grading results.
- Assessing the system's scalability and performance under different loads to determine its efficiency in handling multiple grading requests simultaneously.
- Conducting end-to-end testing to validate the integration of the BERT model, Bi-LSTM network, and grading algorithm in the system.

8.1.1 Types of Testing

8.1.1.1 Unit Testing

Unit testing in this project involves testing individual components of the system in isolation to ensure their functionality and correctness. Key aspects of unit testing include:

- Designing test cases to validate the behavior of the BERT model and Bi-LSTM network independently.
- Verifying that each unit performs its specific grading tasks accurately and efficiently.
- Conducting boundary value analysis and equivalence partitioning to test the units under different input conditions.

8.1.1.2 Integration testing

Integration testing focuses on testing the interaction between the BERT model and Bi-LSTM network when combined in the grading process. Key activities in integration testing include:

- Verifying that the components communicate effectively and exchange data seamlessly during the grading process.
- Testing the integration points to ensure that the system functions as a cohesive unit.
- Conducting regression testing to validate that changes in one component do not adversely affect the overall system performance.

8.1.1.3 Functional Testing

Functional testing is crucial for validating that the system's grading functions align with the specified requirements and criteria. Key aspects of functional testing include:

- Designing test cases based on the system's functional specifications and user requirements.
- Verifying that the system accurately interprets and evaluates short answers based on semantic understanding and context.
- Testing the system's feedback mechanisms to ensure that users receive meaningful and actionable feedback on their responses.

8.1.1.4 White Box Testing

White box testing involves examining the internal logic and structures of the system to ensure its correctness and efficiency. Key activities in white box testing include:

- Conducting code reviews and inspections to identify potential coding errors or inefficiencies.
- Performing static code analysis to detect vulnerabilities and improve code
- Implementing code coverage analysis to ensure that all code paths are tested and validated.

8.1.1.5 Black Box Testing

Black box testing evaluates the system's functionality from an external perspective without knowledge of its internal workings. Key aspects of black box testing include:

- Designing test cases based on the system's external behavior and specifications.
- Conducting equivalence partitioning and boundary value analysis to test the system's input and output boundaries.
- Performing exploratory testing to uncover hidden defects and usability issues in the system.

CHAPTER 9

RESULTS AND DISCUSSIONS

9.1 Evaluation Measures

There are two measurement scales have been used for this proposed work.

9.1.1 Cohen's Kappa: Cohen's kappa is an effective measure, used for interrater reliability between items. The formula is as follows:

$$K = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \quad (17)$$

Table 9.1.1.1 Cohen's Kappa Description

Item	Description
Pr(a)	A total number of the percentage that is consistent between raters.
Pr(e)	If a percentage of total measurement changes, then the number of measurements changes between raters.

The short answer grading has been evaluated in Roshan Tara School, Mehrabpur, Sindh Pakistan. Out of 78 students, 60 students in 10th grade participated in the quiz. Out of 60 students, 52 responses were complete. For the assessment, we have involved a human expert to evaluate the performance and grade the assessment. Later, we compared the human assigned grades with a deep learning model that grades the answers automatically. However, the measurement provides consistency between the two raters. Moreover, Cohen's kappa scale is only applied to qualitative measurement. So for that, the data must be in a categorical form. The experiment shows the consistent equivalency between raters. Furthermore, we have evaluated the categories with the help of a confusion matrix to check the accuracy of the model.

Table 9.1.1.2 Comparison of grading between manual and automatic grading

Student no	Question no	Automatic short answer grading	Manual grading
1	1	Incorrect	Incorrect
1	2	Correct	Correct
1	8	Correct	Incorrect
2	1	Correct	Correct
2	8	Correct	Correct
3	5	Correct	Correct
4	1	Correct	Correct
52	1	Correct	Correct
3	8	Incorrect.	Correct
4	8	Correct	Correct
52	8	Correct	Correct
52	1	Correct	Correct

Table 9.1.1.3 Values of consistency between grades.

Grading		Correct	Incorrect (Incomplete or contradictory)
Manual grading	Correct	77%	4%
Automatic grading	Incorrect	3%	15%

$$K = \frac{(0.77*0.15)-(0.81*0.79)+(0.19+0.21)}{1-(0.81*0.079)+(0.19+0.21)}$$

$$K=0.77\%$$

9.1.2 Confusion matrix: This matrix is used to check the performance of the machine learning algorithm. The confusion matrix represents the predictions and actual conditions generated by the algorithm. Following are the performance evaluator's parameters of the confusion matrix.

