MACHINE LEARNING ALGORITHM TO DETECT IMPERSONATION IN AN ESSAY-BASED ELECTRONIC EXAMINATION

Submitted by

JOSEPH KOMBE SAMUEL

TU01-IC321-0003/2015

Masters of Science in Information Technology

Taita Taveta University

September 2023

DECLARATION

This thesis is my original work and has not been presented for award of a degree in any other University.

Signature: [illegible] Date: 11/09/2023

Joseph Kombe Samuel

This thesis has been submitted for examination with my approval as University Supervisor.

Signature: [illegible] Date: 11/09/2023

Dr. Peter Ochieng', PhD.

Department of Informatics and Computing

Taita Taveta University, Kenya.

Signature: [illegible] Date: 11/09/2023

Dr. Solomon Mwanjele, PhD.

Department of Informatics and Computing

Taita Taveta University, Kenya

DEDICATION

I dedicate this thesis to the Almighty God Jesus Christ for the opportunities He granted me. To my wonderful supervisors. You have been encouraging me all through. My wife Irene Twala, sons Ryan Sifa and Joel Muvera. You have been caring for me and showing me love which enabled me concentrate in my work. To my parents, your prayers have been keeping me going. My friend Joshua Serem, your encouragements went along way to pushing me complete my work. May God guide you always.

ACKNOWLEDGEMENT

I would like to express my sincere appreciation and gratitude to all those who have contributed to the successful completion of this thesis.

First and foremost, I am deeply thankful to my supervisors, Dr. Peter Ochieng' and Dr. Solomon Mwanjele, for their unwavering support, guidance, and mentorship throughout this research. Their expertise, insightful feedback, and dedication to my academic growth have been instrumental in shaping this work.

I am also thankful to Taita Taveta University for providing the resources, facilities, and academic environment essential for conducting this research.

I would like to acknowledge the contributions of my research collaborators and colleagues at Informatics and Computing Department, whose collaboration and discussions have enriched this work and expanded my intellectual horizons.

My heartfelt appreciation goes to my friends and family for their unwavering encouragement, understanding, and emotional support during this challenging journey. Your belief in me kept me motivated, and I am truly thankful for your presence in my life.

Lastly, I want to acknowledge the countless hours of hard work and dedication put forth by all the participants and volunteers who contributed to the data collection process. Your willingness to participate in this research made this study possible, and I am indebted to you for your cooperation.

In conclusion, I am deeply grateful to all those who have played a role, big or small, in the completion of this thesis. Your support and contributions have been invaluable, and I am honoured to have had the opportunity to work alongside such talented and dedicated individuals.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
CBT	Computer-Based Training
CF-Score	Content Feedback Score
COVID-19	Coronavirus Disease of 2019
CPU	Central Processing Unit
DNN	Dynamic Neural Network
E-assessment	Electronic Assessment
E-exam	Electronic Exam
E-learners	Electronic Learners
E-learning	Electronic Learning
EOG	Electroocoulogram
ERR	Equal Error Rate
GHz	Gigahertz
GMM-UBM	Gaussian Mixture Model - Universal Background Model
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
ICT	Information Communication Technology

IP	Internet Protocol
KDD	Knowledge Discovery in Databases
LSTM	Long Short-Term Memory
MB	Megabyte
MHz	Megahertz
ML	Machine Language
NER	Named Entity Recognition
OTS	Occurrence Tracking System
POS	Part of Speech
RNN	Recurrent Neural Networks
SVM	Support Vector Machine

ABSTRACT

With the introduction of e-learning systems, online essay-based exams have become common in universities and colleges. Impersonation during e-exams is a challenging issue that is difficult to detect and report. Several studies have been done to develop e-exam authentication schemes, but they have not completely eliminated the problem of impersonation. This research aimed to tackle this issue using machine learning techniques to analyze students' writing techniques and create unique patterns for each candidate. The study developed, trained, and evaluated real-time LSTM, RNN, and GRU algorithms for detecting impersonation and plagiarism in essay-based e-exams at the word and character levels. The algorithms were benchmarked against other state-of-the-art models in the same area. One hundred datasets of English words in a text files of size 286 MB were used to represent dummy writings of 100 students, and spaCy was used for data pre-processing. The performance of the models were presented in terms of prediction accuracy. The GRU model achieved the highest accuracy of 98.6% compared to other models in similar studies. Therefore, GRU model, can accurately detect cheating in essay-based online exams.

CHAPTER ONE

INTRODUCTION

1.0. Background

E-learning is a type of learning that takes place through the use of electronic media (Janelli, 2018). In 1999, it was first used during a seminar on Computer-Based Training (CBT) systems (Soni, 2020). E-learning, also referred to as virtual or online learning, is becoming an essential aspect of contemporary education, highlighting the significant impact of ICT in the teaching and learning process (Soni, 2020). The growth of online devices has facilitated the delivery of information on E-learning platforms to students, wherever and whenever they need it. Urosevic (2019) stated that in the year 2017 there were approximately 23 million new online learners, boosting the total number of E-learners to 81 million globally. With the COVID-19 pandemic, most universities and colleges were shut down to stop the virus infection from spreading. This made the universities and colleges think of alternative teaching methods during the lockdown period and thus increased the use of E-learning (Almaiah, Al-Khasawneh & Althunibat, 2020).

The existence of E-learning environment introduced the aspect of E-assessments. In assessing students, universities and colleges mostly use essay-based E-exams other than choice-based ones because they require students to support their arguments with evidence (Hashim et al., 2018). Essay-based E-exams require answers to be written out at some length in an E-learning platform. The questions require a response with multiple paragraphs and should be logical and well-structured (Frederiks, Derrington & Bartlett, 2021).

Adopting E-learning has made authentication of the identity of students and their work's authenticity during assessment critical to reduce academic dishonesty like impersonation and copy-pasting content from other sources (Okada, Whitelock, Holmes & Edwards, 2019). Authentication is also essential for quality assurance purposes in education, though, implementing e-learning to combat impersonation and plagiarism is a big challenge (AV & Rathi, 2021).

Various researches have been proposed to curb cheating and plagiarism. In their study, Rathgeb, Pöppelmann & Gonzalez-Sosa (2020) recommended biometric technologies to be integrated into E-learning platforms to assure the presence of the actual student during E-exam. However, Gomez-Barrero et al. (2021) stated that in the COVID-19 era, the use of hygienic masks and other actions taken to prevent the spread of the virus have negatively affected biometrics technology. Tiong & Lee (2021) suggested an intelligent agent for e-cheating. This agent integrated IP and behavioral detector protocols that monitored the IP addresses of the students' devices, and assessed the speed at which the students were answering questions respectively. It was tested on DNN, DenseLSTM, LSTM and RNN. Although the agent achieved the highest overall accuracy of 95.32%, it could still not determine impersonation.

Sokout, Purnama, Mustafazada & Usagawa (2020) proposed a model that uses mouse-tracking approach to track learners' behaviors. They employed Support Vector Machine (SVM) to classify and predict students committing illegal behaviors through secret reconstruction of mouse activities and transparent space. While the model had a 94% prediction accuracy, it was impossible to detect impersonation and could not notify the examiner of any potential cheating in real-time (Tzafilkou & Protogeros, 2020). Research by Mattsson (2020) proposed a method that utilizes keystroke dynamics to authenticate students in E-exams. A Gaussian Mixture Model (GMM-UBM) used as a Universal Background Model was used to test and evaluate the method. Despite the fact that the approach had 94.5% accuracy, the author recommended that this solution should not be used in a production environment because of limitations in hardware and the potential flight times inconsistency that can occur when capturing keystrokes on an E-learning platform.

In this research, we propose a mechanism that will detect the impersonation of the students during the exam period. Our focus is on essay-based exams in E-learning and a machine learning technique to detect whether the person doing the exam is the right student. The examiner would get a notification if there is any suspicion and the E-exam platform would lock out the student automatically.

1.1. Research Problem Statement

The advent of E-learning systems has led to the implementation of online essay-based exams, primarily in higher education institutions, allowing students to substantiate their answers with evidence. Despite concerns of cheating and plagiarism in E-learning, the challenge of impersonation during E-exams remains difficult to detect and report (Ullah, Xiao & Barker, 2019).

Numerous studies have aimed to develop E-assessment and E-exam authentication methods but have not completely resolved the impersonation issue. Rathgeb et al. (2020) suggested using biometric technology to address impersonation, but this approach has faced criticism due to the high costs of specialized sensors and limitations related to sensor coverage (Gomez-Barrero et al., 2021). Tiong and Lee (2021) proposed an E-cheating intelligent agent that monitors students' behavior by assessing their response speed, although it is better suited for multiple-choice questions and not essay-based exams. Sokout et al. (2020) introduced a mouse tracking technique to detect illicit actions during E-exams but encountered challenges with real-time detection and impersonation (Tzafilkou & Protogeros, 2020). Another authentication method involves keystroke dynamics, as suggested by (Mattsson, 2020). However, Mattsson (2020) cautioned against using this approach in a production environment due to hardware limitations and timing variations when capturing keystrokes in an E-learning platform.

In the field of computer science and cyber security, there exists a critical challenge pertaining to the prevention of impersonation and plagiarism within the context of essay-based E-examinations. Existing solutions are often constrained by their focus on multiple-choice questions, lack of real-time detection capabilities, and dependence on costly and specialized biometric sensors, potential user interaction issues with these sensors, and their inappropriateness for deployment in production environments.

This research leverages machine learning techniques to develop a novel approach for identifying and preventing impersonation in essay-based E-exams. The solution involves the analysis of students' writing styles, leading to the creation of distinctive patterns for each candidate. Whenever the machine learning algorithm detects signs of potential impersonation, access to the system will be temporarily restricted, requiring authentication by an administrator. This research enhances the security and reliability of online examinations within the realm of computing and cyber security.

1.2. Research Questions

i. Can a real-time word-level and character-level LSTM, RNN and GRU algorithms be employed for detecting impersonation and plagiarism in an essay-based E-exam?
ii. Can the LSTM, RNN and GRU algorithms be trained to detect impersonation and plagiarism both at word and character level?
iii. Can the detection accuracy of LSTM, RNN and GRU algorithms be evaluated?
iv. Can the performance of LSTM, RNN and GRU algorithms in detecting impersonation and plagiarism be compared to other cutting-edge models in the same field of study?

1.3. Research Objectives

1.3.1 General Objective

The overall goal of this research was to devise a machine learning algorithm to detect impersonation and plagiarism during an essay-based exam in an E-learning environment.

1.3.2 Specific Objectives

i. To develop a real-time Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN) and Gated Recurrent Unit (GRU) algorithms to detect impersonation and plagiarism in an essay-based exam using students' inputted words and characters extracted from those words.

ii. To train the developed LSTM, RNN and GRU algorithms to be able to detect impersonation and plagiarism both at word and character level.

iii. To evaluate the accuracy of the developed LSTM, RNN and GRU algorithms in detecting impersonation and plagiarism both at word and character level.

iv. To benchmark the performance of the developed LSTM, RNN and GRU algorithms against other cutting-edge models in the same field of study in detecting cheating in exam in an E-learning environment.

1.4. Justification of the Study

There is satisfactory agreement among researchers that collusion is broadly practiced by students and represents a significant problem across educational institutions. Kasliwal (2015) ascertained that past studies of scholarly untruthfulness have systematically distinguished the mental and social factors corresponded to cheating, yet how students really cheat has frequently been neglected.

The expanding need to join the knowledge society and utilization of the Internet has been reflected in the increasing acceptance and utilization of e-learning in training institutions, (Kahiigi Kigozi et al., 2012). This poses the need to address security and authentication of examination and examiner. Hence, this study if very important as it will propose and provide a mechanism to curb essay-based exam cheating in e-learning.

1.5. Research Scope

The field of computing is diverse, and it keeps expanding with innovations and inventions; this creates a continuum of opportunities that are created by over-reliance of computer systems in enhancing standards of living.

For this reason, this research confines itself to machine learning algorithms, specifically, supervised learning algorithms. The study will focus on three algorithms, these are LSTM, RNN and GRU.

