AI GENERATED TEXT DETECTION USING DEEP LEARNING

A FINAL YEAR PROJECT REPORT

Submitted by

ADITHYA MANIKANDAN J (311520104002)

K ANIRUDH (311520104005)

ARJUN KRISHNAN R (311520104008)

in partial fulfillment for the award of

the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE,

KODAMBAKKAM, CHENNAI – 600 024

ANNA UNIVERSITY: CHENNAI 600025

MAY 2024

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"AI GENERATED TEXT DETECTION USING DEEP LEARNING "** is the bonafide work of "**ADITHYA MANIKANDAN J (311520104002) , K ANIRUDH (311520104005) AND ARJUN KRISHNAN R (311520104008)"** who carried out the project work under my supervision.

SIGNATURE
Dr S Aarthi, M.E, PhD
HEAD OF THE DEPARTMENT
Computer Science and Engineering,
Meenakshi Sundararajan Engineering College,
No.363, Arcot Road, Kodambakkam,
Chennai - 600 024.

SIGNATURE
Mrs. C Jerin Mahibha, M.E, PhD
INTERNAL GUIDE
ASSOCIATE PROFESSOR
Computer Science and Engineering,
Meenakshi Sundararajan Engineering College,
No.363, Arcot Road, Kodambakkam,
Chennai - 600 024.

Submitted for the practical examination held on _____ at Meenakshi Sundararajan Engineering College, Kodambakkam, Chennai- 600 024

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we express our sincere gratitude to our Respected Correspondent **Dr K. S. Lakshmi**, our beloved Secretary **Mr. N**. **Sreekanth**, Principal **Dr S. V. Saravanan** and Dean Academics **Dr K. Umarani** for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks go to **Dr S. Aarthi**, **M.E., PhD,** Associate Professor and Head of the Department, Department of Computer Science and Engineering, for her profound inspiration, kind cooperation and guidance.

We're grateful to **Mrs. C. Jerin Mahibha, M.E., PhD,** Internal Guide, Associate Professor, Department of Computer Science and Engineering, **Dr. M. K.Sandhya, M.E., PhD**, Professor, project coordinator for their invaluable support in completing our project. We are extremely thankful and indebted for sharing expertise, and sincere and valuable guidance and encouragement extended to us.

We would also like to thank all teaching and non-teaching staff members of our department for sharing their knowledge and constant support during the course of the project.

We are grateful to our external guide **Mrs. Deepa B**, HR Operations and Program Manager at Settyl Tech India Private Limited, Chennai for giving us the opportunity to work on this project at their company. We also thank her for providing the necessary tools and guidance during the development of the project.

Above all, we extend our thanks to God Almighty without whose grace and blessings it wouldn't have been possible.

ABSTRACT

In recent years, the application of transformation models in artificial intelligence (AI) has emerged as a potent paradigm for various tasks, including text detection. This system provides a comprehensive exploration of the utilization of transformation models in AI-based text detection methodologies. Beginning with an exposition on the intricacies and challenges inherent in text detection, such as variable orientations, scales, and perspectives, the system lays the groundwork for understanding the efficacy of transformation models in addressing these complexities. It elucidates the fundamental principles behind transformation models, including their capacity for capturing geometric deformations and spatial relationships within textual content. The paper subsequently delves into an in-depth analysis of diverse transformation models employed in AI text detection, ranging from classic techniques like affine transformations to more sophisticated approaches such as spatial transformers and geometric matching networks. Each model is meticulously scrutinized in terms of its theoretical underpinnings, architectural design, and practical implications for text detection tasks. Furthermore, the system investigates the pivotal role of dataset preprocessing, augmentation strategies, and training methodologies tailored specifically for leveraging transformation models in text detection. It's critical to respect gifted writers and independent content producers by using AI-generated text detection. Using their classification measures, the performance of several models, including the CNN model and the DistilBert Classifier, is evaluated. Train both the models and evaluate the models by comparing accuracy, precision, recall, and F1 scores. DistilBert Classifier's fine tuning for a specific text classification demonstrated how the system changed. The CNN’s model accuracy is 99.62 and the DistilBert Classifier model’s accuracy is 99.76.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS AND EXPANSIONS

ABBREVIATIONS	EXPANSIONS
CNN	Conventional Neural Networks
ML	Machine Learning
NLU	Natural Language Understanding
AI	Artificial Intelligence
UML	Unified Modeling Language
BERT	Bidirectional Encoder Representations from Transformers
NLP	Natural Language Processing
GB	Gigabyte
TB	Terabyte

CHAPTER 1

INTRODUCTION

1.1 ABOUT THE PROJECT

The proliferation of artificial intelligence (AI) technologies has led to an increase in the generation of AI-generated text, posing significant challenges in distinguishing between genuine human-authored content and machine-generated content. Detecting AI-generated text is crucial in various contexts, including combating misinformation, ensuring content authenticity, and upholding the integrity of online platforms. In response to this challenge, researchers have developed various machine learning models and natural language processing (NLP) techniques to differentiate between AI-generated and human-generated text. In this study, the aim is to investigate and compare the effectiveness of two prominent models, a Convolutional Neural Network (CNN) and a DistilBERT model, for the task of detecting AI-generated text. The CNN model, known for its effectiveness in image classification tasks, is adapted to process text data, while the DistilBERT model leverages transformer-based architecture pre-trained on large text corpora to extract contextual information from input text. To evaluate the performance of these models, a diverse dataset comprising essays, instructional texts, and other textual content, annotated with labels indicating whether the text was generated by AI or by humans, is utilized. By training and testing both models on this dataset, the goal is to assess their ability to accurately classify text into AI-generated and human-generated categories. The significance of this research lies in its potential to contribute to the development of robust AI-generated text detection systems, thereby addressing concerns related to misinformation and ensuring the authenticity of online content. By comparing the performance of different models, insights into the strengths and limitations of each approach are sought, providing guidance on the most effective methods for AI-generated text detection.

1.2 DOMAIN OVERVIEW

The domain of AI-generated text detection encompasses a broad range of interdisciplinary fields, including natural language processing (NLP), machine learning, and computational linguistics. With the rapid advancement of AI technologies, particularly in language generation models such as GPT (Generative Pre-trained Transformer) and BERT (Bidirectional Encoder Representations from Transformers), the generation of AI-authored text has become increasingly prevalent.One of the primary challenges in this domain is distinguishing between AI-generated text and human-generated text, a task that has significant implications for combating misinformation, preserving content authenticity, and upholding ethical standards in online communication. AI-generated text detection systems play a crucial role in identifying and mitigating the spread of fake news, deceptive content, and propaganda across various digital platforms.Researchers and practitioners in this domain leverage a variety of approaches to tackle the problem of AI-generated text detection. These include traditional machine learning algorithms, deep learning models, and ensemble techniques that combine multiple methods to improve classification accuracy and robustness. Additionally, techniques such as feature engineering, semantic analysis, and anomaly detection are employed to extract meaningful patterns and characteristics from textual data. Furthermore, the domain of AI-generated text detection intersects with broader discussions around AI ethics, responsible AI deployment, and algorithmic transparency. As AI models become increasingly sophisticated and capable of generating highly convincing text, there is a growing need to develop ethical guidelines, regulatory frameworks, and accountability mechanisms to ensure the responsible use of AI-generated content. T. By advancing our understanding of AI-generated text detection techniques and addressing emerging challenges, researchers and practitioners strive to promote trust, integrity, and authenticity in digital communication platforms.

1.3 EXISTING SYSTEM

In existing systems for AI-generated text detection, researchers have employed various machine learning algorithms, including logistic regression, support vector machines (SVM), decision trees, k-nearest neighbors (KNN), and random forests, to classify text into AI-generated or human-generated categories. These algorithms are applied to features extracted from the text data, such as word frequencies, n-grams, syntactic patterns, and semantic representations. The choice of algorithms often depends on factors such as the nature of the data, the complexity of the classification task, and computational constraints. Logistic regression, for example, is a simple yet effective algorithm for binary classification tasks, while SVMs are known for their ability to handle high-dimensional data and non-linear decision boundaries. Decision trees and random forests offer interpretability and robustness against overfitting, while KNN is a non-parametric method that relies on similarity measures between data points. In previous studies, researchers have reported achieving maximum F1 scores of 0.76 using these algorithms for AI-generated text detection tasks. The F1 score, which combines precision and recall, is a common evaluation metric for binary classification tasks and provides a balanced measure of a model's performance.While these algorithms have demonstrated promising results, they may have limitations in capturing complex linguistic patterns and contextual information present in AI-generated text. Additionally, their performance may vary depending on the characteristics of the dataset and the preprocessing techniques applied.By incorporating state-of-the-art deep learning techniques and exploring ensemble methods that combine multiple algorithms, researchers aim to further improve the accuracy and robustness of AI-generated text detection systems.

