

**AI FOR A POSITIVE WEB:  
ANALYZING HATE IN SOCIAL MEDIA  
BY  
CHAI YUN WAI**

**A REPORT  
SUBMITTED TO  
Universiti Tunku Abdul Rahman  
in partial fulfillment of the requirements  
for the degree of  
BACHELOR OF INFORMATION SYSTEMS (HONOURS) BUSINESS INFORMATION  
SYSTEMS  
Faculty of Information and Communication Technology  
(Kampar Campus)**

**JUNE 2024**

## REPORT STATUS DECLARATION FORM

**Title:** \_\_\_\_\_ AI FOR A POSITIVE WEB: \_\_\_\_\_  
\_\_\_\_\_ ANALYZING HATE IN SOCIAL MEDIA \_\_\_\_\_  
\_\_\_\_\_

**Academic Session:** \_\_\_Jun 2024\_\_\_


I \_\_\_\_\_ CHAI YUN WAI \_\_\_\_\_  
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

\_\_\_\_\_ *chaiyunwai* \_\_\_\_\_  
(Author's signature)

\_\_\_\_\_  \_\_\_\_\_  
(Supervisor's signature)

**Address:**

\_\_\_\_\_ LC 697, Kampung Baru \_\_\_\_\_  
\_\_\_\_\_ 31700, Malim Nawar, \_\_\_\_\_  
\_\_\_\_\_ Perak \_\_\_\_\_

Dr. Abdulkarim Kanaan Jebna  
Supervisor's name

**Date:** \_\_\_13 September\_2024\_\_\_\_\_

**Date:** 13/09/2024 \_\_\_\_\_

<b>Universiti Tunku Abdul Rahman</b>			
Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b>			
Form Number: <b>FM-IAD-004</b>	Rev No.: <b>0</b>	Effective Date: <b>21 JUNE 2011</b>	Page No.: <b>1 of 1</b>

**FACULTY/INSTITUTE\* OF \_\_\_\_\_ FICT \_\_\_\_\_**

**UNIVERSITI TUNKU ABDUL RAHMAN**

Date: \_\_\_\_\_ 13 September 2024 \_\_\_\_\_

**SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS**

It is hereby certified that \_\_\_\_\_ Chai Yun Wai \_\_\_\_\_ (ID No: 22ACB00222 ) has completed this final year project/ dissertation/ thesis\* entitled “ AI FOR A POSITIVE WEB: ANALYZING HATE IN SOCIAL MEDIA ” under the supervision of \_\_\_\_\_ Dr Abdulkarim Kanaan Jebna \_\_\_\_\_ (Supervisor) from the Department of \_\_\_\_\_ FICT \_\_\_\_\_, Faculty/Institute\* of \_\_\_\_\_ Universiti Tunku Abdul Rahman \_\_\_\_\_ , and \_\_\_\_\_ Dr Choo Peng Yin \_\_\_\_\_ (Co-Supervisor)\* from the Department of \_\_\_\_\_ FICT \_\_\_\_\_, Faculty/Institute\* of \_\_\_\_\_ Universiti Tunku Abdul Rahman \_\_\_\_\_.

I understand that University will upload softcopy of my final year project / dissertation/ thesis\* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

\_\_\_\_\_ Chai Yun Wai \_\_\_\_\_  
(Student Name)

\*Delete whichever not applicable

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**METHODOLOGY, CONCEPT AND DESIGN OF A 2-MICRON CMOS DIGITAL BASED TEACHING CHIP USING FULL-CUSTOM DESIGN STYLE**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : \_\_\_\_\_ *chaiyunwai* \_\_\_\_\_

Name : \_\_\_\_\_ Chai Yun Wai \_\_\_\_\_

Date : \_\_\_\_\_ 13 September 2024 \_\_\_\_\_

## **ACKNOWLEDGEMENTS**

I would like to express thanks and appreciation to my supervisor, Dr. Dr Abdulkarim Kanaan Jebna and my moderator, Dr Choo Peng Yin who have given me a golden opportunity to involve in the Internet of Things field study. Besides that, they have given me a lot of guidance in order to complete this project. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor and moderator.

## ABSTRACT

Hate speech detection on social media is a significant challenge due to the diverse and evolving nature of online language. This project aims to create an effective and user-friendly hate speech detection system using advanced machine learning and deep learning techniques. By developing various models, including Logistic Regression, Naive Bayes, Decision Trees, LSTM, BiLSTM, and CNN-LSTM, and incorporating an ensemble learning approach with a voting classifier, the system improves detection accuracy and reliability. A web interface built with Streamlit allows users to test text inputs and understand model decisions through explainability tools like SHAP and LIME. The best model achieved an accuracy of 88% with strong precision and recall, demonstrating the effectiveness of the proposed solution in detecting hate speech while maintaining *CNN\_LSTM Training Phase* interpretability and ease of use.