- True Positive: Predicts positive classes as positive classes
- True Negative: Predicts negative class as negative class
- False Positive: Predicts the negative class as positive class
- False Negative: Predicts positive class as negative class

Table 9.1.2.1 Prediction type

	Predicted Positive (PP)	Predicted Negative (PN)
True	True Positive (TP) 77%	True Negative (TN) 15%
False	False Positive (FP) 4%	False Negative (FN) 3%

Table 9.1.2.2 Definition of parameters included in the confusion matrix

Factors	Definition
Precision	“The ratio of accurately predicted positive observations to total expected positive observations”.
Recall	“The ratio of actual positive instances that were predicted properly”.
F1 Score	“F1 score is the harmonic mean between Precision and Recall”.
Accuracy	“The number of true negatives (TN) and true positive (TP) samples divided by the total number of samples.”

$$Precision = \frac{TP}{TP+FP} = \frac{77}{77+4} * 100 = 0.95\% \quad (18)$$

$$Recall = \frac{TP}{TP+FN} = \frac{77}{77+3} * 100 = 0.96\% \quad (19)$$

$$F1 = \frac{Precision*Recall}{Precision+Recall} * 2 = 0.96\% \quad (20)$$

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} = 0.93\% \quad (21)$$

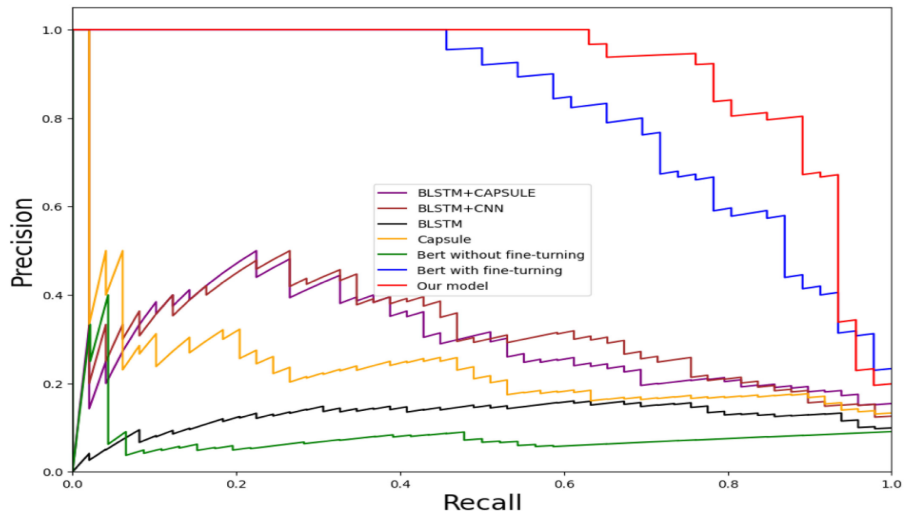


Fig 9.1.2.1 Comparison on Precision and recall with other models

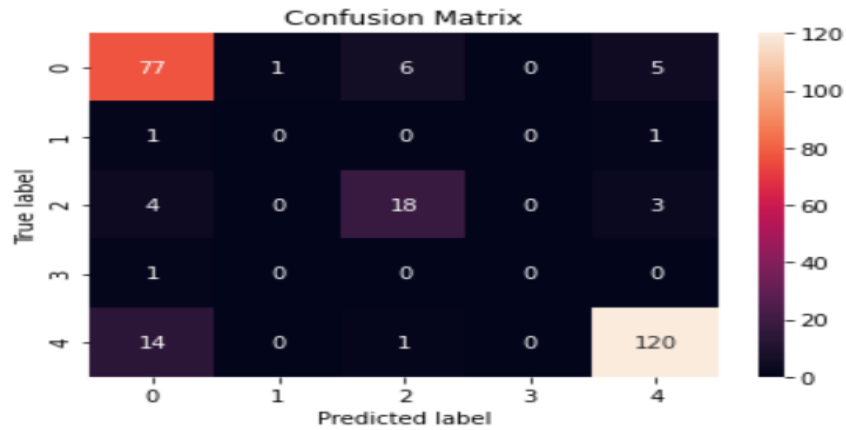


Fig 9.1.2.2 Confusion matrix

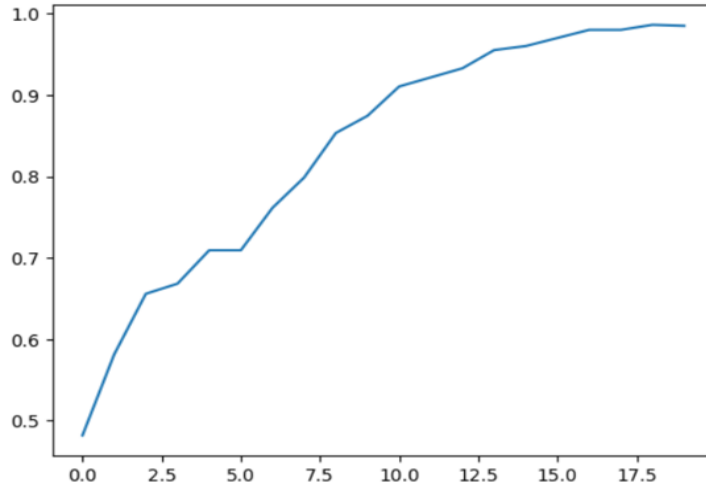


Fig 9.1.2.3 ASAG accuracy

9.2 Performance Analysis

ASAG using BERT and Bi-LSTM network obtained the best average performance. In most cases, it achieves the best accuracies compared to the other models. In Table 9.2.1 the ablation result of w/o refinement shows that our semantic refinement layer significantly improves the grading accuracy of the BERT model on the Mohler dataset and the SemEval-2013 UQ subset. As with the UQ subset in the SemEval-2013 dataset, the testing questions and training questions on the Mohler dataset originate from the same domain. Therefore, this ablation result means that our semantic refinement layer can significantly improve the generalization ability of the BERT model to the features in the domain. The ablation result of w/o Bi-LSTM with BERT shows that after directly replacing Bi-LSTM with the output of BERT, the grading accuracy of the model on two datasets has decreased to varying degrees. This means that the complex gate structures in the Bi-LSTM network can extract more refined context information from the output of the BERT model, just as it performs in other natural language processing tasks, such as Named Entity Recognition.