1.4 PROBLEM STATEMENT

The proliferation of AI-generated text presents a significant challenge in online content moderation and misinformation detection. As AI technologies continue to advance, distinguishing between AI-generated and human-generated text has become increasingly difficult, leading to concerns regarding the spread of fake news, deceptive content, and propaganda.Existing text classification algorithms, such as logistic regression, support vector machines, decision trees, k-nearest neighbors, and random forests, have shown promising results in AI-generated text detection tasks, achieving maximum F1 scores of 0.76. However, these algorithms may struggle to capture the complex linguistic patterns and semantic nuances present in AI-generated text.Furthermore, the emergence of transformer-based models, such as BERT, GPT, and DistilBERT, has revolutionized natural language processing by enabling deep learning architectures to understand and generate human-like text. These models have demonstrated remarkable performance in various NLP tasks, raising the need to evaluate their effectiveness in AI-generated text detection. Therefore, the problem statement of this study is to investigate and compare the efficacy of traditional machine learning algorithms and transformer-based deep learning models, specifically a Convolutional Neural Network (CNN) and a DistilBERT model, in detecting AI-generated text. The objective is to determine which approach yields the highest classification accuracy and robustness in differentiating between AI-generated and human-generated text, thereby contributing to the development of more reliable and effective AI-generated text detection systems.

1.5 CHAPTER OVERVIEW

The project report is organized with various chapters that denote the various functionalities and aspects of the system being developed.

Chapter 1 gives a general description about the project. It represents the basic idea of the project and introduces the topics of the existing system and proposed system.

Chapter 2 deals with the related works of the project. A literature review for each related work is explained in detail.

Chapter 3 presents the system architecture and requirements. It specifies the hardware and software components that are required. It also lists the technologies used in the implementation of the project.

Chapter 4 explains the system design with the use of UML diagrams and the data flow diagrams.

Chapter 5 contributes a detailed description of different modules that are there in the design and how they are implemented.

Chapter 6 gives a detailed description of the different test cases that were performed on the system.

Chapter 7 provides the conclusion. It also elucidates how the project can be further enhanced.

CHAPTER 2

LITERATURE SURVEY

2.1 "DistilBERT: A distilled version of BERT for faster inference" by Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf.

The paper addresses the computational challenges posed by BERT's large size, which hinders its deployment in resource-constrained environments or applications requiring real-time processing. To overcome this limitation, the authors employ knowledge distillation to train DistilBERT, distilling the knowledge from BERT into a smaller model. Despite its reduced parameters and computational footprint, DistilBERT maintains competitive performance across various natural language processing tasks, making it suitable for applications where speed and efficiency are crucial.

2.2 "Understanding and Detecting AI-Generated Text" by Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith.

The paper begins by discussing the rapid advancements in natural language generation (NLG) models and the potential risks associated with their misuse, such as spreading misinformation or generating harmful content. It then explores various characteristics of AI-generated text, including fluency, coherence, and factual accuracy, highlighting the difficulty in distinguishing it from human-generated text. The authors propose a framework for detecting AI-generated text, incorporating features such as linguistic patterns, stylistic inconsistencies, and domain-specific knowledge

2.3 "Detecting AI-Generated Text with BERT" by Max Glockner, Leo Schwinn, and Iryna Gurevych.

The paper addresses the growing concern regarding the proliferation of AI-generated content, which can be used to spread misinformation or manipulate

public opinion. The authors propose a methodology for detecting AI-generated text by leveraging BERT's contextual embeddings and fine-tuning techniques. They experiment with different datasets and evaluation metrics to assess the performance of their approach. The results demonstrate the effectiveness of BERT in distinguishing between human-written and AI-generated text, highlighting its potential utility in combating the spread of disinformation.

2.4 "BERTweet: A Pre-trained Language Model for English Tweets" by Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen.

The paper addresses the unique characteristics of tweets, such as informal language, misspellings, and use of emojis and hashtags, which pose challenges for traditional language models trained on formal text. BERTweet is trained on a large dataset of English tweets and incorporates modifications to the BERT architecture to better handle these characteristics. The authors evaluate BERTweet on various downstream tasks and demonstrate its effectiveness in capturing tweet-specific linguistic features and outperforming generic language models.

2.5 "A survey on machine learning approaches for spam detection in online social networks" by Dalal Bataineh, Husam Al Jawaheri, and Omar Al-Jarrah.

The paper begins by highlighting the increasing prevalence of spam in OSNs and its detrimental effects on user experience and platform credibility. It then systematically categorizes and reviews existing machine learning approaches for spam detection, including supervised, unsupervised, semi-supervised, and deep learning methods. The authors discuss the strengths and limitations of each approach, as well as the features commonly used for spam detection, such as content-based features, user-based features, and network-based features.

2.6 "Unsupervised BERT-based Embedding for Authorship Attribution" by Zhouhan Lin, Minlie Huang, and Juanzi Li.

The paper addresses the task of determining the author of a given text based on their unique writing style, which has applications in forensic linguistics, plagiarism detection, and literary analysis. Unlike traditional approaches that rely on handcrafted features or supervised learning, the proposed method leverages unsupervised pre-trained BERT embeddings to represent text passages. By fine-tuning BERT on a large corpus of unlabeled data, the model learns to encode textual features that capture the nuances of an author's writing style.

2.7 "BERTweet: A BERT-based Language Model for Twitter" by Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen.

The paper addresses the unique characteristics of tweets, such as informal language, misspellings, hashtags, and emojis, which pose challenges for traditional language models trained on formal text. BERTweet is trained on a large dataset of English tweets and incorporates modifications to the BERT architecture to better handle these characteristics. The authors evaluate BERTweet on various downstream tasks, such as sentiment analysis and named entity recognition, and demonstrate its effectiveness in capturing tweet-specific linguistic features.

2.8 "Adapting BERT for Targeted Aspect-based Sentiment Analysis" by Canwen Xu, Junyi Li, and Xiaojun Quan.

The paper addresses the challenge of effectively leveraging BERT's contextualized representations for TABSA by proposing a novel adaptation approach. The authors introduce a hierarchical architecture that combines BERT's representation with target-specific representations to capture both global and local contextual information. This architecture consists of a global module

and a local module. The global module utilizes BERT to encode the entire sentence, while the local module focuses on the target aspect and its surrounding context. These representations are then combined to predict the sentiment polarity towards the target aspect.

2.9 "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context" by Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov.

The paper addresses the challenge of modeling long-range dependencies in sequences, which is crucial for tasks such as language modeling. Traditional transformers have a fixed-length context due to memory constraints, which limits their ability to capture long-range dependencies effectively. Transformer-XL introduces two key innovations to address this limitation: the recurrence mechanism and the segment-level recurrence mechanism.

2.10 "BERT for Coreference Resolution: Baselines and Analysis" by Arun Kumar, Steven C.H. Hoi, Peilin Zhao, and Khanh Nguyen.

The paper presents several baseline models that utilize BERT embeddings for coreference resolution and conducts a detailed analysis of their performance. The authors explore different strategies for incorporating BERT embeddings into existing coreference resolution architectures, such as fine-tuning BERT on coreference resolution tasks and using BERT embeddings as features in traditional coreference resolution models.

CHAPTER 3

SYSTEM ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project leverages Flask, a Python web framework, to develop a user-friendly interface for text classification, distinguishing between AI-generated and human-generated text. The architecture begins with user input, facilitated through a web form or API endpoint. The incoming text undergoes preprocessing steps to standardize and clean the input. Subsequently, the preprocessed text is fed into two distinct models: a Convolutional Neural Network (CNN) and a DistilBERT classifier. The CNN model specializes in text classification tasks, while the DistilBERT classifier offers state-of-the-art natural language processing capabilities. Both models generate predictions regarding the origin of the input text, whether AI-generated or human-generated. These predictions are then relayed back to the user interface through the Flask application, providing clear feedback on the nature of the input text. Deployment on a web server ensures accessibility, with platforms like Heroku, AWS, or Azure offering robust hosting solutions. Continuous monitoring and maintenance of the deployed application ensure its reliability and performance over time. This architecture offers a seamless and efficient solution for discerning between AI-generated and human-generated text through a user-friendly web interface. Throughout the training phase, the model is optimized using a validation set to ensure robust performance. Evaluation metrics such as accuracy, precision, recall, and F1-score are used to assess the model's effectiveness. Once trained, the model is deployed into a production environment, where its performance is monitored and maintained. Regular updates and retraining ensure the model remains effective over time, while also considering ethical implications and advancements in AI research.