## Contents

<b>CHAPTER 1</b> .....	<b>1</b>
<b>INTRODUCTION</b> .....	<b>1</b>
1.1 Problem Statement and Motivation .....	2
1.2 Objectives .....	3
1.3 Project Scope and Direction.....	4
1.4 Contributions.....	5
1.5 Report Organization.....	6
<b>CHAPTER 2</b> .....	<b>8</b>
<b>LITERATURE REVIEW</b> .....	<b>8</b>
2.1 Review of the Existing Work.....	8
2.1.1 HateBERT .....	8
2.1.2 HateXplain .....	10
2.1.3 Deep Learning for Hate Speech Detection .....	11
2.1.5 Application of the Dataset in Model Training .....	14
2.1.6 Summary of Existing Systems .....	16
2.2 Review of the Technologies.....	17
2.2.1 Hardware Platform.....	17
2.2.2 Firmware/OS.....	17
2.2.3 Database .....	18
2.2.4 Programming Language.....	18
2.2.5 Algorithm.....	19
2.2.6 Summary of the Technologies Review .....	21
<b>SYSTEM METHODOLOGY/APPROACH</b> .....	<b>23</b>
3.1 CRISP-DM Methodology Overview .....	23
3.2 Description of CRISP-DM Phases.....	23
<b>SYSTEM DESIGN</b> .....	<b>32</b>
4.1 System Block Diagram .....	32
4.2 Use Case Description.....	41

4.3 System Components Specifications .....	45
<b>SYSTEM IMPLEMENTATION.....</b>	<b>46</b>
5.1 Implementation of CRISP-DM Methodology .....	46
<b>SYSTEM EVALUATION AND DISCUSSION.....</b>	<b>63</b>
6.1 Testing Cases .....	63
6.2 Model Evaluation.....	64
6.3 Project Challenges .....	66
<b>CONCLUSION AND RECOMMENDATIONS.....</b>	<b>67</b>
7.1 Conclusion .....	67
7.2 Recommendations.....	68
<b>REFERENCES.....</b>	<b>70</b>
<b>FINAL YEAR PROJECT WEEKLY REPORT .....</b>	<b>72</b>
<b>PLAGIARISM CHECK RESULT .....</b>	<b>74</b>



# LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 3.1.1	<b>CRISP-DM Methodology</b>	1
Figure 4.1.1	Training Process for Machine Learning Approach	10
Figure 4.1.2	Training Process for Deep Learning Approach	13
Figure 4.1.3	Hate Speech Detection System Use Case Diagram	14
Figure 5.1.1	Word Cloud for All Comments	19
Figure 5.1.2	Word Cloud for NonHate Comments	20
Figure 5.1.3	Word Cloud for Hate Comments	
Figure 5.1.4	CNN_LSTM Training Phase	
Figure 5.1.3		

## **LIST OF TABLES**

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 2.1	<b>Specifications of laptop</b>	4
Table 2.2	<b>Comparison between the existing applications and proposed application</b>	16
Table 3.1	<b>Use Case Description of Register Account</b>	18



## LIST OF ABBREVIATIONS

<i>TFIDF</i>	Term Frequency-Inverse Document Frequency
<i>CNN_LSTM</i>	Convolutional Neural Network-Long Short-Term Memory
<i>LSTM</i>	Long Short-Term Memory
<i>SHAP</i>	SHapley Additive exPlanations
<i>LIME</i>	Local Interpretable Model-agnostic Explanations
<i>BiLSTM</i>	Bidirectional Long Short-Term Memory

# Chapter 1

## Introduction

In recent years, the prevalence of hate speech on social media platforms has reached alarming levels, prompting urgent calls for effective detection and mitigation strategies. A 2023 report by the European Union Agency for Fundamental Rights revealed that over 52% of social media users in the EU have encountered hate speech online, highlighting the widespread nature of this issue. In the United States, a 2022 study by the Anti-Defamation League indicated that 41% of Americans experienced online harassment, with marginalized groups, such as racial minorities, facing disproportionately higher rates of abuse.

The rise of hate speech is not merely a digital concern; it has real-world implications. The number of hate crimes reported in major U.S. cities increased by 11% in 2023, with notable spikes in anti-Jewish and anti-Muslim incidents, driven in part by socio-political tensions [16]. This escalating trend underscores the urgent need for comprehensive approaches to combat hate speech and its associated risks, which can lead to discrimination, violence, and societal fragmentation.