In the e-exams, the study will restrict itself to essay-based exams in e-learning.

CHAPTER TWO

LITERATURE REVIEW

2.0. Introduction

The chapter revisit and summarize past research that has been implemented to address impersonation in e-exam. It also surveys neural network models that have been proposed to be utilized in this study. The models to be discussed include RNN, LSTM and GRU models.

2.1. Existing Impersonation Detection Techniques for E-Exams

Several studies have been recommended to detect impersonation in E-exams, Okada et al. (2019). In their study, Rathgeb et al. (2020) recommended biometric technologies to be integrated in to e-learning platforms to assure the presence of the actual student during e-exam. However, Gomez-Barrero et al. (2021) stated that in the COVID-19 era, the use of surgical masks that cover the nose and mouth, as well as the indirect effects of strict hygiene measures taken to prevent the spread of the virus have negatively affected biometrics technology. In addition, biometric technology has been criticized for the high costs of purchasing special biometric sensors and if the subjects are positioned outside of a biometric sensor's capture area or cannot get in touch with the biometric sensor (Gomez-Barrero et al., 2021).

Tiong & Lee (2021) suggested an intelligent agent for e-cheating that integrates IP and behavior detector protocols. The IP detector monitored the IP addresses of the students' devices. The behavior detector assessed the speed at which the students were answering questions, they were labeled as 'abnormal' if they moved too quickly or too slowly; otherwise were considered as ‘normal’. It was tested in four deep learning algorithms, which were the DNN, DenseLSTM, LSTM and RNN. The DenseLSTM achieved the highest overall accuracy of 95.32%. This intelligent agent could monitor and detect the presence and abnormal behavior of the students but it could not determine whether there was impersonation.

Sokout et al. (2020) proposed a model to track the students’ behavior using a mouse-tracking approach which utilizes Support Vector Machine (SVM) to classify and predict illegal activities committed by students. The classification resulted in the hidden reconstruction of mouse activity and clear space, which resulted to detection of actions like in-activity, copy or cut, paste, double-click, opening new tab and scrolling. These actions determined whether the student was cheating or not. During the online mid-term exam, the model correctly predicted that 94% of the students would cheat. Nonetheless, it is not possible to detect impersonation and does not notify the examiner of any potential cheating in real-time, (Tzafilkou & Protogeros, 2020).

Mattsson (2020) proposed a method that utilizes keystroke patterns to authenticate students during online exams. A GMM-UBM was used to test and evaluate the method. This approach produced an accuracy rate of 94.5% and an Equal Error Rate (ERR) of 5.4%. Despite its high rate of accuracy, the author recommended that this solution not to be used in a production environment because of hardware constraints and unpredictable flight durations that can occur when capturing keystrokes on an e-learning platform.

2.2. Other Existing Approaches for Detecting Exam Cheating Online

2.2.1. Visual Eye Tracking Algorithm

Javed & Aslam (2013) conducted a study that used the Visual Eye Tracking Algorithm that uses cameras to track a learner’s eye movement. Visual Eye Tracking Algorithm detects online exam cheating by reviewing visual attention of an examinee depending on their concentration to the screen. The algorithm utilizes an intelligent alarm system to be used in examination environments to significantly reduce cheating based on the eye movement of the student. It is developed in a way that it can detect a human figure in the exam environment and use face detection and visual eye tracking recognition for authentication. The intelligent visual eye tracking algorithm ensures that examination is free and fair by capturing the face of the examinee and monitoring their eye movement. The system has the capability of edges detection and analysis of eye movement detection using Kalman filtration algorithm while human face is detected using Viola Jones algorithm due to excellent results produced even when using low resolution cameras. Every frame captured is processed and analyzed by comparing it with the previous frame, if there is a significant difference between the frames that satisfies the threshold outlined, an alarm is raised of noted cheating incident. The results showed that the algorithm can follow eye movement of an exam taker and performs pupil analysis with a success rate of 93% and a processing time of 0.9 seconds. This shows its effectiveness in maintaining eye movement and pupil analysis compared to other methods like Tree Classifier, SVM, and EOG algorithms. More research is recommended to include other factors like voice detection to ensure that the exam environment has unauthorized individuals to help in handling the exam.

Another study that used the visual analysis approach was by (Bawarith, Basuhail, Fattouh & Gamalel-Din, 2017). The approach combined a fingerprint reader for authentication as well as an eye tribe tracker for visual analysis during exam sessions. The system worked in such a way that, the examinee was supposed to use fingerprint scanner to allow access into the system as well as tracker for eye tribe to ensure the authentic examinee is sticks throughout the assessment period. If the system noticed the examinee was absent it locked and required authentication using fingerprint. The study had 30 participants who were divided into two groups of 15 cheating and 15 non-cheating participants. Every participant was supposed to undertake three exam sessions so that the data sample equaled 90. The results showed that Sensitivity which assessed the proportion true positive rate to show that the system correctly identified the participants was 100 percent successful. The Specificity which measured the true negative rate to mean the correctly identified cheating instances was 95.56 %. Also, the Precision which measured the fraction of relevant retrieved instances which is the positive predictive value was 95.74% and the Accuracy which is the proximity of obtained results to the true value was 97.78 %. Overall the system had a success rate of 97.83%. The authors recommended that the research could be expanded by implementing it over the internet for distributed systems and include other features like voice detectors to improve accuracy.

2.2.2. Data Mining Techniques

Hernándeza, Ochoab, Muñozd & Burlaka (2006) conducted an experiment to detect cheating in online student exams by employing data mining strategies. The researchers used a nontrivial process called Knowledge Discovery in Databases (KDD) to identify valid, theoretically useful, novel, and understandable patterns in the dataset that could help detect cheating students in exams. They started by analyzing the student behavior who participate in exam cheating and found out that the leading factors include laziness where students do not study or prepare for the tests. There is also pressure to improve grades or pass a course unit, the pressure to excel in academics, the complexity of a class, and lack of information to answer questions. Seventy percent from a sample of 1,800 undergraduates from nine Unites State higher learning institutions around the country confirmed involvement in academic malpractice during exams, (Kopun, 2018).

For the data mining engine, they used Weka which is an assortment of machine learning algorithms employed to carry out data mining tasks. Weka comprises of tools that help in data pre-processing, categorization, grouping, regression, linking, and visualization to show data trends, (Aaron & Roche, 2013). The result of the research survey involved 97 students who were put in a controlled environment to undertake a test, no suspicious patterns were observed that would be termed as cheating. The explanation for the results given was that the server and nodes used were in the same network, the used OTS evades undesirable practices, the supervisor was present in exam room throughout, and the dataset used was small. It was concluded that although the results did not show any cheating patterns, data mining was a powerful tool that could be used to study student behavior over time to successfully detect cheating in online assessments, (Hernández et al., 2006).

In a study conducted by Chen & Chen (2017), Data Mining algorithms were employed to detect cheating in exam rooms by analyzing patterns in students' answers. The study used multivariate statistical tools to observe association patterns in the answer sheets and utilized the Hierarchical Clustering and Dendrogram Tree algorithm for clustering affinity behavior. A Heat Map was employed to visualize score patterns, focusing on the top 20 percent of the most challenging questions among 25 multiple-choice queries to enhance cheating detection. The research involved 75 students seated in groups of three at 25 small tables in a restricted classroom environment. Each student was given a different version of the exam sheet, and cell phones or laptops were prohibited to prevent communication. Data mining strategies, including Heat Map, Principal Component Analysis (PCA), and Clustering Analysis, demonstrated high prediction accuracy in identifying similar answer patterns among students seated at the same table during the exam. However, Multivariate Correlation was found to have limitations in distinguishing cheating due to potential similarities in answers despite differing answering patterns.

2.2.3. Text Mining Techniques

Cavalcanti, Pires, Cavalcanti & Pires (2012) in their research study employed text mining methodology and algorithms to detect academic dishonesty (cheating) by evaluating open-ended college assessments using document classification techniques. The authors note that cheating in Brazilian public universities is a prevalent behavior that lacks a concrete solution, (Cavalcanti et al., 2012).

The study focuses on showing how text mining algorithms are a promising technique for finding a solution that not only detects cheating, but it also estimates cheating on open-ended exams. There are two types of classification techniques namely supervised classification when the information of the classes is already available and non-supervised classification when the information is absent. In this research study for detecting cheating on scholar exams was developed and administered to Business Management and Computer Science students at the Federal University of Campina Grande in Brazil. The case study had thirty scholar exams and each exam had four open-ended questions on administration and marketing written in the Portuguese language. The exams were handled by the selected students and answers were stored electronically in plain text format. Strong evidence of cheating was detected every time the program identified documents with high similarity index due to a large number of identical words, (Cavalcanti et al., 2012).

Decision Tree was used to detect and assess cheating on examinations by applying two classification models namely co-sine based and overlap based models in the supervised algorithm. Results showed that overlap based model performed better by attaining accuracy of approximately 99.43 percent which is an excellent inference quality for cheating detection and evaluation. The results of an overlap that considers two answers from the same question were defined to show that an overlap score of less than 0.22 indicated no cheating, between 0.22 and 0.2 indicated low cheating, a score between 0.3 and 0.38 meant intermediate cheating while a score of over 0.38 showed high cheating. This research study showed that text mining can be used for educational purposes to curb cheating by detection and evaluation mechanisms in a manner that can help a teacher to identify exam malpractices in labor-intensive evaluation tasks, (Cavalcanti et al., 2012).

Another text mining study by Pertile, Moreira & Rosso (2016) to analyze cheating in academic papers using plagiarism detector tools, 85 pairs for PubMed and 96 for ACL were considered. The case study employed 10 human assessors who were tasked with the responsibility of detecting and reporting similarity cases noted. Pooling method had been used select the pairs from the huge pool of dataset in the PubMed and ACL databases. After assessing the evaluations from the assessors, the agreement rate was noted to be 84 percent for ACL and 80 percent for PubMed journals. ParsCit which uses supervised machine-learning method was employed to extract information from the selected scientific papers for content and reference analysis.

Results showed that the intersection of text in the documents from ACL collection ranged between 15 percent and 46 percent. On the other hand, intersection between the content and reference-based metrics was higher reporting a range of 23 to 61 percent. The larger intersection does not translate to higher plagiarism in the documents, but it shows common content and citations were used by the authors. The selected pairs were tested with machine-learning techniques and the findings revealed that hybrid decision-table/Naïve Bayes classifier had better results for ACL and a decision-tree classifier, J48 produced better results for PubMed. It was concluded that with a CF-Score of 0.8 for ACL and 0.9 for PubMed, there was high probability of plagiarism cases among PubMed than ACL journals, (Pertile et al., 2016).

2.2.4. Convolutional Neural Networks

Another study by Kuin (2018) sought to explore three convolutional neural networks to identify fraudulent behavior among students in digital platforms like Canvas or BlackBoard using screen recordings. The study conducted by (Kuin, 2018) proposes the creation of a framework that permits students to handle an assessment using resources of their choice that includes search engines such as Bing or Google. The proposed framework has three parts namely an interface, frame classification, and a video processor. The interface captures and sends student's screen recordings as videos to a pipeline with a series of classification methods. The pipeline executes video processing by shortening long videos into several thousand frames while frame classification creates a series of methods to receive the processed videos. The frame classification categorizes these frames, compiles the results, and sends to the supervisor’s interface indicating instances of fraudulent behavior, (Kuin, 2018).

The study used screen recordings of three digital exam sessions of two hours long in an environment that gave student the freedom to chat on social media and write notes. The videos recorded are then converted to frames that are labeled either fraud or not fraud using ANVIL tool for the collection of images for training the neural network. The total number of frames used was 25,000 images that were divided into 3 categories namely train, validate and test set and allocated with 50 percent, 25 percent, and 25 percent respectively. The model was able to categorize the 25,000 photographs into 12,000 images identified as fraud and 13,000 images categorized as not fraud, according to the results (Kuin, 2018). The results show that VGG16 yields 96.8% accuracy on traditional approach while the cross-validation technique produces 67.1% accuracy. Likewise, Inception-v4 produces results that show 96.0% accuracy when using the traditional approach while showing 46.8% accuracy for the cross-validation method. Lastly, MobileNets which produced the underwhelming results in comparison with the other two by showing a precision of 48.8%

when using a conventional approach and 48.2 percent using the cross-validation method (Kuin, 2018).