Table 9.2.1. Accuracy obtained from different models

Different models	Unseen answer	Unseen questions	Unseen domain
Our model	76.5	69.2	66.0
CNN	70.1	68.8	63.4
BERT without Bi-LSTM	72.3	65.7	65.7
Without multihead	70.9	68.4	64.5

Table 9.2.2. Comparison of different models on Mohler dataset

Different models	MAE	RMSE	Pearson
Our model	0.248	0.827	0.897
CNN	0.309	0.821	0.884
BERT without Bi-LSTM	0.254	0.833	0.884
Without multihead	0.483	1.341	0.751

From the result we can observe that our semantic refinement layer can significantly improve the generalization ability of the BERT model to the features in the domain. Lower is better for MAE and RMSE; higher is better for accuracy and Pearson’s r. The ablation result of w/o Bi-LSTM with BERT shows that after directly replacing Bi-LSTM with the output of BERT, the grading accuracy of the model on two datasets has decreased to varying degrees. This means that the complex gate structures in the Bi-LSTM network can extract more refined context information.

CHAPTER 10

CONCLUSION AND FUTURE WORKS

10.1 CONCLUSION

In this article, we proposed a novel BERT-based deep neural network model for ASAG. Through extensive experimental comparison, this article reveals the following theoretical implications:

- Word-embedding-based self-training neural networks, such as CNN and LSTM cannot play an important role in ASAG tasks with small datasets.
- The BERT model, which has a deep transformer coding structure and is pretrained on a large-scale corpus, can perform better by simply fine-tuning in the ASAG task.
- On the fine-tuned BERT model, the LSTM can extract more fine semantics to deepen the BERT model’s understanding of the answer text and further improve the feature generalization ability of the BERT model in the domain.

10.2 FUTURE WORKS

- We plan to improve the performance of their open-domain ITSs, such as the Unseen Domain subset in the SemEval-2013 benchmark. The Unseen Domain subset in the SemEval-2013 benchmark is a challenging task because it requires the ITS to be able to answer questions on a domain that it has not been trained on.
- We plan to do this by training the models on a larger corpus. Training the model on a larger corpus will help it to generalize better to new domains.
- In addition, we plan to use the BERT model to eliminate pronouns in students' answers in order to further improve the scoring accuracy of the ASAG task.