Despite the growing awareness of these issues, current mechanisms for detecting hate speech often fall short. Many automated systems lack transparency and accessibility, leaving users without a clear understanding of how their content is evaluated. Furthermore, the nuanced nature of language complicates the identification of hate speech, as definitions can vary widely across different contexts.

This project aims to address these challenges by developing an interactive web page that empowers users to assess their comments for hate speech. By promoting accessibility, transparency, and community engagement, this initiative seeks to foster a more responsible and respectful online environment. Through this platform, users will gain insights into the potential impact of their communications, contributing to a broader culture of awareness and accountability in the digital landscape.

# 1.1 Problem Statement and Motivation

## Problem Statement

The proliferation of hate speech on social media platforms has become a pressing concern in today's digital age. Hate speech, defined as written or oral communication that abuses or threatens a specific group or target based on characteristics such as race, religion, ethnic origin, national origin, sex, disability, sexual orientation, or gender identity, poses significant risks to individuals and communities [1]. The spread of hate speech can lead to increased tensions, discrimination, and even violence against targeted groups. Despite the growing awareness of this issue, effective mechanisms for detecting and mitigating hate speech remain elusive due to the complexity of language and the subtleties involved in identifying hate speech content [2].

Current approaches to hate speech detection often rely on automated systems integrated within larger platforms, which may not provide users with direct access to these tools or transparency into how they operate [3]. This lack of accessibility and understanding can hinder efforts to combat hate speech effectively. Moreover, existing models frequently struggle with the nuances of language and the varying definitions of hate speech across different contexts [2]. Additionally, the availability of training data for these systems is limited, further complicating their development and refinement [2].

## Motivation

To address the challenges of detecting hate speech effectively, this final year project aims to create an interactive web page where individuals can test their comments or texts for hate speech. By providing a user-friendly interface for evaluating hate speech content, this project seeks to empower users with insights into the potential impact of their online communications. The initiative is driven by the following key factors:

- i. **Accessibility:** By making hate speech detection accessible to individual users, the project fosters greater awareness and responsibility among online communicators. Even if the model's predictions are not always perfect, the tool provides users with a means to understand the potential risks associated with their language or content.

ii. **Transparency:** Offering insights into the decision-making process behind hate speech detection can enhance trust in automated systems. The project employs interpretability methods such as LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations) to show users how their text is being evaluated. This transparency is crucial for building trust, even if the model's judgments are occasionally inconsistent.

iii. **Community Engagement:** Engaging individuals directly in the detection process promotes a sense of community responsibility for maintaining a positive online environment. By allowing users to test and understand the impact of their comments, the project encourages more thoughtful and respectful online communication.

In summary, this project aims to meet the critical need for accessible and transparent hate speech detection tools by developing a user-friendly web page that empowers individuals to test their content for hate speech. Although the current models may sometimes provide imperfect results, the approach focuses on raising awareness, promoting engagement, and fostering a culture of responsibility among online users.

## 1.2 Objectives

The primary objectives of this project are to develop an effective hate speech detection system and to ensure its usability and interpretability. The specific objectives are:

- i. **Create a Model Able to Distinguish Hate and Non-Hate:**
  - Develop a robust model that accurately distinguishes between hate speech and non-hate speech. This includes comprehensive data cleaning and preprocessing to ensure data quality, such as removing inconsistencies, handling missing values, and standardizing text data.

ii. **Experiment with Different Algorithms to Build Models:**

- Experiment with various algorithms to build and optimize hate speech detection models. This includes training machine learning algorithms such as Logistic Regression, Naive Bayes, and Decision Trees, as well as deep learning architectures like LSTM, BiLSTM, and CNN-LSTM. Evaluate the performance of these models using metrics such as accuracy, precision, recall, and F1-score.

iii. **Incorporate Ensemble Learning Techniques:**

- Implement ensemble learning methods, such as Voting Classifiers, to combine the predictions of multiple models. This approach aims to enhance overall performance and robustness by leveraging the strengths of different models and reducing individual model weaknesses.

iv. **Enhance Model Interpretability:**

- Utilize interpretability techniques such as SHAP and LIME to provide transparent and understandable explanations of the model's predictions. This will help users to understand how different features contribute to the detection of hate speech and build trust in the system's decisions.

v. **Develop a User Interface:**

- Create a web-based interface using Streamlit that allows users to easily input text and receive real-time feedback on potential hate speech. The interface should be intuitive and provide clear insights into the results and explanations generated by the models.