Hence, the created framework for detecting fraudulent behavior in online platforms was VGG16 and Inception-v4 approaches due to their high accuracy levels of over 96.8 percent. However, more study is recommended so that the framework can be able to guarantee that the right student is undertaking the tests and supervise the examination environment to verify that the student is taking the exam independently, (Kuin, 2018).

2.2.5. Leakage Theory Approach

A study conducted by Korman (2010) studying the possibility of detecting cheating in online assessment by considering human-computer interaction dynamics. The motivation of this research study is to assist in the detection of prevalent cheating in online assessment by utilization of 'behaviometrics' that considers keystroke, linguistic, and mouse dynamics. The study conducted by Korman (2010) used three scenarios to perform observational analysis showed that there were similar differences that were noted after simulation that changed the mode of writing text, copying from paper or screen, listening in order to write, and copying text by reformulating. This illustrates that the developed algorithm had the capability of identifying specific behavioral anomalies. Developing the behavioral profile of an individual in specified conditions potentially enhanced the indication reliability.

It was concluded that there was a possibility to identify activities like writing text authentically, listening to audios to copy, and copying text by paraphrasing in an exam environment if these activities are prohibited. Availability of behavioral profile increases the possibility of detecting anomalous behavior, valid detection, and classification of the type of anomaly. Further research studies were recommended by the inclusion of automated aspects such as voice detection, face recognition, or fingerprint scanners. These automation approaches seal the detection accuracy loopholes of impersonation, identity concealment, and masquerading that can bypass the system, (Korman, 2010).

2.3. Neural Network Models

A neural network refers to a set of computer algorithms that simulate the functioning of the human brain and can identify patterns. They categorize or classify raw data using a kind of machine learning to interpret sensory inputs. To make sense of any kind of real-world data, be it in the form of images, sound, text, or time series, it is necessary to convert it into numerical patterns that can be represented as vectors (Nicholson, 2018). When compared to support vector machines, neural networks outperform them in many classification situations because they can handle complicated, high-dimensional data sets, (Chalapathy, Menon & Chawla, 2018). Some of the neural network models used in prediction problems are RNN, LSTM as well as GRU.

2.3.1. Recurrent Neural Network (RNN)

RNN are a type of neural network that have hidden states and enable the use of previous outputs as inputs (Amidi & Amidi, 2019). In most cases, the inputs and outputs in a neural network are considered independent of each other. However, there are situations where the context of previous inputs is crucial, such as predicting the next word in a phrase. In such cases, the neural network needs to remember the previous words in order to generate a more accurate prediction. Consequently, the RNN (Recurrent Neural Network) algorithm was created, which utilized a Hidden Layer for addressing the issue. The essential part of the RNN is its hidden state, which preserves particular details about a sequence (Britz, 2017).

The name "RNNs" comes from their characteristic of executing the same operation on each element of a sequence, while using previous computations to determine the final outcome (Britz, 2017).

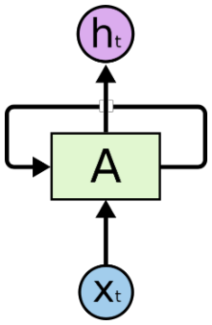


Figure 2.1 Recurrent Neural Networks loop (Olah, 2015)

In Figure 1, a neural network segment A, evaluates some input x(t) and returns a value h(t). A loop can be used to convey data from one network phase to the next. As stated below, an RNN is a collection of multiple replicas of the same network, each of which delivers a message to the next:

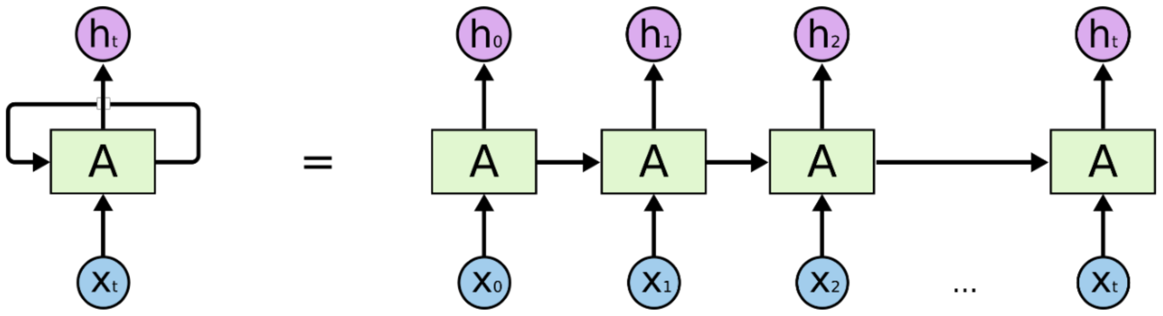


Figure 2.2 An Unrolled Recurrent neural network (Mittal, 2020)

First, it takes X_0 from the sequence of inputs and outputs h_0, and then the next step's inputs would be h_0 and X_1. Likewise, the input for the phase that follows would be h_1 from the previous step, and so on. As a result, when training, it remembers the context.

2.3.2. Long short-term memory (LSTM)

LSTM networks belong to the category of recurrent neural networks and are capable of capturing order dependencies in sequence prediction tasks. LSTM control flow is identical to the RNN control flow. As it moves forward, the data undergoes processing before being transmitted, and the functions performed within the LSTM cells vary (Nguyen, 2018). LSTM has a lot more nodes inside compared to RNN and with two inputs and outputs since it keeps track of the long- and short-term memories.

The fundamental building block of LSTMs is the cell state as well as its multiple gates. While moving through the sequence loop, each cell state functions as a conduit that carries pertinent information (Nguyen, 2018). It can be compared to the network's memory. During the sequence processing, the cell state has the capability to convey crucial information. This enables the data obtained from earlier time steps to flow directly into subsequent time steps, mitigating the influence of short-term memory. As the cell state progresses, the gates regulate the addition or removal of information from it. The gates consist of a group of neural networks that determine which information about the cell state should be retained. During training, the gates can learn what information is crucial to remember or discard.

Memory blocks that are repeatedly connected blocks make up an LSTM layer. Each memory block contains multiple connected memory cells, as well as three multiplicative units - input, output, and forget gates - that serve as control mechanisms analogous to read, write, and reset operations for the cells (Graves & Schmidhuber, 2005).

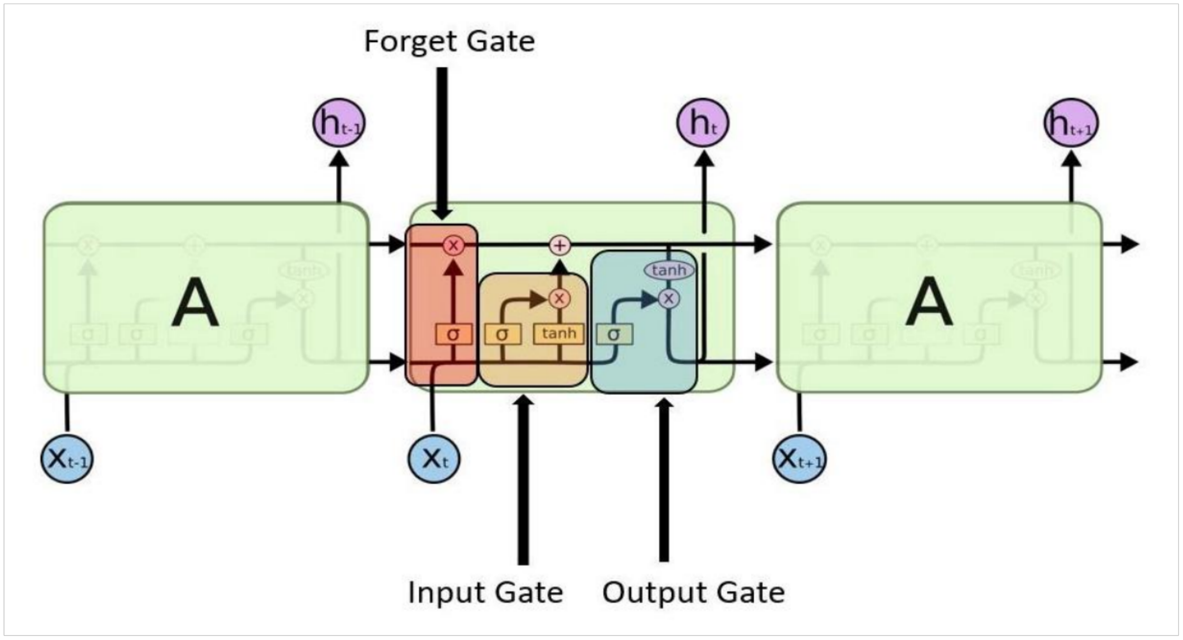


Figure 2.3 The Structure of LSTM Network (Mittal, 2020)

The three gates inside the LSTM network are shown in Figure 3 above.

In order to alter the memory, the input gate determines which input value should be utilized. The sigmoid function defines which integers can pass through the 0,1. The *tanh* function gives the input values weight and determines overall relevance level, which ranges from -1 to 1.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Input gate Equation (Mittal, 2020)

The forget gate is responsible for deciding which information within the block should be erased. This is determined by sigmoid function. The forget gate examines both the previous state (*h_{t-1}*) and the input content (*X_t*) for each value in the cell state (*C_{t-1}*) as well as producing a number ranging between 0 and 1 (omitting or keeping respectively).

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

Forget gate Equation (Mittal, 2020)

The input as well as the memory of the block are used to decide the output of the output gate. The sigmoid function sets the threshold for which values are allowed to pass through, limiting them to values between 0 and 1. The tanh function then multiplies the output of the sigmoid by the weight given to the input data, indicating its level of importance on a scale from -1 to 1.

$$\begin{aligned} o_t &= \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right) \\ h_t &= o_t * \tanh\left(C_t\right) \end{aligned}$$

Output gate Equation (Mittal, 2020)

2.3.3. Gated Recurrent Units (GRU)

GRU belongs to the family of RNN designs that governs and manages overall information flow across cells inside the neural network using gating mechanisms., as its name suggests (Cho et al., 2014).

The design of GRU enables it to dynamically capture relationships in long sequences of data while retaining the information from earlier parts of the sequence. This is done through its gating mechanisms, that are comparable to those found in LSTMs and are responsible for selecting whether data should be kept or erased at every time step (Le, Yapp & Yeh, 2019).

GRUs have only hidden state which is transmitted across time steps, whereas LSTMs have cell state as well as hidden state that keep short and long-term memories. This can retain short-term as well as long-term dependencies at the same time due to the gating techniques and computations which the hidden state as well as incoming information undergo.

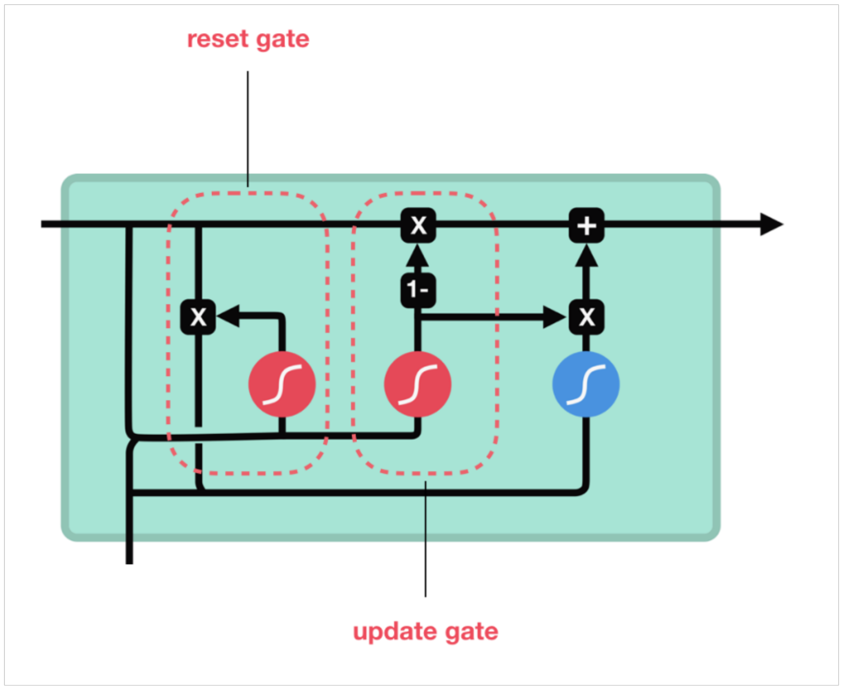


Figure 2.4 The Structure of GRU Network (Nguyen, 2018)

The gates found in the GRU cell are the Update as well as the Reset gates. Just like LSTMs gates, gates in GRU are taught to precisely block out any extraneous input while keeping what's really relevant (Kostadinov, 2017).