REFERENCES

1. Xinhua Zhu, Han Wu, and Lanfang Zhang, “Automatic Short-Answer Grading via BERT-Based Deep Neural Networks”, IEEE TRANSACTIONS ON LEARNING TECHNOLOGIES, vol.15, NO. 3, pp.364-371, JUNE 2022.
2. W. Song, Z. Wen, Z. Xiao, and S. Park, “Semantics perception and refinement network for aspect-based sentiment analysis,” Knowl.-Based Syst., vol. 214, Art. no. 106755, 2021.
3. M . T. Nguyen, D. T. Le, and L. Le, “Transformers-based information extraction with limited data for domain-specific business documents,” Eng. Appl. Artif. Intell., vol. 97, Art. no. 104100, 2021.
4. C. N. Tulu, O. Ozkaya, and U. Orhan, “Automatic short answer grading with SemSpace sense vectors and MaLSTM,” IEEE Access, vol. 9, pp. 19270–19280, 2021.
5. W. X. Liao, B. Zeng, X. W. Yin, and P. F. Wei, “An improved aspect category sentiment analysis model for text sentiment analysis based on RoBERTa,” Appl. Intell., vol. 51, pp. 3522–3533, 2021.
6. G. G. Smith, R. Haworth, and S. Zitnik, “Computer science meets education: Natural language processing for automatic grading of open-ended questions in eBooks,” J. Educ. Comput. Res., vol. 58, no. 7, pp.1227–1255, 2020

7. M. Uto and Y. Uchida, “Automated short-answer grading using deep neural networks and item response theory,” in *Proc. Int. Conf. Artif. Intell. Educ.*, pp. 334–339, 2020.
8. H. Tan, C. Wang, Q. Duan, Y. Lu, and R. Li, “Automatic short answer grading by encoding student responses via a graph convolutional network,” *Interact. Learn. Environ.*, vol. 2020, pp. 1–15, 2020.
9. N. Seuzen, A. N. Gorban, J. Levesley, and E. M. Mirkes, “Automatic short answer grading and feedback using text mining methods,” *Procedia Comput. Sci.*, vol. 169, no. 2020, pp. 726–743, 2020.
10. A. Sahu and P. K. Bhowmick, “Feature engineering and ensemble-based approach for improving automatic short-answer grading performance,” *IEEE Trans. Learn. Technol.*, vol. 13, no. 1, pp. 77–90, Jan.–Mar. 2020.
11. C. Leon and F. Anna, “Investigating transformers for automatic short answer grading,” in *Proc. Int. Conf. Artif. Intell. Educ.*, pp. 43–48, 2020.
12. Y. Zhang, C. Lin, and M. Chi, “Going deeper: Automatic short-answer grading by combining student and question models,” *User Model. UserAdapted Interact.*, vol. 30, no. 1, pp. 51–80, 2020.
13. H. Yang, B. Zeng, J. Yang, Y. Song, and R. Xu, “A multi-task learning model for Chinese-oriented aspect polarity classification and aspect term extraction,” *Neurocomputing*, vol. 419, pp. 344–356, 2020.
14. J. Zhou, X. Huang, Q. Hu, and L. He, “SK-GCN: Modeling syntax and knowledge via graph convolutional network for aspect-level sentiment classification,” *Knowl.-Based Syst.*, vol. 205, no. 3, Art. no. 106292, 2020.
15. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proc. Annu.*

- Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., pp. 4171–4186, 2019.
- 16.Y. Liu et al., “RoBERTa: A robustly optimized BERT pretraining approach,”arXiv:1907.11692v1,2019.
 - 17.T. Liu, W. Ding, Z. Wang, J. Tang, G. Huang, and Z. Liu, “Automatic short answer grading via multiway attention networks,” in Proc. Int. Conf. Artif. Intell. Educ.,pp. 169–173,2019.
 - 18.B. Yang, L. Wang, D. F. Wong, L. S. Chao, and Z. Tu, “Convolutional self-attention networks,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., vol. 1, pp. 4040–4045, 2019.
 - 19.S. Saha, T. I. Dhamecha, S. Marvaniya, R. Sindhgatta, and B. Sengupta, “Sentence level or token level features for automatic short answer grading?: Use both,” in Proc. Int.Conf. Artif. Intell. Educ., pp. 503–517, 2018.
 - 20.S. Marvaniya, S. Saha, T. I. Dhamecha, P. Flotz, R. Sindhgatta, and B. Sengupta, “Creating scoring rubric from representative student answers for improved short answer grading,” in Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.,pp. 993–1002, 2018.
 - 21.W. Zhao, J. Ye, M. Yang, Z. Lei, S. Zhang, and Z. Zhao, “Investigating capsule networks with dynamic routing for text classification,” in Proc. Conf. Empirical Methods Natural Lang. Process., pp. 43–48. 2018.
 - 22.A. Vaswani et al., “Attention is all you need,” in Proc. 31st Conf. Neural Inf. Process. Syst., pp. 5998–6008,2017.
 - 23.S. Kumar, S. Chakrabarti, and S. Roy, “Earth movers distance pooling over Siamese LSTMs for automatic short answer grading,” in Proc. Int. Joint Conf. Artif.Intell., pp. 2046–2052,2017.