3.2 SYSTEM ARCHITECTURE

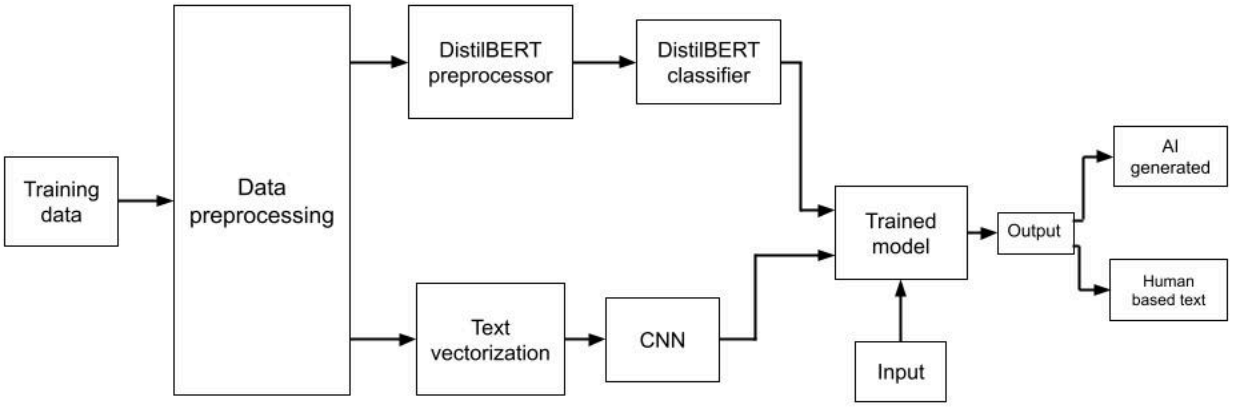

Figure 3.1 System Architecture

The **Convolutional Neural Network (CNN)** architecture comprises Convolutional 2D, Max Pooling, and Fully Convolutional Layers of various dimensions. The CNN model has an input layer which accepts strings of any length,then it goes through text vectorization layer. It is here where it undergoes tokenization and text vectorization, max tokens parameter is set to maximum vocabulary size and only most frequent words are considered. Then the global average pooling layer computes the average of the embeddings across the time dimension. Then the **DistilBERT classifier** is built on top of the DistilBERT preprocessor. It consists of a pre-trained DistilBERT model followed by a classification layer. The classifier is compiled with the specified loss function, optimizer, and any additional settings. In this case, the loss function is Sparse Categorical Cross Entropy, which is suitable for multi-class classification tasks like binary classification.The preprocessed text undergoes feature extraction using a pre-trained DistilBERT model, which generates contextualized embeddings capturing semantic information. These embeddings serve as input to a text detection model, typically a neural network, trained to distinguish between genuine and AI-generated text. Throughout training and evaluation, the model's performance is fine-tuned and validated against established metrics.

3.3 HARDWARE REQUIREMENTS

The Hardware Requirements are crucial for efficient training and inference processes. While CPUs can suffice for small to medium-sized datasets, GPUs, particularly NVIDIA GeForce GTX or RTX series with CUDA support, offer significant acceleration, especially for larger datasets and more complex models. Google's Tensor Processing Units (TPUs) are also viable options, known for their high throughput and cost-effectiveness, particularly for large-scale training tasks. Adequate system memory (RAM) is essential to handle the memory-intensive operations of DistilBERT, especially for larger models and datasets.Choosing the right hardware configuration ensures optimal performance and efficient resource utilization in AI-generated text detection tasks using DistilBERT. The following subsections discuss the various aspects of hardware requirements:

S.No	REQUIREMENTS	RECOMMENDED	MINIMUM REQUIREMENTS
1	Processor	Intel Core i3 and Above	Intel Core i3
2	Speed	1.8+ GHz	1.8 GHz
3	RAM	8 GB	8 GB
4	System Type	32Bit/64Bit	32Bit/64Bit
5	Hard Disk	256 GB	128 GB
6	Monitor	SVGA or higher resolution	SVGA
7	Key Board	Standard Windows Keyboard	Standard Windows Keyboard

Table 3.1 Hardware Requirements

3.4 SOFTWARE REQUIREMENTS

Software requirements define the functional and non-functional specifications that a software system or application must fulfill. They describe what the software should do, how it should behave, and any constraints or performance expectations. These requirements specify the desired behavior and functionality of the software. They outline the measures and safeguards that the software must have to protect data, ensure confidentiality, integrity, and availability, and prevent unauthorized access or misuse.

S.No	REQUIREMENTS	RECOMMENDED	MINIMUM REQUIREMENTS
1	Operating System	Windows 10	Windows 7
2	Programming language	Python 3.6 and Above	Python 3.6
3	Platform	Google Collab	Jupiter

Table 3.2 Software Requirements

3.4.1 TENSOR FLOW

TensorFlow offers both high-level APIs like Keras, which simplify the process of building and training neural networks, as well as lower-level APIs that provide more flexibility and control over model architecture and training process.With TensorFlow, you can build and train complex neural network architectures, deploy models to various platforms including mobile and web, and scale up training on distributed computing infrastructure.

3.4.2 PYTHON IDLE

IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python. The Python installer for Windows contains the IDLE module by default. IDLE can be used to execute a

single statement just like Python Shell and also to create, modify, and execute Python scripts. IDLE provides a fully-featured text editor to create Python script that includes features like syntax highlighting, auto completion, and smart indent. It also has a debugger with stepping and breakpoints features. During the preprocessing stage, Python IDLE allows developers to write and execute scripts for tasks like text normalization, tokenization, and data cleaning. These scripts can be tailored to the specific requirements of the text data and easily modified as needed. For model development, Python IDLE facilitates the creation and testing of text detection models. Developers can write Python code to define neural network architectures, integrate DistilBERT for feature extraction, and implement training and evaluation procedures. Additionally, Python IDLE provides access to libraries such as TensorFlow or PyTorch, which are commonly used for building and training deep learning models.

3.4.3 PYTHON

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Python's extensive collection of libraries, such as Pandas, NumPy, and NLTK, provides robust support for data preprocessing tasks. With Python, developers can efficiently clean, tokenize, and transform text data, preparing it for further analysis.

CHAPTER 4
SYSTEM MODELING

4.1 UNIFIED MODELING LANGUAGE(UML)

Unified Modelling Language is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

The primary goals in the design of the UML as follows:

- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language
- Encourage the growth of the OO tools market
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.

4.2 USE CASE DIAGRAM

The use case diagram is used to define the core elements and processes that make up a system. The key elements are termed as "actors" and the processes are called "use cases". The use case diagram shows which actors interact with each use case. This definition defines what a use case diagram is primarily made up of - actors and use cases.

In software and system engineering, a use case is a list of steps, typically defining interactions between a role (known in UML as an “actor”) and a system, to achieve a goal. The actor can be a human or an external system. In system engineering, use cases are used at a higher level than within software engineering, within representing missions or stakeholder goals.

The purposes of use case diagrams can be as follows

1. Used to gather requirements of a system.
2. Used to get an outside view of a system
3. Identify external and internal factors influencing the die system.
4. Showing the interacting among the requirements are actors.

Use cases help in identifying the operations that can be performed by an actor. It gives a list of the various applications that can be utilized by the system. The actor can be a real time human or a system. It helps in identifying the various modules present in the system. A single use case diagram captures a particular functionality of a system. Hence to model the entire system, a number of use case diagrams are used.

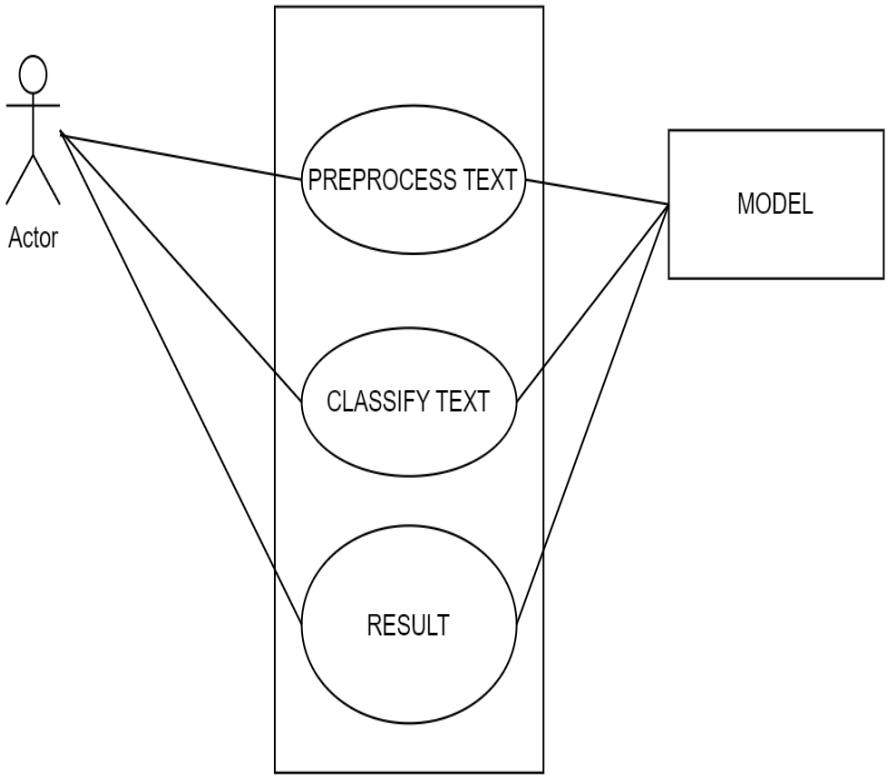

Figure 4.1: Use case diagram

4.3 CLASS DIAGRAM

Class diagram is a static diagram. It is the building block of every object-oriented system and helps in visualizing and describing the system. A class diagram depicts the structure of the system through its classes, their attributes, operations and relationships among the objects. A class is a blueprint that defines the variables and methods common to all objects of a certain kind. Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. The characteristics of class diagram are:

1. Each class is represented by a rectangle having a subdivision of three compartments - name, attributes and operations
2. There are three types of modifiers which are used to decide the visibility of attributes and operations: + is used for public visibility, a is used for protected visibility, - is used for private visibility.

In the diagram, classes are represented with boxes that contain three compartments. The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized. The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase. The bottom compartment contains the operations the class can execute. They are also left- aligned and the first letter is lowercase.