2.4. Research Gaps

In the realm of E-learning systems and online examinations, several critical research gaps come to the forefront. Foremost among these is the challenge of effectively detecting impersonation during E-exams. While various authentication methods have been proposed, none have proven entirely successful in mitigating this issue, emphasizing the need for the development of more robust and precise impersonation detection techniques.

Additionally, the feasibility of utilizing biometric technology for authentication purposes has been put forward, yet its adoption is hindered by the high costs associated with specialized sensors and constraints regarding sensor coverage. Research endeavors should thus concentrate on identifying cost-effective alternatives or enhancing the accessibility of biometric solutions within educational institutions.

Furthermore, the adaptation of authentication methodologies to suit essay-based exams represents another significant gap. Many existing solutions, such as the E-cheating intelligent agent and the mouse tracking technique, are better suited for multiple-choice questions, leaving a void in research concerning the adaptation of these methods or the creation of novel techniques tailored specifically for essay-based assessments.

In the realm of real-time detection, criticism has been directed towards the mouse tracking technique due to its inability to operate in real-time. Therefore, research efforts should be dedicated to the development of real-time detection mechanisms capable of identifying impersonation and other forms of cheating during E-exams as they occur.

Lastly, the application of keystroke dynamics as an authentication method has been proposed, but it is marred by challenges related to hardware limitations and timing variations. Extensive research should be conducted to explore avenues for overcoming these challenges and rendering keystroke dynamics a viable option for E-exam authentication.

In summary, the identified research gaps center around the enhancement of impersonation detection methods, the pursuit of cost-effective biometric solutions, the adaptation of authentication techniques for essay-based exams, the development of real-time detection mechanisms, and the resolution of challenges associated with keystroke dynamics. Tackling these gaps is pivotal for fortifying the security of E-exams within the domain of E-learning systems.

CHAPTER THREE

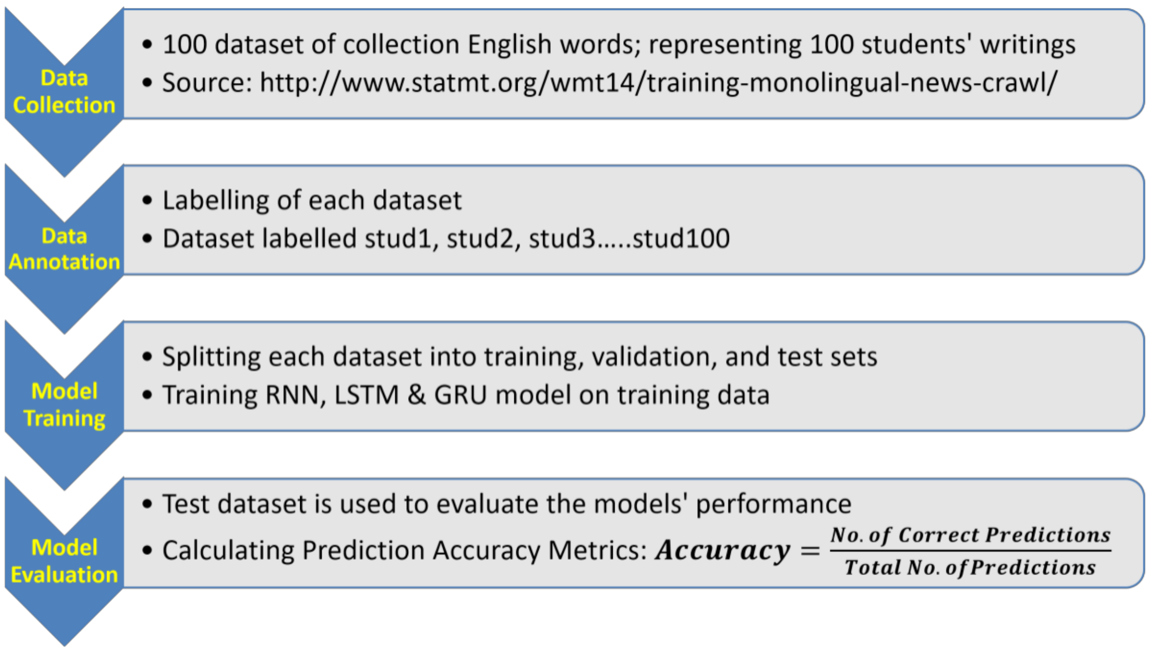
METHODOLOGY

3.1 Introduction

This section explains how the approach used to meet the research's goals was developed. It includes how data was collected, data preprocessing, designing word-level and character-level RNN, LSTM and GRU models, as well as the performance comparison of the models in the two levels. We also compare our model's performance to that of other existing models.

3.2 Methodology Flowchart

The diagram below outlines the steps that were taken to undertake this research.



Data Collection
• 100 dataset of collection English words; representing 100 students' writings
• Source: http://www.statmt.org/wmt14/training-monolingual-news-crawl/
Data Annotation
• Labelling of each dataset
• Dataset labelled stud1, stud2, stud3.....stud100
Model Training
• Splitting each dataset into training, validation, and test sets
• Training RNN, LSTM & GRU model on training data
Model Evaluation
• Test dataset is used to evaluate the models' performance
• Calculating Prediction Accuracy Metrics: $\boldsymbol{Accuracy} = \frac{\boldsymbol{No. of\ Correct\ Predictions}}{\boldsymbol{Total\ No. of Predictions}}$

Figure 3. 1 Methodology Flowchart

3.3 Experimental Setup

3.2.1 Dataset

A collection of words in a form of essay was required for training the models in this research. This set of words represents writing of essay by 100 students in an actual e-exam. Therefore 100 hundred sample data was retrieved from a pool of existing data (at http://www.statmt.org/wmt14/training-monolingual-news-crawl/). Data was in text form in a text file of size 286 MB with English words. This dataset represents dummy writings of students.

The dataset was used to train the models in two scenarios; in the first one it was utilized in the word-level RNN, LSTM as well as GRU models and the other one in character-level RNN, LSTM as well as GRU models. Therefore, the same dataset was used in both cases, where in the first instance, the data for training was divided into words, so, the dataset contained 453668 unique English words. In the second instance the training data got divided to 285579163 English characters. These datasets provided us with enough words and characters to evaluate the effectiveness of our models.

3.2.2 Data pre-processing

In data preprocessing, spaCy is used for natural language processing because the data must be represented in a computer-readable format (spaCy, n.d.). SpaCy is a Python-based NLP library that comes with a lot of capabilities in-built within.

When the dataset is passed into spaCy, the following tasks are performed:

Sentence Detection. The beginning and ending of sentences in the dataset are defined here, allowing the text to be divided into linguistically units of meaning. After sentence identification, the next stage is tokenization. It enables you to recognize the text's basic units. Tokens are the fundamental units. Tokenization is beneficial since it divides a text into logical components. The other step is Lemmatization. This is when a word's inflected forms are reduced while still guaranteeing that the reduced form is linguistically acceptable. *Organized*, *organizes*, as well as *organizing*, for example, all are synonyms for *organize*. Organize is the lemma in this.

Part of speech (POS) tagging is the next step. Each token is given a POS tag based on how it is used in the phrase. The interjection, conjunction, preposition, adverb, verb, pronoun, adjective as well as noun are the eight components of speech. Each word can be assigned a syntactic category using POS tags. One of the phases in extracting data from a dataset is rule-based matching. It was used to discover and extract tokens and trends based on grammatical characteristics and patterns, like lowercase, as part of speech. Another step in extracting a sentence's dependency parse to describe its grammatical structure is dependency parsing. It establishes the relationship between headwords and their subordinates. Dependency parsing reveals a word's role in a text as well as how different words are related to one another.

Another process performed is Named Entity Recognition (NER). This is the act of finding identified entities inside an unstructured dataset as well as classifying them into pre-defined classes like organizations' names, people's names, percentages, places, time expressions, and monetary amounts, among other things. Finally, the dataset is represented as a sequence of integer values, with each word in the text file having its own integer value. This process was achieved using neural network embedding layer. This allowed the text data to be consumed in the neural networks.

3.2.3 Model Parameters

This involved setting up necessary parameters required in machine learning modeling. These parameters encompass batch size, denoting the quantity of data in each batch supplied to the models, and it was configured to 64. Batch size of 64 was established as an optimum batch size that led to faster convergence of the training. The selection of 64 as the optimum batch size was as a result of iterative testing and experimentation to find the best configuration for the particular training scenario. The embedding size is a parameter that indicates the number of characteristics or features in the dataset and is configured to have a value of 256. Sequence length parameter corresponds to the number of iterations the dataset is run through our model, it is set to 50. The buffer size parameter is set to be 10,000. The last parameter is neurons which corresponds to the size of the hidden state of the models and is set at 1024. The learning rate of 0.001 was defined for this research. Choosing the learning rate was based on trial and error, I started with a reasonable initial value and adjusting it during training to achieve the best convergence and model performance. In calculating the prediction accuracy, the equation was used:

$$Prediction\ Accuracy = \frac{Sum\ of\ Correct\ Predictions}{Total\ Prediction}$$

3.2.4 Baseline Models

In this section, we conduct a comparison between our model and other state-of-the-art models. We refer to our model as Impersonation Detector. The performance of Impersonation Detector will be compared with the performance of:

1. E-cheating intelligent agent that integrates IP detector and behavior detector protocols. Its performance was tested using DNN, DenseLSTM, LSTM and RNN (Tiong & Lee, 2021).

2. Mouse-Tracker that uses a mouse-tracking technique to track students' behavior and uses a SVM to categorize as well as predict pupils who engage in illegal behavior (Sokout et al., 2020).

3. Keystroke dynamics used for authenticating students during online exams. This model was tested and evaluated using GMM-UBM (Mattsson, 2020).