- 24.D.Alikaniotis, H. Yannakoudakis, and M. Rei, “Automatic text scoring using neural networks,” in Proc. 54th Annu. Meeting Assoc. Comput Linguistics, vol. 1, pp. 715–725, 2016.
- 25.M. A. Sultan, C. Salazar, and T. Sumner, “Fast and easy short answer grading with high accuracy,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.,pp. 1070–1075,2016.
- 26.J.L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,”, arXiv:1607.06450, 2016.
- 27.S. Drissi and A. Amirat, “An adaptive e-learning system based on student’s learning styles: An empirical study,” Int. J. Distance Educ. Technol., vol. 14, no. 3, pp. 34–51, 2016.
- 28.G. Jorge-Botana, J. M. Luzon, I. Gomez-Veiga, and M. Cordero, “Automated LSA assessment of summaries in distance education: Some variables to be considered,” J. Educ. Comput. Res., vol. 52, no. 3, pp. 341–364, 2015.
- 29.M. Heilman and N. Madnani, “ETS: Domain adaptation and stacking for short answer scoring,” in Proc. Joint Conf. Lexical Comput. Semantics, vol. 2, pp. 275–279,2013.
- 30.M. O. Dzikovska et al., “SemEval-2013 task 7: The joint student response analysis and 8th recognizing textual entailment challenge,” in Proc. 2nd Joint Conf. Lexical Comput. Semantics,vol. 2, pp. 263–274, 2013.
- 31.M. Mohler, R. Bunescu, and R. Mihalcea, “Learning to grade short answer questions using semantic similarity measures and dependency graph alignments,” in Proc. Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol., pp. 752–762, 2011.

APPENDICES

BERT MODEL

```
import unicodedata,
from bert4keras.snippets import is_string, is_py2
from bert4keras.snippets import open
from bert4keras.snippets import convert_to_unicode
from bert4keras.snippets import truncate_sequences
from bert4keras.snippets import lowercase_and_normalize
def load_vocab(dict_path, encoding='utf-8', simplified=False, startswith=None):
    token_dict = { }
    with open(dict_path, encoding=encoding) as reader:
        for line in reader:
            token = line.split()
            token = token[0] if token else line.strip()
            token_dict[token] = len(token_dict)
    if simplified:
        new_token_dict, keep_tokens = { }, []
        startswith = startswith or []
        for t in startswith:
            new_token_dict[t] = len(new_token_dict)
            keep_tokens.append(token_dict[t])
        for t, _ in sorted(token_dict.items(), key=lambda s: s[1]):
            if t not in new_token_dict and not Tokenizer._is_redundant(t):
                new_token_dict[t] = len(new_token_dict)
                keep_tokens.append(token_dict[t])
    return new_token_dict, keep_tokens
```

```

else:
    return token_dic

def save_vocab(dict_path, token_dict, encoding='utf-8'):
    with open(dict_path, 'w', encoding=encoding) as writer:
        for k, v in sorted(token_dict.items(), key=lambda s: s[1]):
            writer.write(k + '\n')

class TokenizerBase(object):
    def __init__(
        self,
        token_start='[CLS]',
        token_end='[SEP]',
        pre_tokenize=None,
        token_translate=None
    ):
        self._token_pad = '[PAD]'
        self._token_unk = '[UNK]'
        self._token_mask = '[MASK]'
        self._token_start = token_start
        self._token_end = token_end
        self._pre_tokenize = pre_tokenize
        self._token_translate = token_translate or {}
        self._token_translate_inv = {
            v: k
            for k, v in self._token_translate.items()
        }