### 1.3 Project Scope and Direction

The scope of this project involves developing a comprehensive hate speech detection system using advanced machine learning and deep learning techniques. The project aims to create a robust and user-friendly tool that enables individuals to assess their text content for hate speech. The key components of the project are outlined as follows:

i. **Model Development:**

- **Training Models:** Develop and train various machine learning and deep learning models for hate speech detection. This includes experimenting with different algorithms to build effective models that can accurately classify text as hate speech or non-hate speech.



- **Model Evaluation:** Evaluate the trained models to ensure they meet performance criteria. This involves assessing the models' accuracy, precision, recall, and overall reliability in detecting hate speech.

**ii. User Interface Creation:**

- **Web Interface:** Design and develop an intuitive web-based interface using Streamlit. This interface will allow users to input their text and receive real-time feedback on the likelihood of hate speech.

**iii. Interpretability and Transparency:**

- **Explainability Tools:** Integrate explainability tools such as SHAP and LIME to provide users with insights into how the models make their predictions. This transparency is crucial for building trust and understanding in the system's decisions.

The project is designed to offer a comprehensive solution for hate speech detection with several key capabilities:

- **Versatility in Classification:** Supports multiple classifiers and models, including ensemble methods, allowing users to choose the most suitable model for their needs.
- **Real-Time Analysis:** Provides immediate feedback on text input, enabling quick and efficient detection of hate speech.
- **Enhanced Transparency:** Utilizes advanced explainability tools to make model predictions more understandable and trustworthy.
- **Scalability:** The system is designed to handle varying amounts of data and text inputs, making it adaptable to different use cases and environments.

## 1.4 Contributions

**i. Comprehensive Exploration of Methodologies:**

- **Diverse Method Application:** This project contributes by applying a wide range of methodologies to a dataset that has seen limited exploration. The project explores various preprocessing techniques, including stopword removal and lemmatization, and evaluates different model architectures, such as embedding-based LSTM models. By thoroughly investigating these methods, the project identifies effective approaches

for training models on this dataset, thus contributing new insights into optimizing hate speech detection systems.

**ii. Enhanced Usability and Accessibility:**

- **User Interface:** By creating an intuitive web-based interface, the project makes advanced hate speech detection technology accessible to individual users. This contribution is significant for democratizing the use of such tools and providing users with a straightforward way to assess their text content.

**iii. Increased Transparency and Trust:**

- **Explainability Tools:** The integration of SHAP and LIME for model interpretability enhances the transparency of the hate speech detection system. This contribution helps users understand how the models arrive at their predictions, fostering greater trust and confidence in the technology.

**iv. Educational Value:**

- **Showcase of Practical Application:** The project serves as a practical example of applying advanced machine learning and deep learning techniques to a real-world problem. It provides valuable insights into model development, evaluation, and deployment, which can be useful for researchers, developers, and practitioners in the field.

## 1.5 Report Organization

This report is organized to provide a comprehensive overview of the hate speech detection project, structured to guide the reader through the development process and findings. Chapter 1, Introduction, sets the foundation by presenting the problem statement, motivation behind the project, the scope of work, and the contributions made. It provides context for understanding the importance and objectives of the hate speech detection system.

Chapter 2, Literature Review, delves into the existing body of research on hate speech detection. It reviews current methods, discusses challenges and advancements in the field, and identifies gaps that the project aims to address. This chapter establishes the academic and practical context for the project's approach.

Chapter 3, System Methodology/Approach, outlines the methodologies employed in the project. It details the processes of data cleaning, preprocessing techniques, and the development and

training of various models. This chapter explains the rationale behind the selected methods and the experimental setup used to achieve the project goals.

Chapter 4, System Design, focuses on the design aspects of the hate speech detection system. It describes the architectural decisions, design choices, and the integration of different components, including the machine learning models and the user interface. This chapter illustrates how the system was structured to meet the project's objectives.

Chapter 5, System Implementation, provides insights into the practical implementation of the system. It covers the development of machine learning and deep learning models, the creation of the web interface using Streamlit, and the integration of explainability tools like SHAP and LIME. This chapter details the steps taken to bring the system from design to deployment.

Chapter 6, System Evaluation and Discussion, evaluates the performance of the hate speech detection system. It assesses the models' accuracy, precision, recall, and overall effectiveness, discussing the results and challenges encountered during the evaluation phase. This chapter offers a thorough analysis of how well the system meets its goals.

Finally, Chapter 7, Conclusion and Recommendations, summarizes the project's key findings and contributions. It reflects on the impact of the work and provides recommendations for future research and development. This chapter highlights the overall significance of the project and suggests areas for further exploration.