3.4 Proposed Model

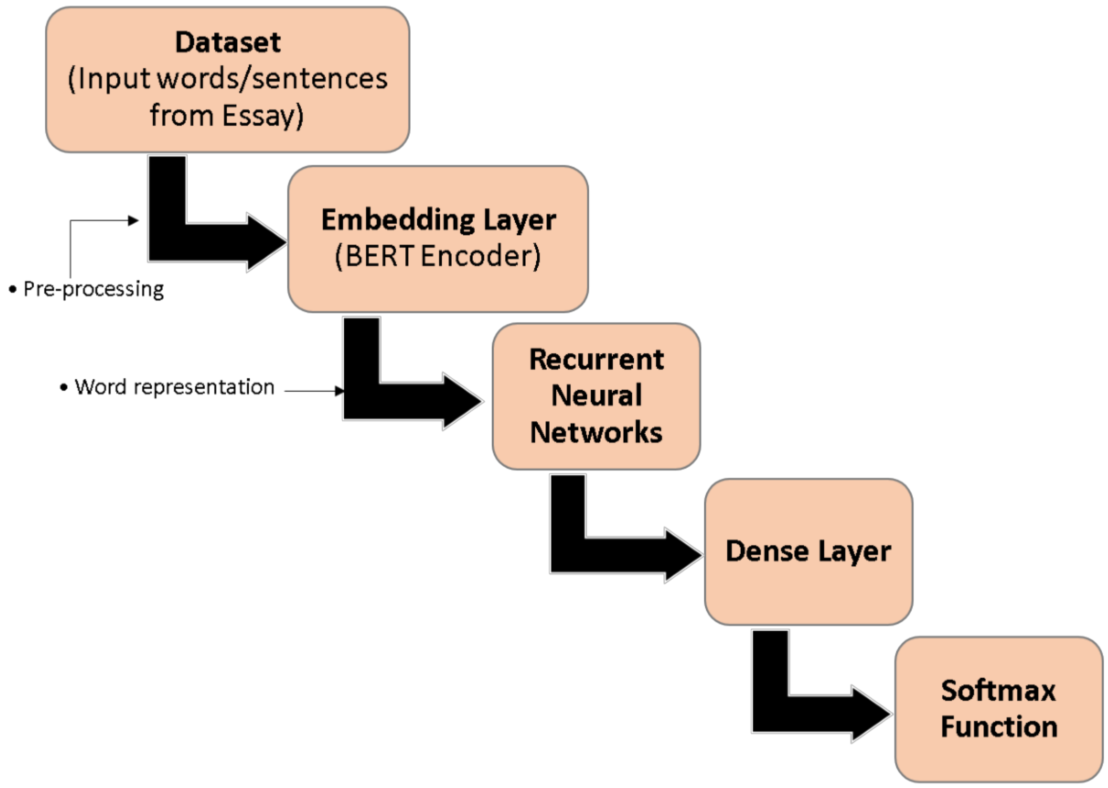


Figure 3.2 Proposed Model

3.4.1 Dataset Layer

This is the first layer in the model. It provides the model with set of words or sentences in a form of an essay for training and testing.

3.4.2 Embedding Layer

This is the second layer in the model in all the neural networks designs (i.e. RNN, LSTM and GRU). It involved word embedding, which is the process of learning a representation for text in which words with related meanings are represented similarly (Brownlee, 2017).

To perform embedding we exploited Bidirectional Encoder Representations from Transformers (BERT) framework (Guo et al., 2020). Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art natural language processing (NLP) framework developed by Google in 2018. BERT is designed to understand the context of words in a sentence or text, allowing it to capture the intricacies of language and perform exceptionally well on a wide range of NLP tasks.

Sentence Encoding

Here, we encode the whole sentence using BERT. Given the datasets described in section 3.2.1, we use SpaCy for sentence chunking. For each sentence we fine-tune BERT to generate the sentence representation. BERT encodes a sentence by breaking it into subword tokens, converting them into word embeddings, and adding positional and segment embeddings. It uses a stack of transformer layers for bidirectional context understanding, capturing complex relationships between words. This process enables BERT to effectively understand and represent the entire sentence. For each sentence BERT generates a vector of dimension i.e. R^d dimension. BERT uses a dimension of 512.

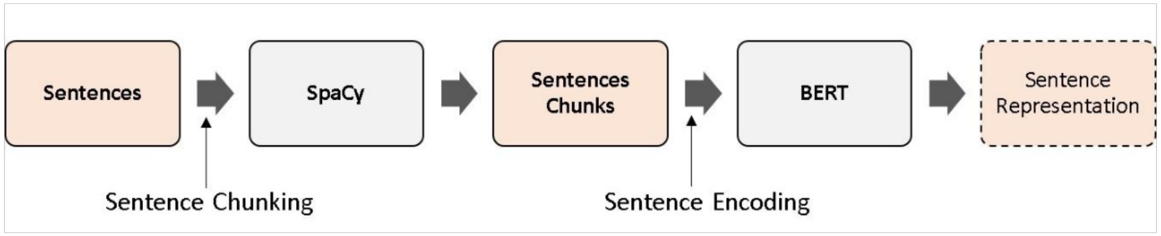


Figure 3.3 Sentence encoding using BERT

Word Encoding

Here, we encode words as opposed to the whole sentence using BERT. Given the dataset described in section 3.2.1, SpaCy is used to extract sentences. The sentences are then fed into BERT. BERT now configured to generate the word level representation of a given sentence. BERT encodes words through tokenization into subword tokens, mapping them to embeddings, adding positional embeddings for word order, and processing them using transformer encoder layers. This process captures contextual information and word meaning within the input sequence, enabling BERT to understand words in their full context bidirectionally. In this case BERT generates a matrix of dimension dxn where d is a word's vector representation and n is the words' number in a sentence i.e. R^{dxn} dimension. BERT uses 512.

3.4.3 Recurrent Neural Networks Layer

This is the third layer of the model where patterns and sequences of the dataset is being learnt by the models. The models trained can be either RNN, LSTM or GRU.

RNN Layers

A RNN is a type of neural network that possesses internal memory and operates in a feed-forward manner. The recurrent nature of the RNN is due to the fact that it executes the same operation for each input of data, and the result of the current input is influenced by the previous computation (Mittal, 2020). After generating the output, it is duplicated and sent back into the recurrent network. During the decision-making process, the network takes into account the current input and the output it has acquired from previous inputs.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = \tanh(a^{(t)})$$

$$o^{(t)} = c + Vh^{(t)}$$

$$\hat{y}^{(t)} = softmax(o^{(t)})$$

RNN Layers Equations (Mittal, 2020)

where the hidden state *h(t)* of the network serves as its "memory" and reflects the hidden state at time *t*. The network's output is represented by *o(t)*, while *y(t)* represents the network targets. At each time step *t*, the input to the network is *x(t)*. The remaining parameters consist of bias vectors b and c, along with weight matrices *U* for input-to-hidden connections, *V* for hidden-to-output connections, and *W* for hidden-to-hidden connections.

In this research, the RNN model was implemented with three hidden layers. The neural network consisted of three layers, where the first and second layers had 400 neurons each and the third layer had 224 neurons.

LSTM Layer

RNNs are known to have issues with short-term memory retention. When dealing with lengthy sequences, RNNs may struggle with propagating information from earlier time steps to the later ones. During back propagation, the vanishing gradient challenge impacts RNNs. The problem of vanishing gradients arises when a gradient decreases in magnitude as it propagates backward through time. If the magnitude of a gradient falls below a certain threshold, it becomes ineffective in contributing to the learning process (Nguyen, 2018). In rnns, layers that receive a minor gradient increase stop learning. All these are usually the first layers to appear. Since this layers don't learn, rnn do not remember what they've been through in lengthier sequences, leading to a short-term memory.

LSTMs are more refined kind of rnns which make it much easier to remember prior events by introducing into the network a memory unit known as a cell (Yan, 2016). LSTM networks have internal units referred to as gates which control the transmission of data. The gates are capable of deciding which pieces of information inside a series should be retained as well as which should be deleted. It has the ability to send pertinent data along a long chain of series in this way, allowing it to make predictions (Nguyen, 2018). Here, the RNN's vanishing gradient problem is solved. LSTM is highly adapted to categorize, analyze, and forecast unexpected time gaps in time series. The model is trained via back-propagation.

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

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

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

LSTM Layers Equations (Olah, 2015)

The letters "i_t", "f_t", "o_t", and "a" represent the input gate, forget gate, output gate, and sigmoid function, respectively. The weights for each gate's neurons are represented by "$w_{(x)}$" and the biases for the relevant gates are represented by "$b_{(x)}$". In the LSTM block, "h_{t-1}" refers to the output from the previous timestamp (t-1), while "x_t" refers to the input at the current timestamp (Olah, 2015).

The LSTM model was developed with two instances, with one layer and with two layers. This was done to determine the effectiveness of the LSTM when the depth of the network is increased by one layer. In the first instance, the LSTM layer had one hidden layer with 1024 as number of nodes within the LSTM cell. The second instance, the LSTM layer had two hidden layers. There were 800 nodes in the first hidden layer and 224 nodes in the second hidden units.

GRU Layer

Cho et al. (2014) suggested gated recurrent unit that enables every recurring unit to gather relationships throughout temporal scales in an adaptable manner. Similar to the LSTM, the GRU also utilizes gating units to regulate the flow of information within the unit. However, unlike the LSTM, the GRU does not use discrete memory cells. Both the GRU and LSTM networks have the ability to comprehend the relationships, both long and short-term, present in sequential data, however GRU networks have fewer parameters and are hence faster to train.

A reset and update gate is a concept in a GRU network that helps guarantee memory isn't taken over by tracking short-term dependencies.

The following formula is used for calculating updating gate z_t for timestep t

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Update gate Equation (Kostadinov, 2017)

Whenever x_t is connected to a network layer, $W^{(z)}$ which is it's own weight is used to be multiplied with x_t. Similarly, the statement applies to h_{t-1}, which not only retains information from previous t-1 time steps but also undergoes multiplication with its weight $U_{(z)}$. By using sigmoid activation function, the values are combined and the outcome is squeezed between 0 and 1.

The model employs this formula to calculate the reset gate, which determines the amount of past information that should be disregarded:

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Reset gate Equation (Kostadinov, 2017)

The formula used for the reset gate is identical to the one used for the update gate. The weights and how the gate is used are the key differences. $h_{(t-1)}$ and x_t are connected to the model, multiplied by their respective weights, summed, and the sigmoid function is applied.

To calculate a fresh content in the memory which utilizes the reset gate to retain pertinent historical data, this formula is used:

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

Current memory content Equation (Kostadinov, 2017)

Subsequently, the network computes the h_t vector, which stores the current layer's information and transfers it down to the subsequent network unit through the update gate. It is done as follows:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^{'}$$

At the present time step in the final memory Equation (Kostadinov, 2017)

In its implementation, the GRU layer had two instances as well. The first instance was where we had the GRU layer with one hidden layer and we used 1024 neurons or nodes within the GRU cell. In the second scenario, a GRU layer was utilized with dual hidden layers. The foremost hidden layer comprised 800 neurons, and the succeeding one consisted of 224 neurons.

3.4.4 Dense Layer

This is the last layer in RNN model. The dense layer learns a weight matrix, where the first dimension of the matrix is the input data′s dimensionality, and the second one is the output data's dimension. The layer utilizes activation functions to convert input signal of nodes in a neural network to an output signal (Walia, 2018).

Here, softmax action function was used. The Softmax function converts numbers into one-to-one probabilities and returns a vector that describes the probability distributions of a set of possible outcomes (Kouretas & Paliouras, 2019). The function is typically used to calculate predicted losses when training a dataset. The input shape of the layer was (453668,) with the number of neurons was 1024.

3.5 Comparing the Performance of the LSTM, RNN, and GRU Models

The three algorithms' effectiveness in word-level and character level will be analyzed by considering the accuracy rates based on accuracy, processing time, and resources needed to execute the tasks. These outputs will be plotted on graphs against each model. The graph will be showing number of iterations and loss percentage at given iteration. Then a consolidated graph for all the models will be developed and results be analyzed. The best performing model will be chosen for recommendation and implementation on eLearning environments.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter highlights the findings of the study from training and testing the RNN, LSTM and GRU models as described in chapter three. The results are then discussed, outline what they mean and the performance comparison to other similar studies is done.

4.2 Data Analysis

The operating system used for this experiment was Windows 10 64bit Professional Edition running on a computer hardware with a processing power of Intel(R) Core(TM) i5-6400 CPU @ 2.70 GHz, 2712 MHz, 4 Core(s), 4 Logical Processor(s). The hardware had physical random access memory of 8 GB and 9.74 GB of total virtual memory.

The RNN, LSTM and GRU models were developed using Tensorflow. TensorFlow framework is a machine learning environment with end-end workflow. It has a comprehensive set of techniques, modules, as well as community resources that support researchers to enhance the cutting-edge technique in machine learning and developers to develop and implement ML-powered apps rapidly. TensorFlow is the only framework that can run machine learning models on everything from the cloud to the tiniest microcontroller device. TensorFlow models may be optimized for both CPU and GPU. TensorFlow.js, on the other hand, is a JavaScript-based framework for running machine learning models in the browser. With no changes to the code, any modern browser can execute the TensorFlow model.