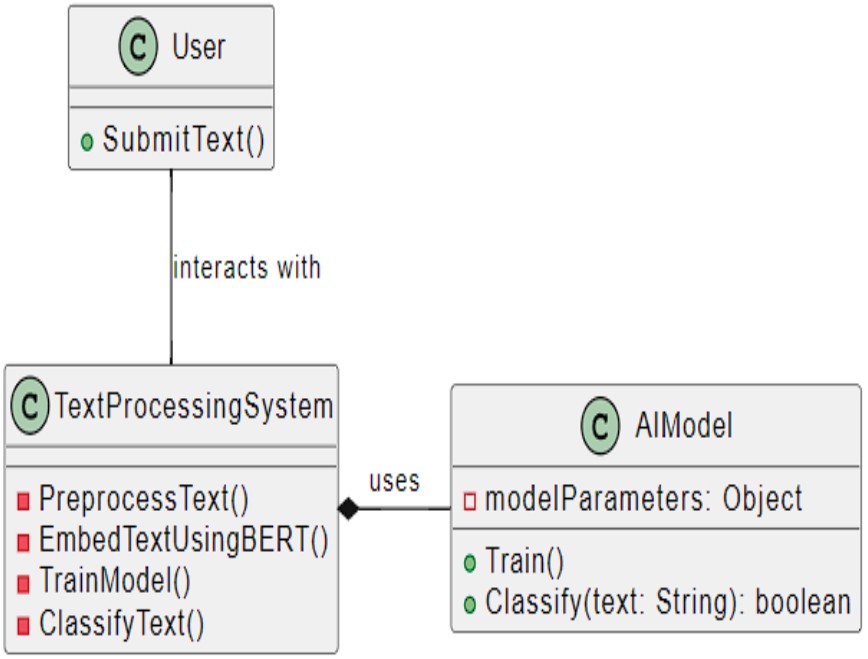

Figure 4.2: Class Diagram

4.4 SEQUENCE DIAGRAM

A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in which order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence.

Sequence diagrams are a popular dynamic modeling solution in UML because they specifically focus on lifelines, or the processes and objects that live simultaneously, and the messages exchanged between them to perform a

function before the lifeline ends.

It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. A sequence diagram shows different processes or objects that live simultaneously as parallel vertical lines (lifelines) and the messages exchanged between them and the order in which they occur as horizontal arrows.

The main purpose of the sequence diagram is

1. To capture the dynamic behavior of a system.
2. To describe the message flow in the system.
3. To describe the interaction among objects.

Sequence diagrams can be used

- To model the flow of control by time sequence.
- To model the flow of control by structural organizations.
- For reverse engineering.

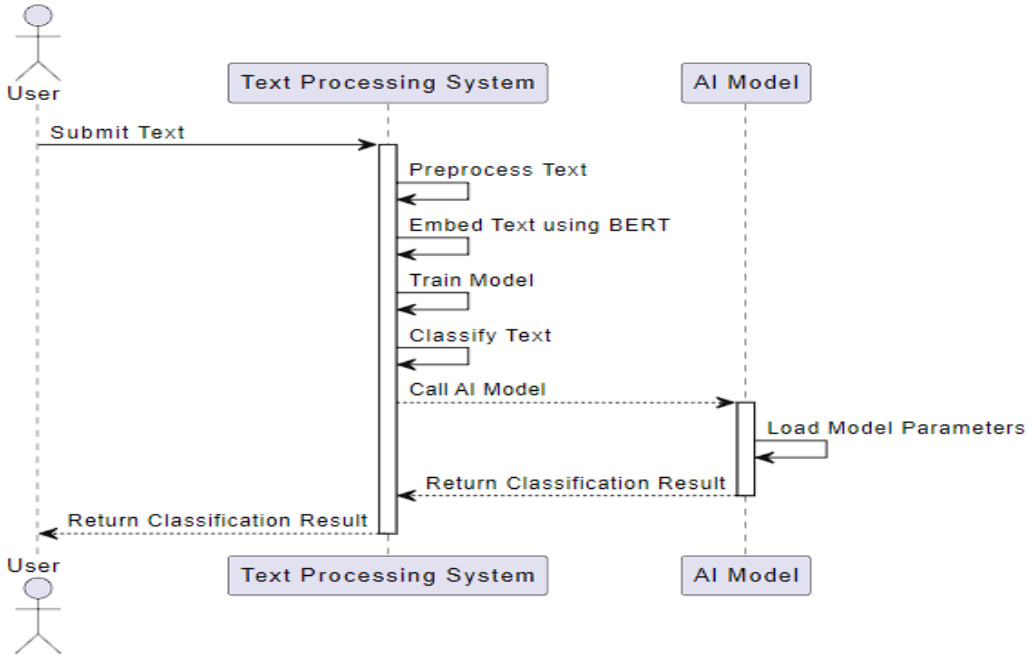

Figure 4.3: Sequence Diagram

4.5 COLLABORATION DIAGRAM

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among objects in the Unified Modelling Language (UML).

Collaboration diagrams convey the same information as sequence diagrams, but focus on object roles instead of the timings of messages. Collaboration diagrams represent a combination of information taken from class, sequence and use case diagrams describing both the static structure and dynamic behavior of a system. The collaboration diagram describes the messages or roles sent between objects.

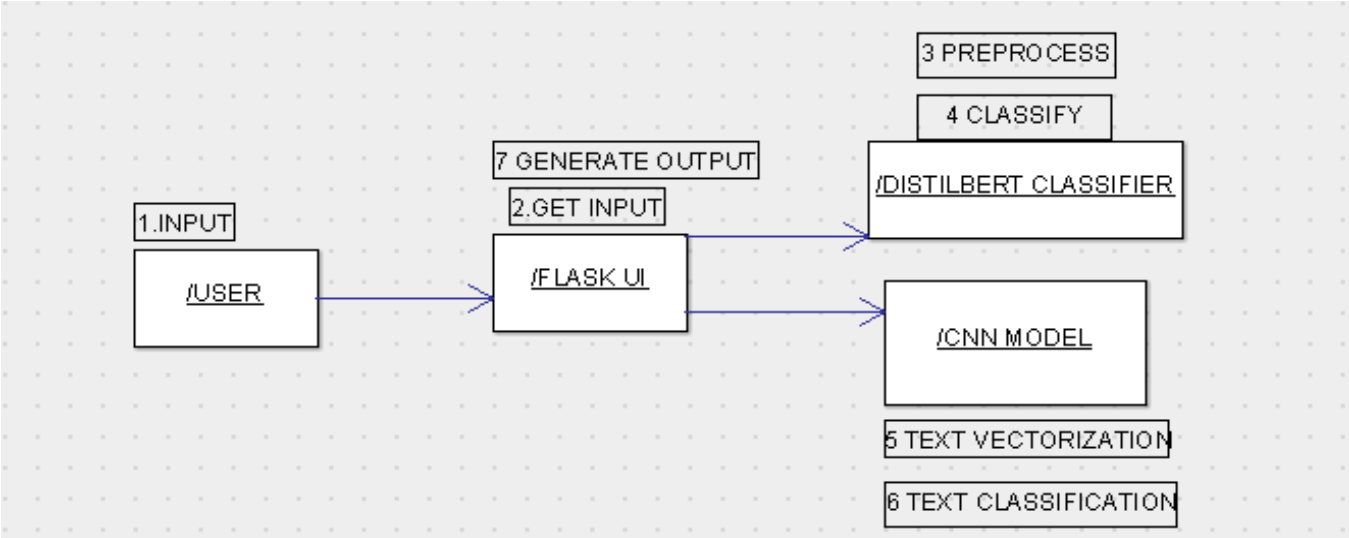

Figure 4.4: Collaboration Diagram

4.6 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams are intended to model both computational and organizational processes (i.e., work flows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.. Thus, flow can be sequential, branched, or concurrent.

Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc. Activity diagrams are constructed from a limited number of shapes, connected with arrows.

The most important shape types:

- rounded rectangles representations
- diamonds represent decisions
- bars represent the start (split) or end (join) of concurrent activities
- a black circle represents the start (initial node) of the workflow
- an encircled black circle represents the end (final node)

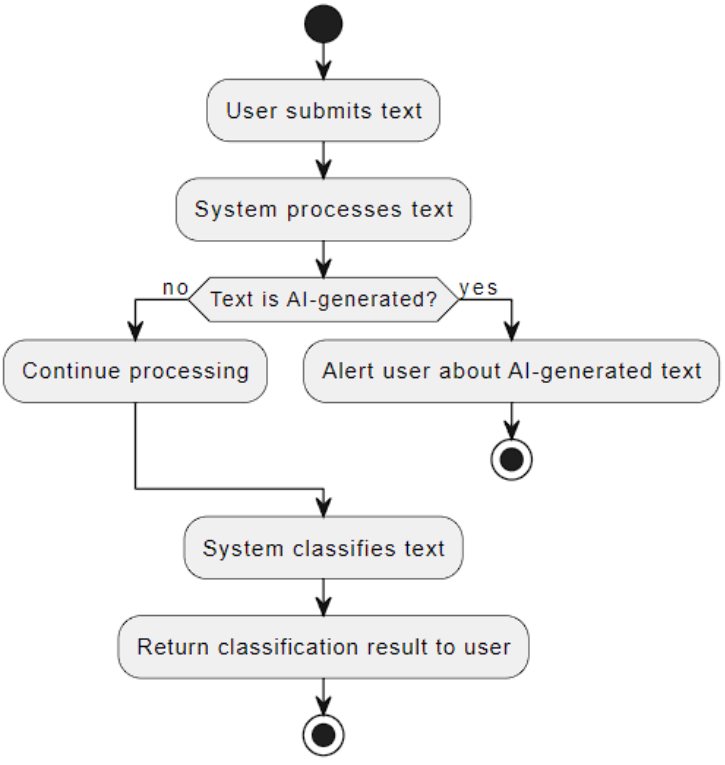

Figure 4.5: Activity Diagram

4.7 STATECHART DIAGRAM

A state chart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events. It describes different states of a component in a system. The states are specific to a component/object of a system. State chart diagrams are used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems.