# Chapter 2

## Literature Review

The emergence of hate speech as a pervasive issue in online communication has prompted significant scholarly and technological interest in developing effective detection systems. As social media platforms continue to grow in popularity and influence, the challenge of identifying and mitigating hate speech has become increasingly critical. Researchers and developers have explored various methodologies, ranging from traditional rule-based approaches to advanced machine learning techniques, in an effort to create systems that can accurately discern hate speech from benign content. Existing literature highlights the complexity of hate speech detection, which involves not only linguistic analysis but also an understanding of contextual nuances and cultural sensitivities. A review of current systems reveals a diverse array of applications, each employing different algorithms and frameworks to tackle the multifaceted nature of hate speech. For instance, some systems utilize natural language processing (NLP) techniques to analyze text, while others incorporate user-generated content and feedback to refine their detection capabilities. Despite advancements in technology, many existing applications face challenges related to accuracy, scalability, and user engagement. The lack of transparency in how these systems operate often leads to mistrust among users, further complicating efforts to combat hate speech effectively. This literature review aims to critically examine the strengths and weaknesses of current hate speech detection systems, providing a foundation for the development of more accessible and transparent tools that empower users to take an active role in addressing this pressing social issue.

### 2.1 Review of the Existing Work

#### 2.1.1 HateBERT

##### Overview

HateBERT is a hate speech detection model that leverages the BERT (Bidirectional Encoder Representations from Transformers) architecture, fine-tuned specifically for identifying hate speech in textual data. BERT's deep learning-based approach allows HateBERT to understand the context and subtle nuances of language, which is crucial for detecting various forms of hate

speech, including covert expressions, sarcasm, or coded language that might not be overtly offensive but still have harmful implications.

### **Key Features**

- i. **Architecture:** HateBERT is based on the BERT model, which is designed to capture the contextual relationships in text by leveraging bidirectional training of Transformer models. The model was further fine-tuned on large, annotated hate speech datasets to enhance its ability to differentiate between hate speech and non-hate speech.
- ii. **Training Data:** The model was trained on diverse datasets that encompass a broad range of hate speech examples, collected from social media platforms, forums, and other online sources. This variety helps ensure that the model can generalize well across different contexts, languages, and expressions of hate speech.
- iii. **Performance Metrics:** HateBERT has demonstrated competitive performance metrics, such as accuracy and F1 scores, on various benchmark datasets. For instance, in some studies, HateBERT achieved an accuracy of approximately 82% on the Stormfront dataset, outperforming many traditional machine learning approaches [2][9]. This performance highlights its capability to effectively handle complex hate speech detection tasks in different contexts.
- iv. **Explainability:** One significant advantage of HateBERT is its explainability. The model uses the attention mechanisms inherent in BERT to provide insights into its decision-making process, highlighting specific parts of the text that contributed most to its classification outcomes. This feature is crucial for building trust in automated hate speech detection systems.

### **Limitations**

- i. **Contextual Challenges:** Despite its strengths, HateBERT may struggle with context-dependent hate speech, such as sarcasm, humor, or coded language that was not explicitly labeled in the training data. Its reliance on labeled data means that any missing context or nuance in the data can impact the model's performance.
- ii. **Resource Intensive:** The model requires significant computational resources for both training and inference. The need for extensive GPU or TPU resources may limit its accessibility for smaller organizations or applications with limited infrastructure.
- iii. **Dataset Bias:** The effectiveness of HateBERT heavily depends on the quality and diversity of the training datasets. If the training data is biased or lacks representation of

certain groups or forms of hate speech, the model's performance may suffer, leading to potential misclassifications.

HateBERT represents a notable advancement in the field of hate speech detection by combining the capabilities of deep learning with the contextual understanding of natural language processing. Its strengths lie in its ability to detect various forms of hate speech and provide explainable outputs. However, the model's reliance on extensive computational resources and potential dataset biases present challenges that require ongoing efforts to improve robustness and ensure fairness.

## 2.1.2 HateXplain

### Overview

HateXplain is a pioneering benchmark dataset specifically designed to advance explainable hate speech detection. Addressing the critical need for transparency and interpretability in automated hate speech detection systems, HateXplain not only provides classifications of hate speech but also offers the rationales behind these classifications. This focus on explainability enhances understanding of model decision-making, thereby fostering trust and accountability in AI systems.