With these working environment specification, the time taken to preprocess data was 12.95 seconds.

Table 4.1 below shows percentage of detection accuracy of the three models, i.e. RNN, LSTM as well as GRU, after we trained and tested them for the same period of time. The epochs for each model in the specified period of time was not considered since our interest was in detection accuracy against the time taken for the model to get to optimum performance.

Table 4.1: Evaluated results of RNN. LSTM and GRU Models in Word-level

Model Name	**Word-level Accuracy at (in minutes)**					
	5 min.	**20 min.**	**40 min.**	**60 min.**	**80 min.**	**100 min.**
RNN models	8%	42.6%	67.7%	78.8%	82.9%	83.9%
LSTM models	21.8%	54.9%	77.6%	86.3%	92.2%	92.3%
GRU models	32.4%	56.4%	82.5%	86.6%	97.7%	98.6%

The initial detection accuracy percentage for each model was recorded after five minutes and the subsequent accuracy percentage were taken in the intervals of 20 minutes during the evaluation, as per the above table. There was no significant change in detection accuracy percentage after 100th minute in all the three models (RNN, LSTM as well as GRU).

In the first 5 minutes the RNN recorded accuracy of 8%, the LSTM recorded 21.8% accuracy and the GRU model achieved 32.4% accuracy. The highest detection accuracy percentage for all the three models were obtained after 100 minutes. With a precision of 98.6%, the GRU model was the most accurate. The LSTM model was the second highest and demonstrated a comparable accuracy of 92.3% in detecting impersonation in an essay-based e-exam. On the other hand, the RNN model exhibited the poorest performance, achieving an accuracy score of 83.9%.

Table 4.2 Evaluated results of RNN. LSTM and GRU Models in Character-level

Model Name	**Character-level Accuracy(%) at (in minutes)**								
	5 min.	**20 min.**	**40 min.**	**60 min.**	**80 min.**	**100 min.**	**120 min.**	**140 min.**	**160 min.**
RNN models	5.3%	38.6%	61.5%	69.6%	72.4%	73.1%	74.9%	77.4%	83.8%
LSTM models	17.8%	52.9%	61.6%	67.5%	70.2%	75.6%	81.3%	85.9%	92.4%
GRU models	29.3%	56.4%	62.5%	70.6%	77.5%	81.2%	85.4%	91.6%	98.1%

The highest percentage prediction accuracy for all the three models was observed after 160th minute as show in the Table above.

4.3 Discussion of results

Word-level models outperformed character-level models for several reasons. Firstly, character-level models have limitations in capturing long-distance dependencies in text. Additionally, as the sequence length increased, character-level models required more training time. Furthermore, character-level models may encounter challenges in capturing the higher-level semantics and meaning found in words, often resulting in longer sequences of characters to convey the same information compared to word-level models.

The performance accuracy of RNN model was the lowest at 83.7% detection accuracy compared to other models because it has short-term memory problem. If the series becomes excessively long, the RNN model faces difficulty in retaining information from earlier time steps to the later ones. The Vanishing Gradient Problem is the name for this weakness. To train an RNN model, you back-program the network through time step, then calculate the gradient at every time step. This gradient is being used to update training weights of the network. If the previous effect of the layer on the current layer is low, then the gradient value would be low, and conversely. Whereas if gradient of the previous layer gets smaller, then gradient of the following layer will become even smaller. These gradients would decrease drastically while back-propagating. A lesser gradient would have no impact on weight updating. Therefore, the network doesn't really remember how prior inputs affect it, and due to this, the short-term memory loss occurs. The equation used to calculate the weight at any given time as show below:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Where h_t refers to the state of the memory at time$t$, and x_t refers to the impute at time t.

The best detection accuracy of GRU and LSTM models achieved at 98.6% and 92.3% respectively is attributed to the presence of memory cells which allow retention of any data without a lot of loss. In addition, they possess gates that assist in controlling the transmission of information to the cell state, and these gates can discern which information in a sequence is important and which is not.

The initial output we observe in a GRU cell is the cell state h_t which corresponds to the output at time t.

$$\tilde{h}_t = \tanh(W \, . \, [r_t * \, h_{t-1}, x_t])$$

The equation preceding that one displays the updated value or candidate which has the potential to replace the cell state at time t. The updated value or candidate, which could substitute the cell state at time t, is calculated using the cell state at the previous time step h_{t-1} and a relevance gate r_t that decides the importance of the previous cell state in computing the current cell state.

$$r_t = \sigma \, (W_r \, . [h_{t-1}, x_t])$$

As indicated, the relevance gate r_t is activated by a sigmoid function that outputs a value between 0 and 1, determining the level of importance assigned to the previous information, and is then incorporated into the updated value or candidate.

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

The revised version of the candidate, represented by $\tilde{h}_t$, is a blend of the previous cell state, h_{t-1} and the current cell state, h_t but filtered to some extent. The amount of the updated candidate to be used in computing the current cell state, as well as the portion of the prior cell state to be retained, are both determined by the update gate, z_t.

$$z_t = \sigma(W_z \, . [h_{t-1}, x_t])$$

The sigmoid function used in the update gate, similar to the relevance gate, plays a crucial role in helping the GRU to maintain the cell state for the necessary duration. By doing so, the GRU is able to store information effectively, and this helps to overcome the Vanishing Gradient Problem that is commonly faced by RNN models.

While the basic concept is the same, an LSTM is a more complicated network. The LSTM has three gates: the forget gate f_t, the update gate i_t, and the output gate o_t. The GRU has two gates: the update gate and the relevance gate. The cell state was equivalent to the activation state/output in GRU, but they aren't quite the same in the LSTM. h_t represents the output at time t, and C_t represents the cell state.

Although the fundamental idea is similar, an LSTM is a more complex network compared to a GRU. Unlike GRU, which has two gates - the update gate and the relevance gate - LSTM has three gates - the forget gate f_t, the update gate i_t, and the output gate o_t. Moreover, while the cell state in GRU is equivalent to the activation state or output, in LSTM, they are not entirely the same. In LSTM, h_t denotes the output at time t, while C_t represents the cell state.

$$\tilde{C}_t = \tanh(W_C . [h_{t-1}, x_t] + b_C)$$

Similar to GRU, the candidate value $\tilde{C}_t$ of the cell state at time t in LSTM also depends on the previous output h_{t-1} and the input x_t.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Like in GRU, in LSTM, the current cell state C_t is a filtered version of the previous cell state and the candidate value. However, in LSTM, this filtering process is determined by two gates - the update gate and the forget gate. The value of $(1 - \mathrm{updateGate}_t)$ in GRU is quite similar to the forget gate. Sigmoid functions are used in both the forget gate and the update gate.

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

The forget gate in LSTM determines the extent to which the information from the previous cell state is required to be retained in the current cell state.

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i)$$

In LSTM, the update gate determines the amount of the candidate value $\tilde{C}_t$ that needs to be incorporated into the current cell state. Between 0 and 1 is the value of both the update and forget gates.

$$O_t = \sigma(W_O . [h_{t-1}, x_t] + b_O)$$

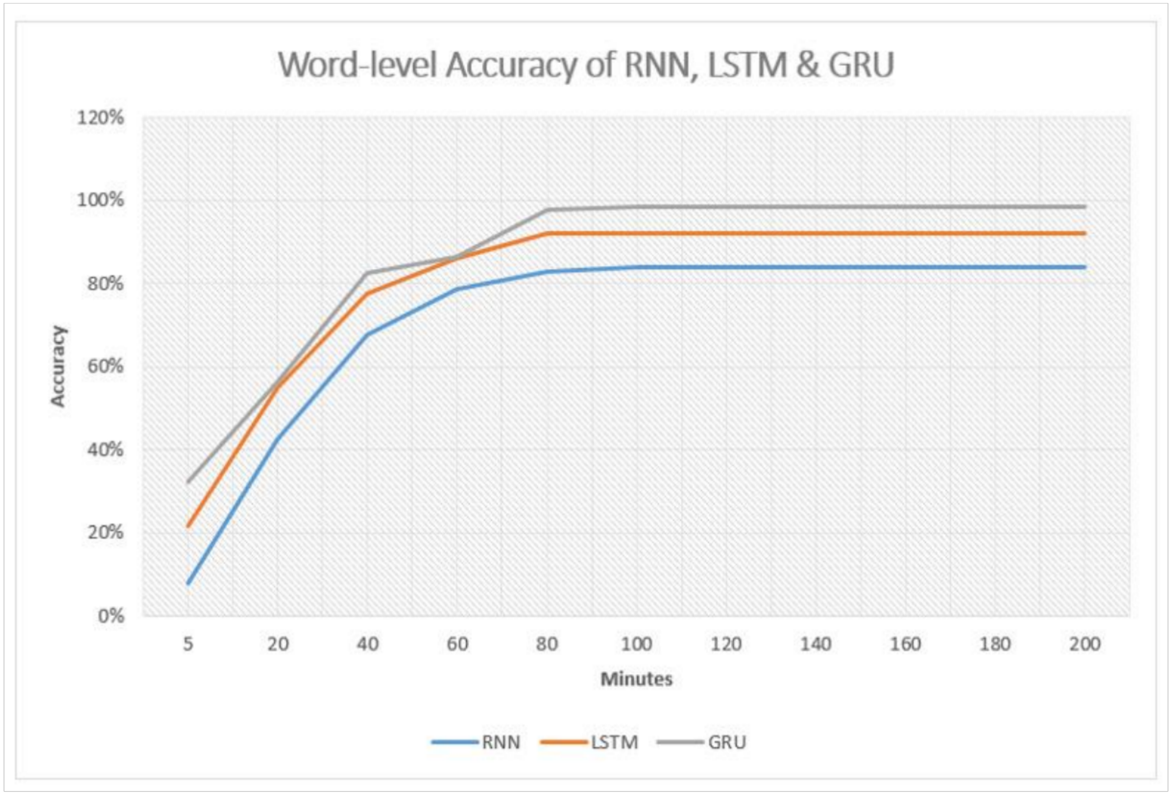
$$h_t = O_t * \tanh(C_t)$$

Finally, we must determine what we will produce. The current cell state in LSTM is a filtered representation, which is passed through a $tanh$ layer to restrict the values between -1 and 1. Next, the output is multiplied by a sigmoid activation function through the output gate to ensure that only the desired information is outputted

Compared to LSTM models, GRU models are simpler and require less computational power. This allows for the creation of very deep networks using GRUs. However, LSTM models are more powerful due to the presence of a greater number of gates. Nonetheless, due to their increased complexity, LSTM models require more computational resources to train and use effectively.

Figures 4.1 and 4.2 present curved graphs that display the trend in the performance of RNN, LSTM, and GRU models in word-level and character-level respectively.