State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of state chart diagrams is to model the lifetime of an object from creation to termination.

The main purposes of using state chart diagrams

- To model the dynamic aspect of a system.
- To model the lifetime of a reactive system.
- To describe different states of an object during its lifetime.
- An encircled black circle represents the end (final node).

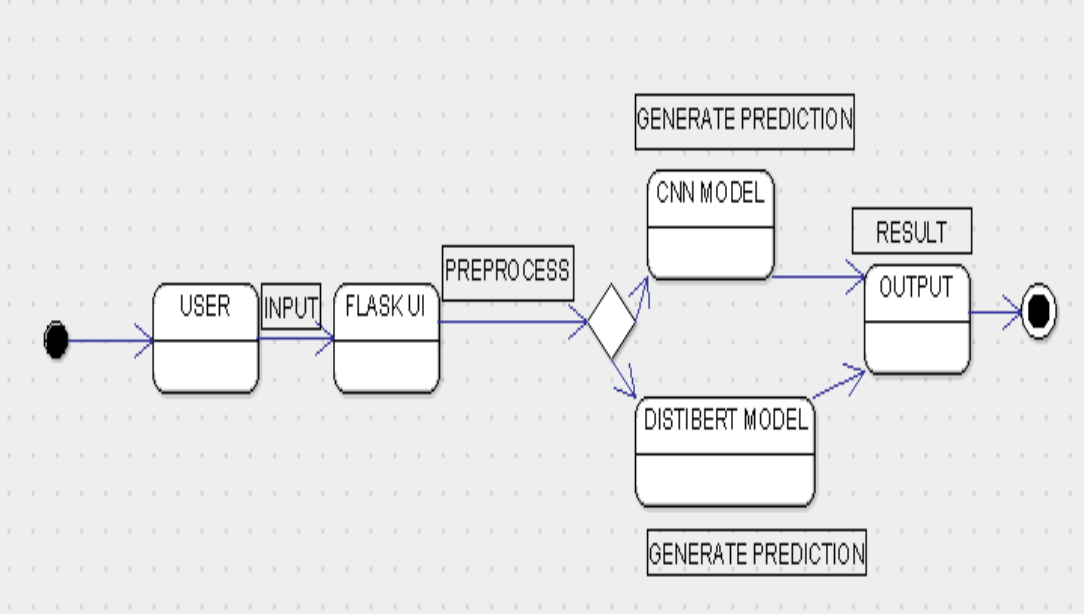

Figure 4.6: State Chart Diagram

4.8 COMPONENT DIAGRAM

In the Unified Modelling Language, a component diagram depicts how components are wired together to form larger components or software systems. They are used to illustrate the structure of arbitrarily complex systems. Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems. Component diagram is a special kind of diagram in UML.

The purpose is also different from all other diagrams. It does not describe the functionality of the system but it describes the components asked to make those functionalities. Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

The purpose of the component diagram can be summarized as visualizing the components of a system.

- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

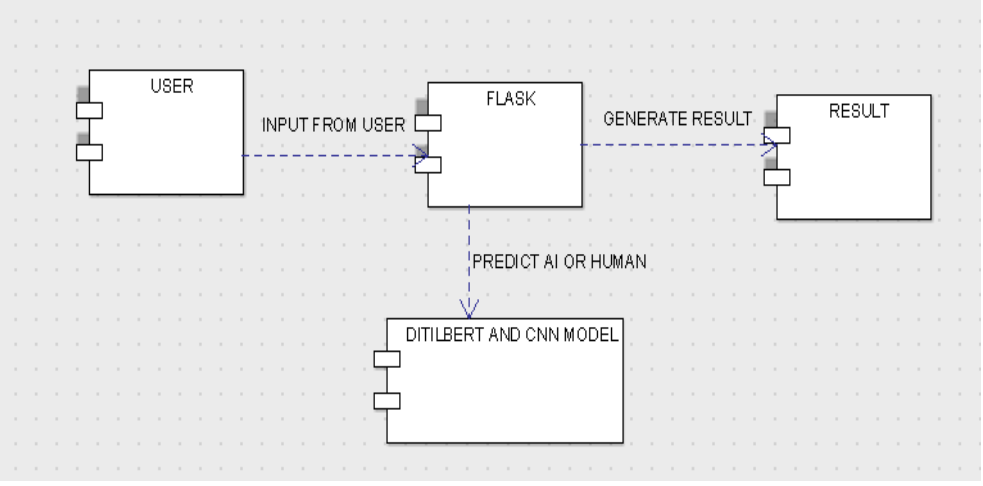

Figure 4.7: Component Diagram

4.9 PACKAGE DIAGRAM

Package diagram is a UML structure diagram which shows packages and dependencies between the packages. The package diagram shows the arrangement and organization of the model elements in a middle to large scale project.

The package diagram can show both the structure and dependencies between subsystems or modules. A package is rendered as a tabbed folder - a rectangle with a small tab attached to the left side of the top of the rectangle. If the members of the package are not shown inside the package rectangle, then the name of the package should be placed inside. The members of the package may be shown within the boundaries of the package. In this case, the name of the package should be placed on the tab. A diagram showing a package with content can show only a subset of the contained elements according to some criterion. Members of the package may be shown outside of the package by branching lines from the package to the members. The dotted arrows are dependencies. Packages can be built to represent either physical or logical relationships. The entire system is divided logically into three packages - User Interface (UI), Technical and Domain. The package UI deals with objects and operations that are user interface specific. The three sub-packages in UI pertain to information exchange between the user and the system. The package technically deals with storage of information in the datasets and the operations on the dataset and the model formed. The package Domain handles all important middleware operations that involve the core processing of the system.

In the following diagram, the CLI represents the Ul package. The UI package contains the user interface of the system, such as the warning messages. The UI package is responsible for displaying the lane markings on the dashboard and warning if they are about to drift out of their lane. The core package (Domain package) contains the core classes of the system, such as the camera, image processor and lane detector. The core classes are responsible for

detecting lane markings in images and tracking the position of the lanes. The process, upload, generate, extract will represent the different services of the system (Technical Services package).

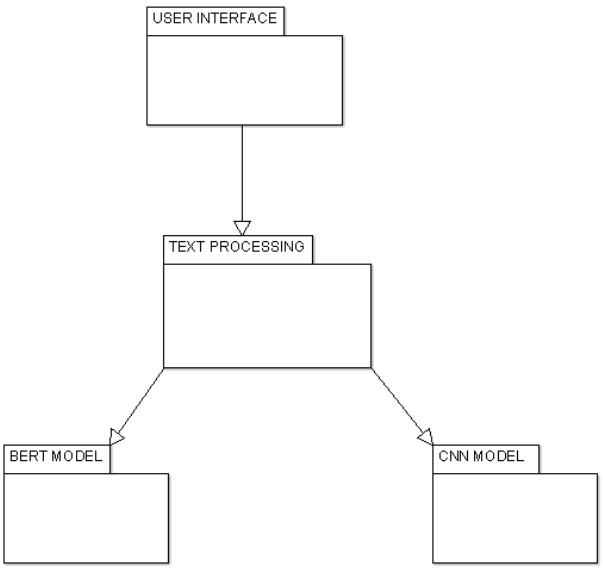

Figure 4.8: Package Diagram

4.10 DEPLOYMENT DIAGRAM

A deployment diagram in the Unified Modelling Language models the physical deployment of artifacts on nodes. To describe a website, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g., JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers. Deployment diagrams are used by system engineers.

The purposes of deployment diagrams can be as follows:

1. Visualize the hardware topology of a system.
2. Describe the hardware components used to deploy software components.
3. Describe the runtime processing nodes.