### Key Features

- i. **Multi-Faceted Annotation:** HateXplain provides annotations from three distinct perspectives:
  - **Classification:** Posts are categorized into three classes: hate speech, offensive language, and normal content.
  - **Target Community:** Identifies the community affected by the hate speech or offensive language.
  - **Rationales:** Highlights specific text segments that informed the labeling decision, offering insight into the reasoning behind the classification.
- ii. **Dataset Composition:** The dataset includes a total of 20,148 posts, with 9,055 sourced from Twitter and 11,093 from Gab [10]. This diverse sourcing ensures broad representation of hate speech across different platforms.
- iii. **Explainability Focus:** HateXplain emphasizes explainability in hate speech detection. Research utilizing this dataset has demonstrated that models trained with human rationales exhibit reduced bias against target communities, thus contributing to the development of fair and accountable AI systems.

- iv. **Performance Evaluation:** HateXplain has been used to benchmark various state-of-the-art models, highlighting that even high-performing classifiers often lack explainability. Models incorporating human rationales tend to achieve higher scores on explainability metrics such as model plausibility and faithfulness.

### Limitations

- i. **Language Restriction:** The dataset is primarily focused on English, limiting its applicability in multilingual contexts. This is a notable limitation given the global nature of online communication.
- ii. **Lack of Contextual Information:** The dataset does not include additional contextual information such as user profiles or historical posting behavior, which could provide further insights for classification tasks.
- iii. **Inter-Annotator Agreement:** Despite efforts to ensure high-quality annotations, the inter-annotator agreement score (Krippendorff's alpha of 0.46) indicates some variability in labeling, which may affect the dataset's reliability.

HateXplain represents a significant advancement in the field of hate speech detection by prioritizing explainability and accountability. Its comprehensive annotation framework enables researchers and developers to build more transparent models that effectively address the complexities of online hate speech. Future improvements, including expanding linguistic coverage and incorporating contextual information, will be essential for enhancing the dataset's utility and impact.

## 2.1.3 Deep Learning for Hate Speech Detection

### Overview

This system presents a comparative study of various deep learning methods for hate speech detection on social media platforms, particularly Twitter. The study evaluates the effectiveness of different deep learning architectures, including Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTMs), in identifying hate speech and compares their performance with traditional machine learning approaches [11][12].

### Key Features

- i. **Comprehensive Evaluation:** The study analyzes several deep learning models, including CNN, LSTM, and Bidirectional LSTM (BiLSTM), to identify the most effective algorithms for hate speech detection. It also compares these deep learning

models to traditional machine learning methods such as Logistic Regression and Support Vector Machines.

- ii. **Datasets:** Researchers utilize multiple publicly available datasets for hate speech, including the Waseem and Hovy dataset, the Davidson et al. dataset, and the Founta et al. dataset. These datasets offer a diverse range of examples, encompassing tweets labeled as hate speech, offensive language, and neutral content.
- iii. **Performance Metrics:** The evaluation is based on standard performance metrics including accuracy, precision, recall, and F1-score. Additionally, the study examines the models' ability to handle imbalanced datasets, a common challenge in hate speech detection.
- iv. **Scalability:** The deep learning models are designed to handle large-scale data typical of social media platforms, showcasing their potential for real-world applications.

### **Findings**

The study reveals that deep learning models, particularly BiLSTM, outperform traditional machine learning methods in detecting hate speech on Twitter. BiLSTM achieves the highest F1-score of 0.78 on the Waseem and Hovy dataset.

CNN models also demonstrate promising results by capturing relevant features and patterns in the text. However, they tend to be more sensitive to the length of the input text compared to LSTM-based models.

The performance of all models is influenced by the quality and diversity of the training data[11][13]. Datasets with more examples and better representation of different types of hate speech lead to improved classification accuracy.

### **Limitations**

- i. The study is limited to English-language datasets, which may affect the performance of the models in other languages or dialects.
- ii. The computational resources required for training deep learning models can be a barrier for smaller organizations or those with limited infrastructure.
- iii. The models may struggle with context-dependent hate speech or subtle, coded forms of hate speech.

The comparative study highlights the potential of deep learning techniques for effective hate speech detection on social media. By leveraging architectures such as BiLSTM and CNN, researchers can develop robust models that surpass traditional approaches. However, ongoing



efforts to improve model robustness, address imbalanced datasets, and adapt to different languages and contexts are essential for broader adoption and real-world impact.

## 2.1.4 A Curated Dataset for Hate Speech Detection on Social Media Text

### Overview

The dataset titled "A Curated Dataset for Hate Speech Detection on Social Media Text" is a comprehensive resource specifically designed for research in hate speech detection[14]. It aggregates data from various social media platforms, such as Twitter and Reddit, and other sources like Kaggle and GitHub, providing a rich dataset for training and evaluating machine learning models aimed at identifying hate speech in text data. This dataset is particularly valuable for researchers and developers seeking to create robust hate speech detection systems.