Figure 4.1: Performance comparison of RNN, LSTM and GRU Models in Word-level



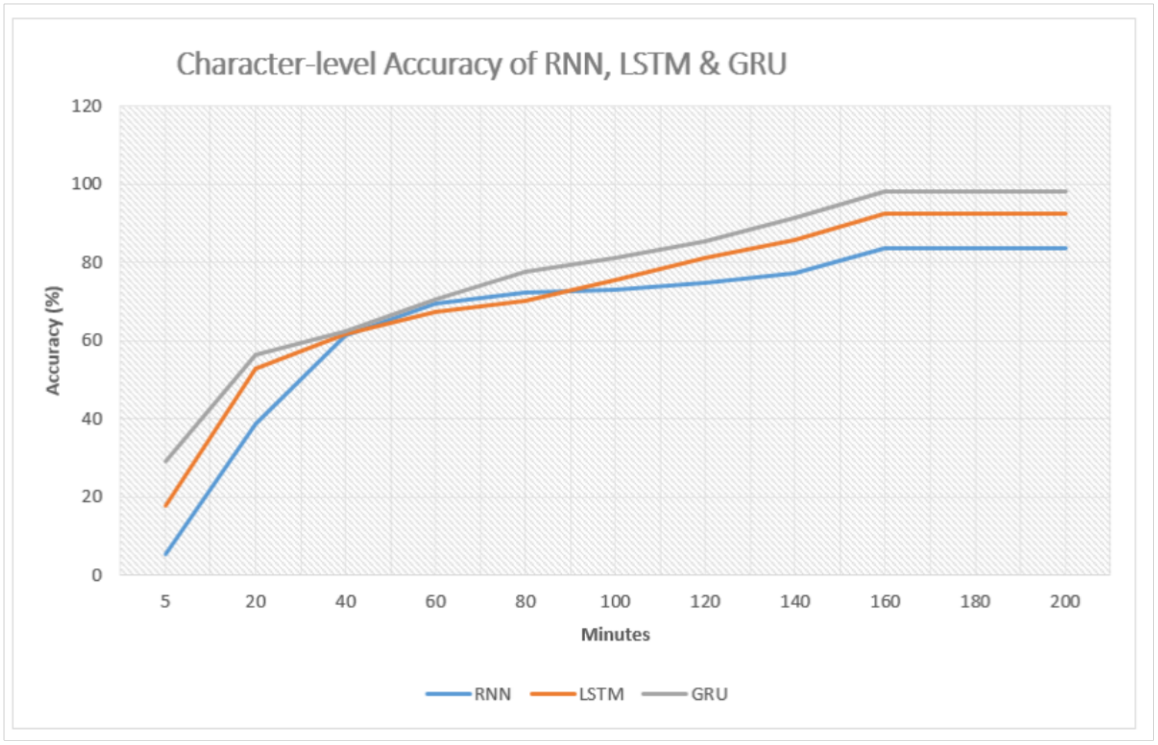


Figure 4.2 Performance comparison of RNN, LSTM and GRU Models in Character-level

4.4 Comparison with other state of art models

The effectiveness of the RNN, the LSTM and the GRU models in detecting impersonation in an essay-based e-exam was benchmarked against similar studies on detecting cheating in online examinations. The comparison is based on performance accuracy of the models used in each study. A table comparing the accuracy of different studies in detecting cheating in e-exams is provided below.

Table 4.3 Comparison Models Accuracy in Similar Studies

Tool	**Reference**	**Model Name**	**Best accuracy (%)**
Impersonation Detector	This study	RNN models	83.9
		LSTM models	92.3
		GRU models	98.6

E-cheating intelligent agent	(Tiong & Lee, 2021)	Dense LSTM	95.32
Mouse-Tracker	(Sokout et al., 2020)	Support Vector Machine (SVM)	94
Keystroke dynamics	(Mattsson, 2020)	Gaussian Mixture Models with Universal Background Model	94.5

From this benchmarking, our research indicates that among the models studied, the GRU model exhibited the highest accuracy, at 98.6%. This makes our tool, Impersonation Detector, perform the best in detecting cheating in an online exam.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

5.1 Introduction

This chapter serves as the culmination of the research efforts, encompassing an overview of the study's achievements, contributions to the field, and recommendations for future work.

5.2 Research Achievements

The primary aim of this study was to devise a machine learning algorithm capable of detecting impersonation and plagiarism during essay-based online examinations within an E-learning environment. This objective was successfully met through a series of steps. Initially, essays composed by students were employed for model training. A predefined collection of English words, sourced from an online dataset, represented the students' inputted words. Subsequently, real-time LSTM, RNN, and GRU models were developed using Python to detect impersonation and plagiarism, considering both words and extracted characters. The first objective of creating such an algorithm was achieved. The models were then trained using Tensorflow and data preprocessed with spaCy and BERT, fulfilling the second objective. Evaluation of the models, based on epoch durations, led to the achievement of the third objective. Ultimately, the best-performing model, GRU, achieved an impressive accuracy of 98.6% in detecting impersonation and plagiarism in e-exams. Hence, it can be concluded that the developed and trained machine learning algorithms excel in impersonation detection during essay-based e-exams, meeting all set objectives.

5.3 Research contributions

This research contributes to the field by advancing the understanding of training RNN, LSTM, and GRU neural networks for predicting impersonation and plagiarism in online examinations. It enhances existing cheating detection methods in online exams, offering a more dependable approach for identifying student impersonation in essay-based online assessments. The findings hold promise for application in real-world online exams, thereby bolstering E-learning's efforts in verifying student identities and the authenticity of their work during online assessments. Furthermore, researchers in this domain can benefit from the developed models, saving time and effort by building upon the existing models for their investigations.

5.4 Recommendations and future work

For future studies, utilizing data generated by actual students in real online exams is recommended to assess various approaches while minimizing biases present in training data. Exploring diverse deep learning algorithms and their performance compared to the models developed in this research is an avenue for further investigation. Large-scale deployments to assess the accuracy of these models in detecting irregularities in E-learning systems represent another potential avenue for future research.

REFERENCES

Aaron, L. S., & Roche, C. M. (2013). Stemming the tide of academic dishonesty in higher education: It takes a village. *Journal of Educational Technology Systems, 42*(2), 161-196.

Almaiah, M. A., Al-Khasawneh, A., & Althunibat, A. (2020). Exploring the critical challenges and factors influencing the E-learning system usage during COVID-19 pandemic. *Education and Information Technologies, 25*, 5261-5280.

Amidi, A., & Amidi, S. (2019). Recurrent Neural Networks cheatsheet: Cited 28-03-2019. Available at: https://stanford. edu/% 7Eshervine/teaching ….

AV, S. K., & Rathi, M. (2021). Keystroke Dynamics: A Behavioral Biometric Model for User Authentication in Online Exams *Research Anthology on Developing Effective Online Learning Courses* (pp. 1137-1161): IGI Global.

Bawarith, R., Basuhail, A., Fattouh, A., & Gamalel-Din, S. (2017). E-exam cheating detection system. *Int. J. Adv. Comput. Sci. Appl, 8*, 176-181.

Britz, D. (2017). {nd}. WILDML, Recurrent Neural Networks Tutorial, Part1-Introduction to RNNs.

Brownlee, J. (2017). How to use word embedding layers for deep learning with keras. *4 October 2017*.

Cavalcanti, E. R., Pires, C. E., Cavalcanti, E. P., & Pires, V. F. (2012). Detection and evaluation of cheating on college exams using supervised classification. *Informatics in Education, 11*(2), 169-190.

Chalapathy, R., Menon, A. K., & Chawla, S. (2018). Anomaly detection using one-class neural networks. *arXiv preprint arXiv:1802.06360*.

Chen, M., & Chen, C. (2017). *Detect Exam Cheating Pattern by Data Mining.* Paper presented at the FSDM.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Frederiks, A., Derrington, K., & Bartlett, C. (2021). Types of Exams. *Academic Success*.

Gomez-Barrero, M., Drozdowski, P., Rathgeb, C., Patino, J., Todisco, M., Nautsch, A., . . . Busch, C. (2021). Biometrics in the Era of COVID-19: Challenges and Opportunities. *arXiv preprint arXiv:2102.09258*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*: MIT press.

Graves, A., & Schmidhuber, J. (2005). *Framewise phoneme classification with bidirectional LSTM networks.* Paper presented at the Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.

Guo, W., Liu, X., Wang, S., Gao, H., Sankar, A., Yang, Z., . . . Chen, B.-C. (2020). *Detext: A deep text ranking framework with bert.* Paper presented at the Proceedings of the 29th ACM International Conference on Information & Knowledge Management.

Hashim, H., Yunus, M. M., Yusuf, N. S. M., Zanzuri, N. A. H., Ruslee, M. H., & Fakhruddin, S. M. (2018). Factors Influencing Students' Selection of Different Types of Essay in Examination. *Creative Education, 9*(14), 2334.

Hernándeza, J.-A., Ochoab, A., Muñozd, J., & Burlaka, G. (2006). *Detecting cheats in online student assessments using Data Mining.* Paper presented at the Conference on Data Mining| DMIN.

Hu, J., Gingrich, D., & Sentosa, A. (2008). *A k-nearest neighbor approach for user authentication through biometric keystroke dynamics.* Paper presented at the 2008 IEEE International Conference on Communications.

Janelli, M. (2018). E-learning in theory, practice, and research. *Вопросы образования*(4 (eng)).

Javed, A., & Aslam, Z. (2013). An intelligent alarm based visual eye tracking algorithm for cheating free examination system. *International Journal of Intelligent Systems and Applications, 5*(10), 86.

Kahiigi Kigozi, E., Vesisenaho, M., Hansson, H., Danielson, M., & Tusubira, F. (2012). Modelling a peer assignment review process for collaborative e-learning. *Journal of Interactive Online Learning, 11*(2), 67-79.

Kasliwal, G. (2015). Cheating Detection in Online Examinations.

Kausar, S., Huahu, X., Ullah, A., Wenhao, Z., & Shabir, M. Y. (2020). Fog-assisted secure data exchange for examination and testing in e-learning system. *Mobile Networks and Applications*, 1-17.

Kopun, D. (2018). A Review of the Research on Data Mining Techniques in the Detection of Fraud in Financial Statements. *Journal of Accounting and Management, 8*(1), 1-18.

Korman, M. (2010). Behavioral detection of cheating in online examination.

Kostadinov, S. (2017). Understanding GRU networks. Towards Data Science. *Towards Data Science, Towards Data Science, 16*.

Kouretas, I., & Paliouras, V. (2019). *Simplified Hardware Implementation of the Softmax Activation Function.* Paper presented at the 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST).

Kuin, A. (2018). Fraud detection in video record-ings of exams using Convolu-tional Neural Networks.

Le, N. Q. K., Yapp, E. K. Y., & Yeh, H.-Y. (2019). ET-GRU: using multi-layer gated recurrent units to identify electron transport proteins. *BMC bioinformatics, 20*(1), 377.

Mattsson, R. (2020). Keystroke dynamics for student authentication in online examinations.

Mittal, A. (2020). Understanding RNN and LSTM. *URL: https://towardsdatascience. com/understanding-rnn-and-lstm-f7cdf6dfc14e. Accessed*, 01-08.

Nguyen, M. (2018). Illustrated Guide to LSTM's and GRU's: A step by step explanation. *URL https://towardsdatascience. com/illustrated-guide-to-lstms-and-grusa-step-by-step-explanation-44e9eb85bf21.*

Nicholson, C. (2018). A Beginner's Guide to Neural Networks and Deep Learning: The Artificial Intelligence Wiki https://skymind. ai/wiki/neural-network ….

Okada, A., Whitelock, D., Holmes, W., & Edwards, C. (2019). e-Authentication for online assessment: A mixed-method study. *British Journal of Educational Technology, 50*(2), 861-875.

Olah, C. (2015). Understanding lstm networks.

Pertile, S. d. L., Moreira, V. P., & Rosso, P. (2016). Comparing and combining C ontent- and C itation-based approaches for plagiarism detection. *Journal of the Association for Information Science and Technology, 67*(10), 2511-2526.

Rathgeb, C., Pöppelmann, K., & Gonzalez-Sosa, E. (2020). *Biometric Technologies for eLearning: State-of-the-Art, Issues and Challenges.* Paper presented at the 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA).

Shilton, K., & Greene, D. (2019). Linking platforms, practices, and developer ethics: Levers for privacy discourse in mobile application development. *Journal of Business Ethics, 155*(1), 131-146.

Sokout, H., Purnama, F., Mustafazada, A. N., & Usagawa, T. (2020). *Identifying potential cheaters by tracking their behaviors through mouse activities.* Paper presented at the 2020 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE).

Soni, V. D. (2020). Global Impact of E-learning during COVID 19. *Available at SSRN 3630073*.

Tiong, L. C. O., & Lee, H. J. (2021). E-cheating Prevention Measures: Detection of Cheating at Online Examinations Using Deep Learning Approach--A Case Study. *arXiv preprint arXiv:2101.09841*.

Tzafilkou, K., & Protogeros, N. (2020). Monitoring Mouse Behavior in E-Learning Activities to Diagnose Students' Acceptance Items of Perceived Usefulness and Ease of Use. *European Educational Researcher, 3*(1), 21-27.

Ullah, A., Xiao, H., & Barker, T. (2019). A dynamic profile questions approach to mitigate impersonation in online examinations. *Journal of Grid Computing, 17*(2), 209-223.