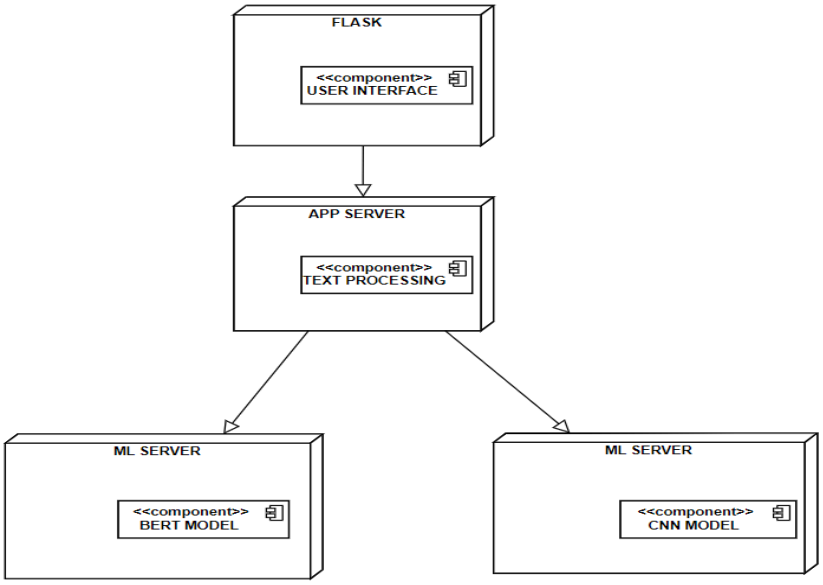

Figure 4.9: Deployment Diagram

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 PROPOSED SYSTEM

The proposed system for AI-generated text detection leverages a combination of DistilBERT and Convolutional Neural Networks (CNNs) to achieve accurate classification. Initially, a diverse dataset encompassing both genuine and AI-generated text samples is collected and preprocessed to ensure uniformity. DistilBERT, a pre-trained transformer-based model, is then employed to extract contextualized embeddings from the preprocessed text data. These embeddings serve as rich feature representations capturing semantic information crucial for accurate classification.Subsequently, a custom-designed CNN architecture is integrated into the system to further process the DistilBERT embeddings and perform the text detection task. The CNN architecture comprises convolutional layers, possibly with varying kernel sizes and depths, followed by pooling layers to capture distinctive patterns within the embeddings. This architecture is optimized through experimentation with different configurations, including activation functions and regularization techniques, to enhance its ability to discriminate between genuine and AI-generated text.During the training phase, the model is trained using the labeled dataset, with the objective of minimizing classification errors. Techniques such as dropout regularization and batch normalization are applied to prevent overfitting and improve generalization performance. The model's performance is evaluated using standard evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to ensure its effectiveness in detecting AI-generated text.Upon successful training and evaluation, the model is deployed into a production environment, where it can be accessed via APIs for real-time inference.

5.1.1 CONVOLUTIONAL NEURAL NETWORKS

The Convolutional Neural Network (CNN) model for text processing commences by accommodating strings of varying lengths at its input layer, where they undergo tokenization and text vectorization in the subsequent layer, with the maximum vocabulary size regulated by the max tokens parameter. This adaptive layer dynamically learns the vocabulary from the training data, adeptly mapping tokens to numerical vectors. Following this, an embedding layer uniformly initializes word embeddings, seamlessly associating each token with a dense vector representation. Subsequently, the global average pooling layer meticulously computes the mean of these embeddings across the time dimension, effectively condensing the input while judiciously retaining essential information. This holistic pipeline not only empowers the model to proficiently handle textual inputs but also facilitates the acquisition of significant word embeddings, thereby consolidating information across sequences. Such meticulous processing is paramount for tasks such as text classification and sentiment analysis, where understanding the underlying semantic structures is imperative for accurate predictions. The CNN architecture's adaptability to variable-length inputs ensures robustness in handling diverse text data, allowing it to accommodate a wide range of sentence lengths without the need for manual preprocessing. The uniform initialization facilitates the creation of a rich semantic space where words with similar meanings are represented closer together, aiding in capturing nuanced semantic relationships. Meanwhile, the global average pooling layer's reduction of input dimensionality helps mitigate the risk of overfitting by focusing on the most salient features of the text while discarding noise. Altogether, intricate architecture forms a robust framework for text analysis, enabling the CNN model to effectively learn and generalize from textual data across a spectrum of natural language processing tasks.

5.1.2 DISTILBERT MODEL

The DistilBERT Classifier is a specialized variant of the BERT model designed specifically for classification tasks, leveraging the power of transfer learning. Initially pre-trained on extensive datasets including Wikipedia and various other sources, it encodes vast linguistic knowledge into its parameters. This classifier is composed of two main components: the DistilBERT preprocessor and the classification layer. The preprocessor tokenizes input text, converting it into token IDs, while the DistilBERT model, which follows, learns hierarchical representations of the text. Atop the DistilBERT model sits the classification layer, tailored for the specific task at hand. During fine-tuning, the classifier adapts its parameters to the nuances of the target dataset through iterative training. In this instance, the loss function employed is Sparse Categorical Cross Entropy, apt for multi-class classification tasks, including binary classification. Finally, the model is evaluated to assess its performance, ensuring its readiness for deployment in real-world scenarios, where it can effectively discern and categorize text inputs. Fine-tuning the DistilBERT Classifier involves exposing the pre-trained model to the specifics of the target classification task, allowing it to adjust its parameters accordingly. This process enhances the model's ability to discern relevant patterns and features in the input data, thereby improving its classification accuracy. By leveraging the pre-existing linguistic knowledge encoded in the pre-trained DistilBERT model, the fine-tuning stage optimizes the model's performance for the specific domain or dataset of interest. Additionally, during compilation, the classifier is configured with the appropriate loss function, optimizer, and any additional hyperparameters, ensuring optimal training dynamics. Through this meticulous process, the DistilBERT Classifier emerges as a potent tool for a wide array of classification tasks, ranging from sentiment analysis to document categorization, exhibiting robustness and efficiency in handling diverse textual data.

5.2 MODULE DESCRIPTION

5.2.1 User Interface

The UI Flask module acts as the interface for user interaction, defining routes and endpoints for input submission. Through HTML templates and forms, users can input text data, which is then processed by backend modules. Integration with these modules enables seamless communication and data flow between the frontend and backend. Upon processing, results are displayed to users, ensuring a smooth and intuitive experience. Error handling mechanisms provide informative feedback to users, enhancing usability and reliability.

5.2.2 Text Preprocessing

It consists of a distilbert preprocessor that prepares the input. It tokenizes the input text and converts it into token ids. The DistilBERT classifier is built on top of the DistilBERT preprocessor. The CNN model has an input layer which accepts strings of any length,then it goes through a text vectorization layer. It is here where it undergoes tokenization and text vectorization, max tokens parameter is set to maximum vocabulary size and only most frequent words are considered. In text preprocessing for a Convolutional Neural Network (CNN) text classification task, the input text undergoes several crucial steps. Initially, the text is tokenized, breaking it down into individual words or tokens. Following tokenization, all tokens are typically converted to lowercase to ensure consistency.

5.2.3 Classifier Model

This module is the actual DistilBERT model fine-tuned for your binary classification task. It takes the preprocessed text input and generates predictions indicating whether the input text is humanized or AI-generated. The classifier is typically trained using a labeled dataset containing examples of both humanized and AI-generated text, with corresponding binary labels. The DistilBERT model itself is a variant of the BERT (Bidirectional Encoder Representations from

Transformers) model, which has been distilled for faster and more efficient training and inference. It comprises a stack of transformer layers that encode the input text into contextualized representations. The final hidden states or pooled output of the DistilBERT model are typically used for downstream classification tasks. On top of the DistilBERT model, a classification head is added to perform the specific task of binary classification. This head typically consists of one or more dense layers followed by a softmax or sigmoid activation function, depending on whether binary cross-entropy or multi-class cross-entropy loss is used.

5.2.4 CNN Model

The CNN model has an input layer that accepts strings of any length. Then to a text vectorization layer. Here it undergoes tokenization and text vectorization. Max tokens parameter is set to maximum vocabulary size and only those words are considered that are most frequent. This layer adapts to training data. It learns the vocabulary from the training data and maps tokens to vectors. Then the embedding layer is initialized uniformly. Then the global average pooling layer: it takes the average of the embeddings across the time dimension. This reduces the size of the input while retaining important information. Then a dense layer with one neuron and sigmoid activation function to map the output of the global pooling layer to one single output that represents the probability of the input class. Thus, it is useful for the binary classification problem. The CNN model has a convolutional layer, a pooling layer, a dense layer, and an output layer. During the training phase, the loss functions of both models are then calculated, and the weights are updated by using a particular optimization algorithm.

CHAPTER 6

SYSTEM TESTING

6.1 INTRODUCTION

Software testing is a process of evaluating a software in a comprehensive manner in order to verify whether it is running in the desired fashion and to identify any bugs present if any. This is done in order to keep the quality of the deliverable high.

In accordance with the standard of ANSI, the definition of Software Testing is A process of analyzing a software item to detect the differences between existing and required conditions (i.e, defects) and to evaluate the features of the software item.

6.2 TESTING APPROACHES

There are two main kinds of testing approaches under software testing which are:

i. White Box Testing
ii. Black Box Testing

6.2.1 WHITE BOX TESTING

White box testing involves analyzing the internal structures of the code and not merely the functionality. It is used to identify any errors present in the code structure. It is usually done for unit testing although it can be done for integration and system testing too. White box testing is used for debugging code within a subsystem, between subsystems and so on. In white box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. It is also called Glass Box, Clear Box and Structural Testing.

Advantages of White Box Testing:

- White Box Testing has simple and clear rules to let a tester know when the testing is done.
- White Box Testing techniques are easy to automate, this results in a developer having to hire fewer testers and smaller expenses.
- It shows bottlenecks which makes the optimization quite easy for the programmers.

6.2.2 BLACK BOX TESTING

As opposed to white box testing, black box testing is used to analyze the functionality of the software. It can be done in all levels of testing from unit testing to system testing. It is generally done for testing higher levels of code. It is also called Behavioural/Specification-Based/Input-Output Testing.

Advantages of Black Box Testing:

- Black box tests are always executed from a user's point of view since it would help in exposing discrepancies significantly.
- Black box testers also do not need to know any programming languages.