```

```

def tokenize(self, text, maxlen=None):
    tokens = [self._token_translate.get(token) or token for token in
self._tokenize(text) ]
    if self._token_start is not None:
        tokens.insert(0, self._token_start)
    if self._token_end is not None:
        tokens.append(self._token_end)
    if maxlen is not None:
        index = int(self._token_end is not None) + 1
        truncate_sequences(maxlen, -index, tokens)
    return tokens

def token_to_id(self, token):
    raise NotImplementedError

def tokens_to_ids(self, tokens):
    return [self.token_to_id(token) for token in tokens]

def encode(
    self,
    first_text,
    second_text=None,
    maxlen=None,
    pattern='S*E*E',
    truncate_from='right'
)
    if isinstance(first_text):
        first_tokens = self.tokenize(first_text)
    else:

```

```

    first_tokens = first_text
if second_text is None:
    second_tokens = None
elif isinstance(second_text, str):
    second_tokens = self.tokenize(second_text)
else:
    second_tokens = second_text
if maxlen is not None:
    if truncate_from == 'right':
        index = -int(self._token_end is not None) - 1
    elif truncate_from == 'left':
        index = int(self._token_start is not None)
    else:
        index = truncate_from
    if second_text is not None and pattern == 'S*E*E':
        maxlen += 1
    truncate_sequences(maxlen, index, first_tokens, second_tokens)
first_token_ids = self.tokens_to_ids(first_tokens)
first_segment_ids = [0] * len(first_token_ids)
if second_text is not None:
    if pattern == 'S*E*E':
        idx = int(bool(self._token_start))
        second_tokens = second_tokens[idx:]
    second_token_ids = self.tokens_to_ids(second_tokens)
    second_segment_ids = [1] * len(second_token_ids)

```

LSTM models

```
from constants import GLOVE_DIR
```

```
from keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
```

```
from keras.models import Sequential
```

```
from utils import tokenizer, load_embedding_matrix
```

```
def get_model(embedding_dimension, essay_length):
```

```
    vocabulary_size = len(tokenizer.word_index) + 1
```

```
    embedding_matrix = load_embedding_matrix(glove_directory=GLOVE_DIR,
embedding_dimension=embedding_dimension)
```

```
    model = Sequential()
```

```
    model.add(Embedding(vocabulary_size, embedding_dimension,
weights=[embedding_matrix], input_length=essay_length, trainable=False,
mask_zero=False))
```

```
    model.add(LSTM(64, dropout=0.4, recurrent_dropout=0.4,
return_sequences=True))
```

```
    model.add(Dropout(0.4))
```

```
    model.add(LSTM(256, dropout=0.4, recurrent_dropout=0.4))
```

```
    model.add(Dropout(0.4))
```

```
    model.add(Dense(1, activation='sigmoid'))
```

```
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
```

```

model.summary()

return model

from constants import GLOVE_DIR

from keras.layers import Embedding, LSTM, Dense, Dropout, Conv1D, Lambda,
Flatten, MaxPooling1D

from keras.models import Sequential

import keras.regularizers

import keras.backend as K

from utils import tokenizer, load_embedding_matrix

def get_model(embedding_dimension, answer_length).

    Returns compiled model.

    vocabulary_size = len(tokenizer.word_index) + 1

    embedding_matrix = load_embedding_matrix(GLOVE_DIR,
embedding_dimension)

    model = Sequential()

    model.add(Embedding(vocabulary_size, embedding_dimension,
weights=[embedding_matrix], input_length=essay_length, trainable=False,
mask_zero=False))

    # model.add(Conv1D(filters=50, kernel_size=5, padding='same'))

```



```

    # model.add(LSTM(300, dropout=0.4, recurrent_dropout=0.4,
return_sequences=True))

    # model.add(Dropout(0.4))

    # model.add(Lambda(lambda x: K.mean(x, axis=1, keepdims=True)))

    model.add(Conv1D(filters=50, kernel_size=5, padding='same'))

    model.add(MaxPooling1D(2))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(1, activation='sigmoid',
activity_regularizer=keras.regularizers.l2(0.0)))

    model.compile(loss='mean_squared_error', optimizer='adam')

    return model

from constants import GLOVE_DIR

from keras.layers import Embedding, LSTM, Dense, Dropout, Lambda, Flatten

from keras.models import Sequential

from layers import Conv1DWithMasking

import keras.regularizers

import keras.backend as K