Urosevic, A. (2019). Student authentication framework for Online exams outside of school.

Walia, A. S. (2018). Activation functions and it's types-which is better?(2017).

Yan, S. (2016). Understanding LSTM and its diagrams. *MLReview. com*.

spaCy. (n.d.). spaCy. https://spacy.io

APPENDIX 1: GRU Algorithm Code

```
import tensorflow as tf
import numpy as np
import time
import os
import matplotlib.pyplot as plt
BATCH_SIZE=64
embedding_size=256
sequence_length=50
BUFFER_SIZE=1000
neuronsa=800
neuronsb=224
data=open("SampleData.txt",encoding="utf8").read()
#data=data.split()
print(len(data))
vocab_size=len(list(set(data)))
words=list(set(data))
char_to_idx={ch:ix for ix,ch in enumerate(words)}
idx_to_char={ix:ch for ix,ch in enumerate(words)}
inputs=[char_to_idx[ch] for ch in data[:-1]]
targets=[char_to_idx[ch]for ch in data[1:]]
inputs_as_tensors=tf.data.Dataset.from_tensor_slices(inputs
)
targets_as_tensors=tf.data.Dataset.from_tensor_slices(targe
ts)
training_data=tf.data.Dataset.zip((inputs_as_tensors,target
s_as_tensors))
training_data=training_data.shuffle(BUFFER_SIZE).batch(sequ
ence_length,drop_remainder=True)
training_dataset=training_data.batch(BATCH_SIZE,drop_remain
der=True)
#print(training_data_final)
def
build_model(vocab_size,embedding_size,neuronsa,neuronsb,bat
ch_size):

model=tf.keras.Sequential([tf.keras.layers.Embedding(vocab_
size,embedding_size,

batch_input_shape=[batch_size,None]),

tf.keras.layers.GRU(neuronsa,return_sequences=True,stateful
=True,recurrent_initializer="glorot_uniform"),
tf.keras.layers.GRU(neuronsb,return_sequences=True,stateful
=True,recurrent_initializer="glorot_uniform"),

tf.keras.layers.Dense(vocab_size)
                              ])
    return model
def training_step(inputs,targets,optimizer):
    with tf.GradientTape() as tape:
        predictions=model(inputs)

loss=tf.reduce_mean(tf.keras.losses.sparse_categorical_cros
sentropy(targets,predictions,from_logits=True))
    grads=tape.gradient(loss,model.trainable_variables)

optimizer.apply_gradients(zip(grads,model.trainable_variabl
es))
    return loss,predictions
smooth_loss =-np.log(1.0/vocab_size)*sequence_length
txt=[]
model=build_model(vocab_size,embedding_size,neuronsa,neuron
sb,BATCH_SIZE)
i=0
lossx=[]
iterations=[]
checkpoint_dir = 'GRU_model_new'
checkpoint_prefix = os.path.join(checkpoint_dir,
"ckpt_{i}")
checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(file
path=checkpoint_prefix,save_weights_only=True)
while(1):
    start = time.time()
    hidden = model.reset_states()
    for x,y in training_dataset:

loss,predictions=training_step(x,y,tf.keras.optimizers.Adam
())
        sampled_indices =
tf.random.categorical(predictions[0], num_samples=1)
        sampled_indices = tf.squeeze(sampled_indices,axis=-
1).numpy()
        for x in sampled_indices:
            txt.append(idx_to_char[x])
        text=tf.strings.join(txt,separator=' ',name=None)
        #print ('----\n %s \n----' % (text.numpy()))
        smooth_loss = smooth_loss * 0.999 + loss * 0.001
        #print(smooth_loss.numpy())
        txt=[]
    print ('Epoch {} Loss {:.4f}'.format(i, loss))
    print ('Time taken for iteration {} is {}
sec\n'.format(i,time.time() - start))
    iterations.append(i)
    lossx.append(loss)
    plt.figure(figsize=(12,9))
    plt.plot(iterations,lossx,label = "Loss Tracker")
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    #plt.xlim(-3, 3)
    plt.xlim(1,400)
    plt.legend(loc='upper right')
    plt.show()
    if i % 5 == 0:
        model.save_weights(checkpoint_prefix.format(i=i))
    model.save_weights(checkpoint_prefix.format(i=i))
    i=i+1
```

APPENDIX 2: LSTM Algorithm Code

```
import tensorflow as tf
import numpy as np
import time
import os
import matplotlib.pyplot as plt
BATCH_SIZE=64
embedding_size=256
sequence_length=50
BUFFER_SIZE=1000
neuronsa=800
neuronsb=224
data=open("SampleData.txt",encoding="utf8").read()
#data=data.split()
print(len(data))
vocab_size=len(list(set(data)))
words=list(set(data))
char_to_idx={ch:ix for ix,ch in enumerate(words)}
idx_to_char={ix:ch for ix,ch in enumerate(words)}
inputs=[char_to_idx[ch] for ch in data[:-1]]
targets=[char_to_idx[ch]for ch in data[1:]]
inputs_as_tensors=tf.data.Dataset.from_tensor_slices(inputs
)
targets_as_tensors=tf.data.Dataset.from_tensor_slices(targe
ts)
training_data=tf.data.Dataset.zip((inputs_as_tensors,target
s_as_tensors))
training_data=training_data.shuffle(BUFFER_SIZE).batch(sequ
ence_length,drop_remainder=True)
training_dataset=training_data.batch(BATCH_SIZE,drop_remain
der=True)
#print(training_data_final)
def
build_model(vocab_size,embedding_size,neuronsa,neuronsb,bat
ch_size):

model=tf.keras.Sequential([tf.keras.layers.Embedding(vocab_
size,embedding_size,

batch_input_shape=[batch_size,None]),

tf.keras.layers.LSTM(neuronsa,return_sequences=True,statefu
l=True,recurrent_initializer="glorot_uniform"),
tf.keras.layers.LSTM(neuronsb,return_sequences=True,statefu
l=True,recurrent_initializer="glorot_uniform"),

tf.keras.layers.Dense(vocab_size)
                              ])
    return model
def training_step(inputs,targets,optimizer):
    with tf.GradientTape() as tape:
        predictions=model(inputs)

loss=tf.reduce_mean(tf.keras.losses.sparse_categorical_cros
sentropy(targets,predictions,from_logits=True))
    grads=tape.gradient(loss,model.trainable_variables)

optimizer.apply_gradients(zip(grads,model.trainable_variabl
es))
    return loss,predictions
smooth_loss =-np.log(1.0/vocab_size)*sequence_length
txt=[]
model=build_model(vocab_size,embedding_size,neuronsa,neuron
sb,BATCH_SIZE)
i=0
lossx=[]
iterations=[]
checkpoint_dir = 'LSTM_model_new'
checkpoint_prefix = os.path.join(checkpoint_dir,
"ckpt_{i}")
checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(file
path=checkpoint_prefix,save_weights_only=True)
while(1):
    start = time.time()
    hidden = model.reset_states()
    for x,y in training_dataset:

loss,predictions=training_step(x,y,tf.keras.optimizers.Adam
())
        sampled_indices =
tf.random.categorical(predictions[0], num_samples=1)
        sampled_indices = tf.squeeze(sampled_indices,axis=-
1).numpy()
        for x in sampled_indices:
            txt.append(idx_to_char[x])
        text=tf.strings.join(txt,separator=' ',name=None)
        #print ('----\n %s \n----' % (text.numpy()))
        smooth_loss = smooth_loss * 0.999 + loss * 0.001
        #print(smooth_loss.numpy())
        txt=[]
    print ('Epoch {} Loss {:.4f}'.format(i, loss))
    print ('Time taken for iteration {} is {}
sec\n'.format(i,time.time() - start))
    iterations.append(i)
    lossx.append(loss)
    plt.figure(figsize=(12,9))
    plt.plot(iterations,lossx,label = "Loss Tracker")
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    #plt.xlim(-3, 3)
    plt.xlim(1,400)
    plt.legend(loc='upper right')
    plt.show()
    if i % 5 == 0:
        model.save_weights(checkpoint_prefix.format(i=i))
    model.save_weights(checkpoint_prefix.format(i=i))
    i=i+1
```

APPENDIX 3: RNN Algorithm Code

```
import tensorflow as tf
import numpy as np
import time
import os
import matplotlib.pyplot as plt
BATCH_SIZE=64
embedding_size=256
sequence_length=50
BUFFER_SIZE=1000
neuronsa=800
neuronsb=224
data=open("SampleData.txt",encoding="utf8").read()
#data=data.split()
print(len(data))
vocab_size=len(list(set(data)))
words=list(set(data))
char_to_idx={ch:ix for ix,ch in enumerate(words)}
idx_to_char={ix:ch for ix,ch in enumerate(words)}
inputs=[char_to_idx[ch] for ch in data[:-1]]
targets=[char_to_idx[ch]for ch in data[1:]]
inputs_as_tensors=tf.data.Dataset.from_tensor_slices(inputs
)
targets_as_tensors=tf.data.Dataset.from_tensor_slices(targe
ts)
training_data=tf.data.Dataset.zip((inputs_as_tensors,target
s_as_tensors))
training_data=training_data.shuffle(BUFFER_SIZE).batch(sequ
ence_length,drop_remainder=True)
training_dataset=training_data.batch(BATCH_SIZE,drop_remain
der=True)
#print(training_data_final)
def
build_model(vocab_size,embedding_size,neuronsa,neuronsb,bat
ch_size):

model=tf.keras.Sequential([tf.keras.layers.Embedding(vocab_
size,embedding_size,

batch_input_shape=[batch_size,None]),

tf.keras.layers.RNN(neuronsa,return_sequences=True,stateful
=True,recurrent_initializer="glorot_uniform"),
tf.keras.layers.RNN(neuronsb,return_sequences=True,stateful
=True,recurrent_initializer="glorot_uniform"),

tf.keras.layers.Dense(vocab_size)
                              ])
    return model
def training_step(inputs,targets,optimizer):
    with tf.GradientTape() as tape:
        predictions=model(inputs)

loss=tf.reduce_mean(tf.keras.losses.sparse_categorical_cros
sentropy(targets,predictions,from_logits=True))
    grads=tape.gradient(loss,model.trainable_variables)

optimizer.apply_gradients(zip(grads,model.trainable_variabl
es))
    return loss,predictions
smooth_loss =-np.log(1.0/vocab_size)*sequence_length
txt=[]
model=build_model(vocab_size,embedding_size,neuronsa,neuron
sb,BATCH_SIZE)
i=0
lossx=[]
iterations=[]
checkpoint_dir = 'RNN_model_new'
checkpoint_prefix = os.path.join(checkpoint_dir,
"ckpt_{i}")
checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(file
path=checkpoint_prefix,save_weights_only=True)
while(1):
    start = time.time()
    hidden = model.reset_states()
    for x,y in training_dataset:

loss,predictions=training_step(x,y,tf.keras.optimizers.Adam
())
        sampled_indices =
tf.random.categorical(predictions[0], num_samples=1)
        sampled_indices = tf.squeeze(sampled_indices,axis=-
1).numpy()
        for x in sampled_indices:
            txt.append(idx_to_char[x])
        text=tf.strings.join(txt,separator=' ',name=None)
        #print ('----\n %s \n----' % (text.numpy()))
```

```
            smooth_loss = smooth_loss * 0.999 + loss * 0.001
            #print(smooth_loss.numpy())
            txt=[]
        print ('Epoch {} Loss {:.4f}'.format(i, loss))
        print ('Time taken for iteration {} is {}
    sec\n'.format(i,time.time() - start))
        iterations.append(i)
        lossx.append(loss)
        plt.figure(figsize=(12,9))
        plt.plot(iterations,lossx,label = "Loss Tracker")
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        #plt.xlim(-3, 3)
        plt.xlim(1,400)
        plt.legend(loc='upper right')
        plt.show()
        if i % 5 == 0:
            model.save_weights(checkpoint_prefix.format(i=i))
        model.save_weights(checkpoint_prefix.format(i=i))
        i=i+1
```