6.3 TESTING LEVELS

Tests are grouped together based on the level of detailing they contain. The purpose of levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level. In general, there are four levels of testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

6.3.1 UNIT TESTING

A unit is a smallest testable portion of a system or application which can be compiled, linked, loaded, and executed. This kind of testing helps to test each module separately.

6.3.2 INTEGRATION TESTING

Integration means combining. In this testing phase, different software modules are combined and tested as a group. Integrating testing checks the data flow from one module to other modules.

6.3.3 SYSTEM TESTING

System testing is performed on a complete, integrated system. It does checking of the system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing. System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional needs.

6.3.4 ACCEPTANCE TESTING

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

6.4 TESTING TYPES

There are two types of testing which are classified as such based on the manner in which the testing is done. They are:

- Manual Testing
- Automation Testing

6.4.1 MANUAL TESTING

Manual testing is the process of testing software by hand. This usually includes verifying all the features specified in requirements documents, but often also includes the testers trying the software with the perspective of their end users in mind. Manual test plans vary from fully scripted test cases, giving testers detailed steps and expected results to high-level guides that steer exploratory testing sessions.

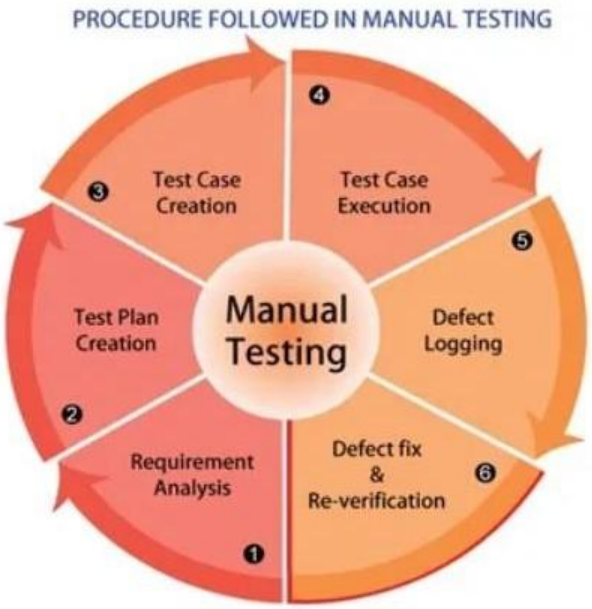

Figure 6.1: Manual Testing

6.4.2 AUTOMATION TESTING

Automation testing is the process of testing the software using an automation tool to find the defects. In this process, testers execute the test scripts and generate the test results automatically by using automation tools.

6.4 TESTING MODULES USING TESTPAD TOOLS

Test pad is a test plan tool that helps you find the bugs that matter. Less time messing around with spreadsheets or old-school test case management means more time actually testing. A simple test planning tool that reinvents test

case management as checklists that anyone can use. It takes a refreshingly simple approach that makes it really fast to write, run and maintain tests.

Test pad is for:

- engineers who want a pragmatic solution to product testing.
- testers in startups who know they can do better than spreadsheets.
- professional testers shackled by managing test cases one form at a time.