```

```

from utils import tokenizer, load_embedding_matrix

def get_model(embedding_dimension, essay_length):

    Returns compiled model.

    vocabulary_size = len(tokenizer.word_index) + 1

    embedding_matrix = load_embedding_matrix(glove_directory=GLOVE_DIR,
embedding_dimension=embedding_dimension)

    model = Sequential()

    model.add(Embedding(vocabulary_size, embedding_dimension,
weights=[embedding_matrix], input_length=essay_length, trainable=True,
mask_zero=True))

    model.add(Conv1DWithMasking(nb_filter=64, filter_length=3,
border_mode='same', subsample_length=1))

    model.add(LSTM(128, dropout=0.4, recurrent_dropout=0.4,
return_sequences=False))

    model.add(Dropout(0.4))

    model.add(Lambda(lambda x: K.mean(x, axis=1, keepdims=True)))

    model.add(Dense(1, activation='sigmoid',
activity_regularizer=keras.regularizers.l2(0.0)))

    model.compile(loss='mse', optimizer='rmsprop')

    return model

```

```

from constants import GLOVE_DIR

from keras.layers import Embedding, LSTM, Dense, Dropout, Lambda, Flatten

from keras.models import Sequential

import keras.backend as K

from utils import tokenizer, load_embedding_matrix

def get_model(embedding_dimension, essay_length):

    vocabulary_size = len(tokenizer.word_index) + 1

    embedding_matrix = load_embedding_matrix(glove_directory=GLOVE_DIR,
embedding_dimension=embedding_dimension)

    model = Sequential()

    model.add(Embedding(vocabulary_size, embedding_dimension,
weights=[embedding_matrix], input_length=essay_length, trainable=False,
mask_zero=False))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(256, activation='relu'))

    model.add(Dropout(0.3))

    model.add(Dense(1, activation='sigmoid'))

```

```

    model.compile(loss='mean_squared_error', optimizer='rmsprop',
metrics=['mae'])

    model.summary()

    return model

from constants import GLOVE_DIR

from keras.layers import Embedding, LSTM, Dense, Dropout, Lambda, Flatten

from keras.models import Sequential

import keras.backend as K

from utils import tokenizer, load_embedding_matrix

def get_model(embedding_dimension, essay_length):

    vocabulary_size = len(tokenizer.word_index) + 1

    embedding_matrix = load_embedding_matrix(glove_directory=GLOVE_DIR,
embedding_dimension=embedding_dimension)

    model = Sequential()

    model.add(Embedding(vocabulary_size, embedding_dimension,
weights=[embedding_matrix], input_length=essay_length, trainable=False,
mask_zero=False))

    model.add(LSTM(64, dropout=0.4, recurrent_dropout=0.4))

    model.add(Dropout(0.5))

    model.add(Lambda(lambda x: K.mean(x, axis=1, keepdims=True)))

```

```

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='rmsprop',
metrics=['mae'])

model.summary()

return model

```

Implements LSTM with Mean over Time layer.

```

from constants import GLOVE_DIR

from keras.layers import Embedding, LSTM, Dense, Dropout, Conv1D, Lambda
from keras.models import Sequential

import keras.regularizers

from utils import tokenizer, load_embedding_matrix

import keras.backend as K

def get_model(embedding_dimension, essay_length):

    Returns compiled model.

    vocabulary_size = len(tokenizer.word_index) + 1

    embedding_matrix = load_embedding_matrix(GLOVE_DIR,
embedding_dimension)

    model = Sequential() model.add(Embedding(vocabulary_size,
embedding_dimension, weights=[embedding_matrix], input_length=essay_length,
trainable=False, mask_zero=False))

```

```

model.add(Conv1D(filters=50, kernel_size=5, padding='same'))

model.add(LSTM(300, dropout=0.4, recurrent_dropout=0.4,
return_sequences=True))

model.add(Lambda(lambda x: K.mean(x, axis=1)))model.add(Dropout(0.4))

model.add(Dense(1, activation='sigmoid',
activity_regularizer=keras.regularizers.l2(0.0)))

model.compile(loss='mean_squared_error', optimizer='adam')

return model

```

Grading System

```

from django.contrib import admin

from . import models

# Register your models here.

admin.site.register(models.Question)

admin.site.register(models.Essay)

from django.apps import AppConfig

class GraderConfig(AppConfig):

    name = 'grader'

from django import forms

from .models import Question, Essay

```

```

class AnswerForm(forms.ModelForm):

    answer = forms.CharField(max_length=100000,
widget=forms.Textarea(attrs={'rows': 5, 'placeholder': "What's on your mind?"}))

    class Meta:

        model = Essay

        fields = ['answer']

from django.db import models

# Create your models here.

class Question(models.Model):

    """ A model of the 8 questions. """

    question_title = models.TextField(max_length=100000)

    set = models.IntegerField(unique=True)

    min_score = models.IntegerField()

    max_score = models.IntegerField()

    def __str__(self):

        return str(self.set)

class Essay(models.Model):

    """ Essay to be submitted. """

    question = models.ForeignKey(Question, on_delete=models.CASCADE)