### Key Features

- i. **Diverse Sources:** The dataset is compiled from multiple online sources, ensuring a wide range of examples that reflect the complexity of hate speech in real-world contexts. This diversity helps models generalize better across different types of social media interactions.
- ii. **Annotation Process:** Each entry in the dataset is meticulously labeled by human annotators. The annotation process classifies posts into two primary categories:
  - *Hate Speech:* Texts labeled as '1', indicating content that promotes violence or hatred against individuals or groups based on attributes such as race, ethnicity, religion, gender, or sexual orientation.
  - *Non-Hate Speech:* Texts labeled as '0', which may not necessarily promote violence but are considered derogatory or disrespectful.
- iii. **Dataset Size:** The dataset originally contains 451,709 entries, with 371,452 entries categorized as hate speech and 80,250 as non-hate speech. To enhance the dataset's balance and representativeness, an augmented version has been generated, consisting of 726,120 samples[14]. This augmented dataset helps create a custom vocabulary of 145,046 words, reducing the number of out-of-vocabulary words and improving model performance.
- iv. **Balanced Representation:** The dataset aims to provide a balanced representation of hate speech across different demographics and contexts. This balance is essential for

reducing bias in machine learning models, ensuring fair performance across various groups.

- v. **Textual Features:** The dataset includes various textual elements such as emoticons, emojis, hashtags, slang, and contractions, which are common in social media communications. This comprehensive set of features enhances the understanding of the intent behind the text and aids in effective hate speech detection.

### **Limitations**

- i. **Language Restriction:** Currently, the dataset is limited to English content, which may hinder its applicability in multilingual contexts, given that hate speech manifests differently across languages.
- ii. **Annotation Quality:** While the dataset is meticulously curated, the quality of annotations may vary. Human annotators may have differing interpretations of what constitutes hate speech, leading to inconsistencies in labeling.
- iii. **Temporal Context:** The dataset may not fully capture the evolving nature of hate speech, as language and social norms change over time. This temporal aspect is crucial for maintaining the relevance of hate speech detection systems.

"A Curated Dataset for Hate Speech Detection on Social Media Text" is a critical resource for advancing research and development in hate speech detection. Its diverse sources, comprehensive annotations, and augmented dataset size make it a valuable tool for training and evaluating machine learning models. However, researchers should be mindful of its limitations, particularly regarding language scope and annotation quality, as they work toward developing effective and fair hate speech detection systems.

## **2.1.5 Application of the Dataset in Model Training**

### **Overview**

The dataset has been widely applied on platforms like Kaggle, where one user achieved a validation accuracy of 0.8863 by training a model using this dataset [15]. This result demonstrates the effectiveness and potential of the dataset for hate speech detection tasks.

### **Preprocessing Steps**

Before training the model, the user performed extensive text preprocessing to enhance the model's performance. The preprocessing steps included:

- **Removal of Irrelevant Information:** Regular expressions were used to remove links, HTML tags, and extra whitespace from the text.
- **Punctuation Removal:** All punctuation marks were removed from the text to focus solely on the words themselves.
- **Number Removal:** Words containing numbers were removed to reduce noise.
- **Stopword Removal:** Common but meaningless words, such as "and" or "is," were filtered out using a defined stopwords list.
- **Lemmatization:** Words were reduced to their base form using lemmatization techniques to reduce vocabulary diversity and improve the model's generalization capability.

After these preprocessing steps, the text was tokenized and prepared for input into a neural network.

### **Model Architecture**

The user employed a simple yet effective deep learning model for hate speech detection, with the following architecture:

- **Embedding Layer:** Maps the input vocabulary to a 100-dimensional vector space to capture semantic relationships between words.
- **SpatialDropout1D Layer:** Applies spatial dropout after the embedding layer to reduce overfitting.
- **LSTM Layer:** Uses a Long Short-Term Memory (LSTM) network with 128 units to capture temporal dependencies and contextual information within the text, applying both dropout and recurrent dropout to further prevent overfitting.
- **Dense Layer:** A final dense layer with a sigmoid activation function outputs a binary classification result, indicating whether the text contains hate speech.

### **Model Performance**

Following the preprocessing and model training steps described above, the user achieved a validation accuracy of 0.8863 on the validation set, demonstrating the powerful capability of deep learning models in the task of hate speech detection.

This successful case not only validates the quality and applicability of the dataset but also provides valuable insights for future research and development. By leveraging the experience

of existing model training approaches, researchers can more effectively build and optimize their own hate speech detection systems.