TEST RESULTS

Figure 6.2 Test result 1

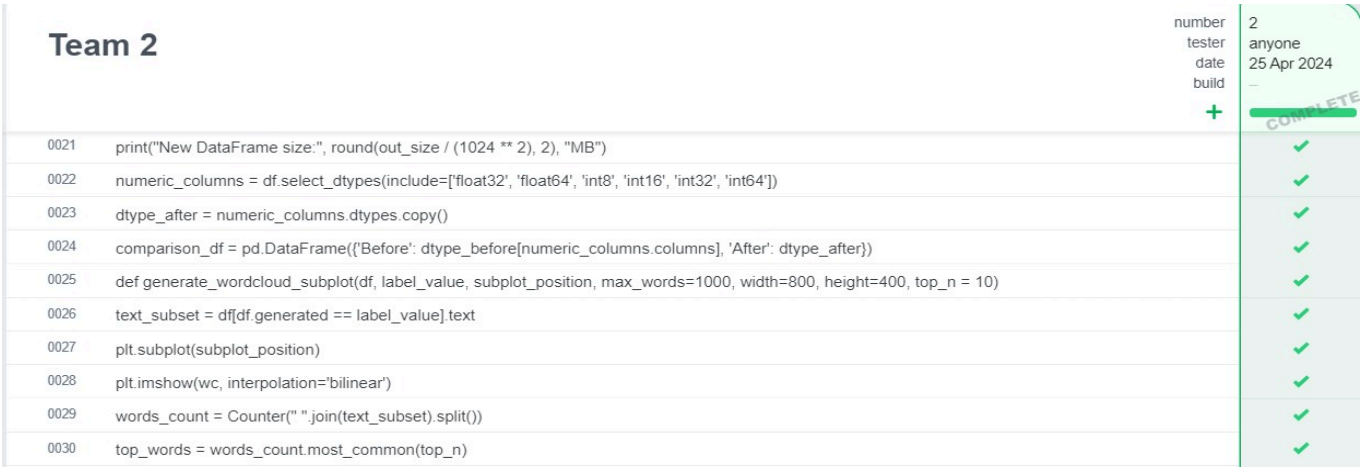

Figure 6.3 Test result 2

Figure 6.4 Test result 3

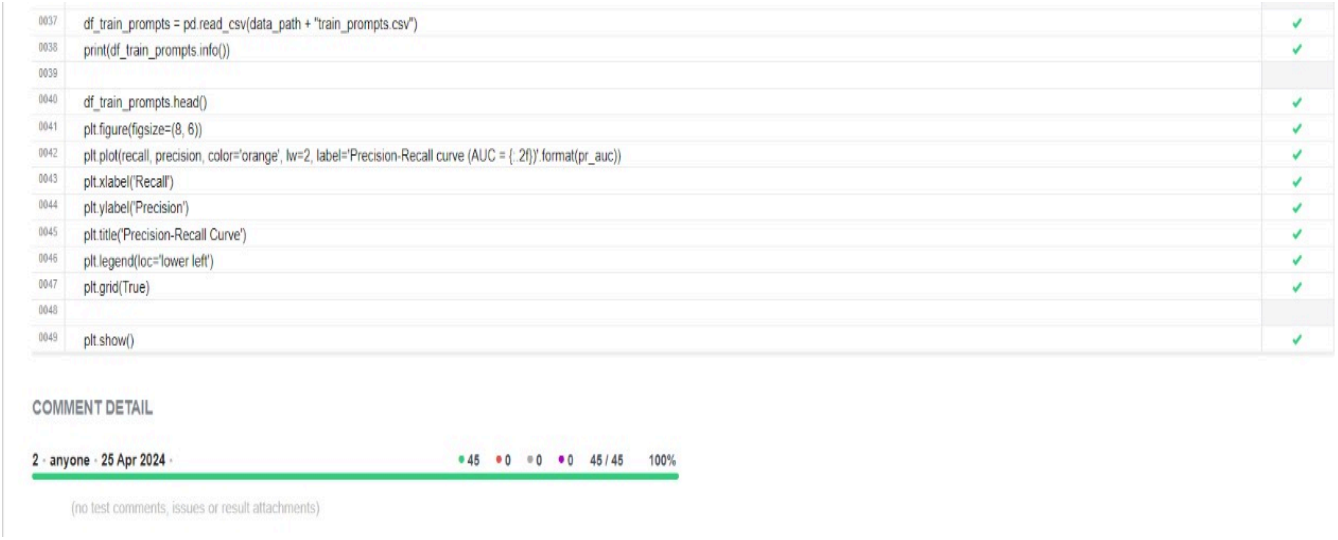

Figure 6.5 Test result 4

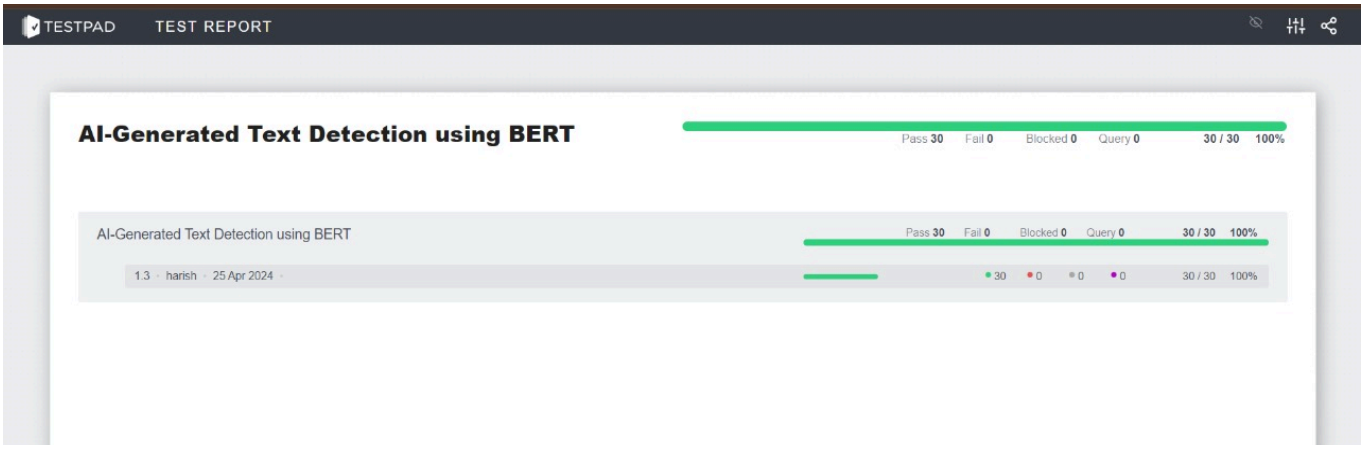

Figure 6.6 Test result 5

CHAPTER 7
CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

In the detection of AI-generated text versus human-generated text, both models CNN and DistilBert demonstrated very similar performance across metrics like F1 score, accuracy, macro average, and weighted average. From the positive results obtained in the confusion matrix, the DistilBERT model showed better capabilities in specific classifications, possibly due to its ability to capture more fine-grained linguistic patterns and semantics compared to the CNN model. While the general performance of the two models was quite close to each other, the DistilBERT model seemed to perform better in some specific cases, leading to fewer misclassifications or false positives/negatives. Our results suggested that both CNN and DistilBERT models could be effective in detecting AI-generated text but that the DistilBERT model had better capabilities in specific classification tasks.From the positive results obtained in the confusion matrix, the DistilBERT model showed better capabilities in specific classifications, possibly due to its ability to capture more fine-grained linguistic patterns and semantics compared to the CNN model. The DistilBERT model can extract semantic information from the text, while the CNN can capture structural and syntactic features, enhancing the model's ability to differentiate between natural and artificially generated text.The above models are robust and generalizable, ensuring reliable detection even with text samples that it hasn't seen or which are varied. As a result, DistilBERT is preferred since even though it performs marginally better, the chances of further enhancing content verification tools for countering the deceptive AI-generated contents are higher.. Overall, the performance of DistilBERT suggests its potential as a preferred choice for detecting AI-generated text, laying groundwork for improved content verification tools to combat deceptive AI-generated content.The CNN’s model accuracy is 99.62 and the DistilBert Classifier model’s accuracy is 99.76.

7.2 FUTURE ENHANCEMENT

For future enhancements to your content verification system based on the DistilBERT model, consider strategies such as data augmentation to diversify the training dataset, continuous fine-tuning on new data to adapt to evolving trends, ensemble methods to leverage complementary model strengths, and transfer learning for improved performance with less labeled data.

In the realm of AI-generated text detection, the fusion of DistilBERT and Convolutional Neural Networks (CNN) stands as a potent approach. Its efficacy lies in DistilBERT's adeptness at contextual comprehension and CNN's proficiency in capturing local patterns. By amalgamating these strengths, the model exhibits a commendable ability to discriminate between human-crafted and AI-generated text. To fortify this framework for future deployments, several enhancements warrant exploration. Firstly, continual fine-tuning of DistilBERT on diverse datasets, including emerging AI-generated text variations, promises to bolster its linguistic acumen and discernment capabilities. Augmentation techniques such as synonym replacement and paraphrasing could inject variability into the training data, rendering the model more resilient to adversarial attacks. Moreover, integrating multi-modal inputs, such as images or metadata associated with text, could enrich contextual understanding and enhance detection accuracy. Ensemble models, aggregating predictions from multiple DistilBERT-CNN iterations, could further elevate performance and confidence levels.

Through these advancements, the fusion of DistilBERT and CNN for AI-generated text detection not only stands poised to tackle contemporary challenges but also anticipates and addresses future complexities in the dynamic landscape of artificial intelligence-generated content.

APPENDIX

SCREENSHOTS

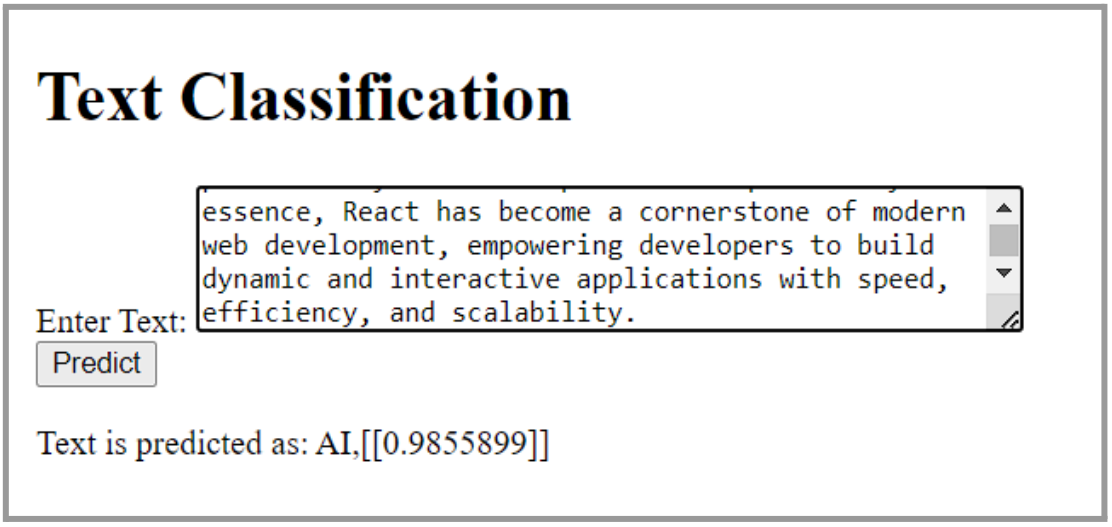

Figure A.1: AI content

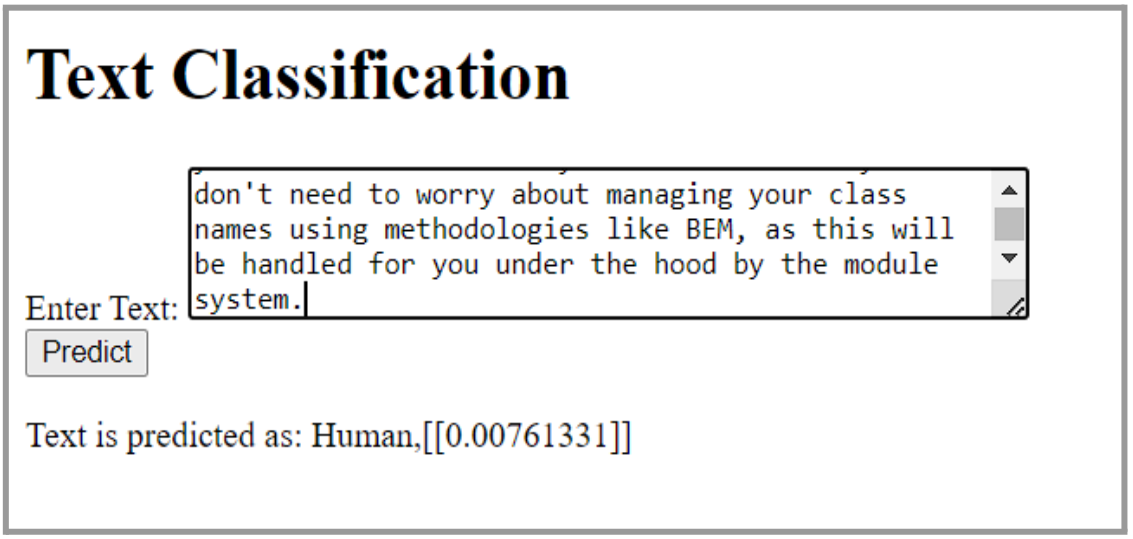

Figure A.2: Human Content

REFERENCES

1. Devlin J, Chang M. W, Lee K & Toutanova K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics.
2. Gehring J, Auli M, Grangier D, Yarats D, & Dauphin Y. N. (2017). Convolutional sequence to sequence learning. Proceedings of the 34th International Conference on Machine Learning.
3. Jerin Mahibha C, Sampath Kayalvizhi , Durairaj Thenmozhi & Sundar Arumina. (2021). Offensive Language Identification using Machine Learning and Deep Learning Techniques.
4. Jerin Mahibha C, Swaathi C M, Jeevitha R , R Princy Martina & Durairaj Thenmozhi .(2023). Brainstormers_msec at SemEval-2023 Task 10 : Detection of sexism related comments in social media using deep learning.
5. Radford A, Wu J, Child R, Luan D, Amodei D & Sutskever I. (2019). Language models are unsupervised multi task learners. OpenAI Blog.
6. Raffel, C, Shazeer N, Roberts A, Lee K, Narang S, Matena M & Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683.
7. Sanh V, Debut L, Chaumond J & Wolf T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.
8. Sivamanikandan S, Santhosh V, Sanjaykumar N & Jerin Mahibha C,Thenmozhi Durairaj. (2022). scubeMSEC@LT-EDI-ACL2022:Detection of Depression using Transformer Models.
9. Zhang Y & Wallace B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820.
10. Zhang X, Zhao J & LeCun Y. (2015). Character-level convolutional networks for text classification. Advances in Neural Information Processing Systems.

LIST OF PUBLICATIONS

1. Jerin Mahibha C, Adithya Manikandan J, Anirudh K, Arjun Krishnan R (2024), "Detecting The Invisible Hand: A Comparative Study Of CNN And DistilBert Models For Distinguishing AI-Generated And Human-Generated Text" published on International Journal Of Research Publication and Reviews.