```

```

content = models.TextField(max_length=100000)

score = models.IntegerField(null=True, blank=True)

from django.test import TestCase

# Create your tests here.

from django.urls import path

from django.conf.urls import url

from . import views

urlpatterns = [

    path("", views.index, name='index'),

    path('<int:question_id>/', views.question, name='question'),

    path('<int:question_id>/essay<int:essay_id>/', views.essay, name='essay'),

]

from django.shortcuts import get_object_or_404, render, redirect

from django.http import HttpResponseRedirect, HttpResponse

from django.urls import reverse

from .models import Question, Essay

from .forms import AnswerForm

from .utils.model import *

from .utils.helpers import *

```



```

import os

current_path = os.path.abspath(os.path.dirname(__file__))

# Create your views here.

def index(request):

    questions_list = Question.objects.order_by('set')

    context = {

        'questions_list': questions_list,

    }

    return render(request, 'grader/index.html', context)

def essay(request, question_id, essay_id):

    essay = get_object_or_404(Essay, pk=essay_id)

    context = {

        "essay": essay,

    }

    return render(request, 'grader/essay.html', context)

def question(request, question_id):

    question = get_object_or_404(Question, pk=question_id)

    if request.method == 'POST':

        # create a form instance and populate it with data from the request:

```

```

form = AnswerForm(request.POST)

if form.is_valid():

    content = form.cleaned_data.get('answer')

    if len(content) > 20:

        num_features = 300

        model =
word2vec.KeyedVectors.load_word2vec_format(os.path.join(current_path,
"deep_learning_files/word2vec.bin"), binary=True)

        clean_test_essays = []

        clean_test_essays.append(essay_to_wordlist( content,
remove_stopwords=True ))

        testDataVecs = getAvgFeatureVecs( clean_test_essays, model,
num_features )

        testDataVecs = np.array(testDataVecs)

        testDataVecs = np.reshape(testDataVecs, (testDataVecs.shape[0], 1,
testDataVecs.shape[1]))

        lstm_model = get_model()

        lstm_model.load_weights(os.path.join(current_path,
"deep_learning_files/final_lstm.h5"))

        preds = lstm_model.predict(testDataVecs)

```

```

    if math.isnan(preds):

        preds = 0

    else:

        preds = np.around(preds)

    if preds < 0:

        preds = 0

    if preds > question.max_score:

        preds = question.max_score

    else:

        preds = 0

    K.clear_session()

    essay = Essay.objects.create(

        content=content,

        question=question,

        score=preds

    )

    return redirect('essay', question_id=question.set, essay_id=essay.id)

else:

    form = AnswerForm()

context = {

```

```

        "question": question,

        "form": form,

    }

    return render(request, 'grader/question.html', context)

#!/usr/bin/env python

import os

import sys

if __name__ == '__main__':

    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')

    try:

        from django.core.management import execute_from_command_line

    except ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you "

            "forget to activate a virtual environment?"

        ) from exc

    execute_from_command_line(sys.argv)

import os

```

```

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

SECRET_KEY = '7a8(4bh(%&amg0skdnt9t7)(4@&8+^!pl_yco3fssz2_zjab+e'

DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [

    'widget_tweaks',

    'grader.apps.GraderConfig',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

```

```

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT_URLCONF = 'mysite.urls'

#BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [

            os.path.join(BASE_DIR, 'templates'),

        ],

        'APP_DIRS': True,

        'OPTIONS': {

            'context_processors': [

                'django.template.context_processors.debug',

                'django.template.context_processors.request',

                'django.contrib.auth.context_processors.auth',

```

```

        'django.contrib.messages.context_processors.messages',

    ],

},

},

]

WSGI_APPLICATION = 'mysite.wsgi.application'

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),

    }

}

AUTH_PASSWORD_VALIDATORS = [

    {

        'NAME':

'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',

    },

    {

```

```

        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',

    },

    {

        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',

    },

    {

        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',

    },

]

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

STATIC_URL = '/static/'

STATICFILES_DIRS = [

```



```

    os.path.join(BASE_DIR, 'static'),

]

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

    path("", include('grader.urls')),

    path('admin/', admin.site.urls),

]

```

manage.py

```

import os

import sys

if __name__ == '__main__':
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

```