### 2.1.6 Summary of Existing Systems

The reviewed systems and datasets demonstrate the diverse approaches to hate speech detection, ranging from specialized models like HateBERT and HateXplain to comprehensive datasets and deep learning methodologies. Each of these systems offers unique strengths and faces specific challenges:

- i. **HateBERT** leverages the BERT architecture's contextual understanding capabilities, fine-tuned for hate speech detection. It excels in handling nuanced language and provides explainable outputs through its attention mechanisms. However, its reliance on extensive computational resources and potential biases in training data are significant limitations.
- ii. **HateXplain** introduces a benchmark dataset emphasizing explainability by providing classifications with rationales and identifying affected communities. This dataset promotes transparency and fairness in hate speech detection models but is limited by its focus on English and the lack of additional contextual data.
- iii. **Deep Learning Methods** offer powerful solutions for hate speech detection, with architectures like BiLSTM and CNNs outperforming traditional machine learning models. However, they require substantial computational resources and may struggle with context-dependent or subtle hate speech.
- iv. **Curated Datasets** like "A Curated Dataset for Hate Speech Detection on Social Media Text" provide extensive resources for training and evaluating hate speech detection models. They offer diverse examples and comprehensive annotations, facilitating improved model generalization. Nonetheless, the limitations regarding language scope and the temporal context of hate speech remain challenges.
- v. **Applications in Model Training** highlight the effectiveness of datasets when combined with deep learning techniques. Through careful preprocessing and model design, high validation accuracy can be achieved, demonstrating the potential for practical deployment in real-world scenarios.

## **2.2 Review of the Technologies**

### **2.2.1 Hardware Platform**

In this project, the choice and utilization of hardware platforms play a critical role in the training and development process, given the substantial computational resources required for handling large datasets and training deep learning models. The following hardware platforms were selected to support different tasks:

The local Jupyter Notebook environment is used for developing and running most of the machine learning algorithms. These algorithms generally have lower computational complexity, allowing them to execute quickly on a local CPU. The local environment provides greater flexibility and control, enabling rapid iteration and debugging of models. However, the local Jupyter Notebook lacks GPU support, which can result in memory constraints or longer training times for deep learning tasks.

To overcome the limitations of local hardware, Google Colab is used as the primary platform for deep learning training. Google Colab offers free TPU and GPU options that significantly accelerate the model training process, as well as ample RAM to handle larger datasets. For deep learning tasks, either a GPU or TPU is chosen depending on specific requirements. When faster computation is needed, the GPU is selected; when the model demands extensive RAM, the TPU is utilized. However, in Google Colab, only one type of hardware (GPU or TPU) can be selected per training session, requiring careful consideration to make the best choice for each task.

When switching between Google Colab and the local Jupyter Notebook, it is essential to ensure consistency in the environment. Issues were encountered where a model trained on Google Colab could not be loaded correctly on the local Jupyter Notebook due to version mismatches. To mitigate this, efforts were made to maintain consistent software packages and framework versions across both platforms, ensuring cross-platform stability and reliability of the models.

### **2.2.2 Firmware/OS**

Operating Systems

- i. **Local Operating System:** The local development and training environment utilizes Windows. This operating system supports running machine learning algorithms via Jupyter Notebook for tasks that do not require extensive computational resources.
- ii. **Cloud Platform Operating System:** For tasks demanding higher computational power, Google Colab is employed, which operates on a Linux-based system. This environment provides access to advanced hardware resources, including TPUs and GPUs, essential for efficient deep learning model training.

### **Impact on the Project**

Windows is adequate for running machine learning models locally and handling initial experiments. However, the Linux-based environment in Google Colab enhances performance for deep learning tasks by offering additional computational resources.

## **2.2.3 Database**

### **Data Storage**

The dataset is obtained from Kaggle in CSV format. It includes two columns: "Content" and "Label." The "Content" column contains the text data, while the "Label" column indicates whether the text is categorized as hate speech (1) or non-hate speech (0).

### **Data Management**

- i. **Local Handling:** The CSV file is read and processed locally using Jupyter Notebook with the pandas library. This approach allows efficient loading and manipulation of the data for machine learning tasks.
- ii. **Cloud Handling:** For cloud-based training on Google Colab, the CSV file is stored in Google Drive and accessed through Google Colab's drive API. Data is read and processed using pandas to maintain consistency across environments.

### **Impact on the Project**

Utilizing CSV files and pandas for data management in both local and cloud environments ensures a streamlined and effective approach for handling and processing the dataset.

## **2.2.4 Programming Language**

### **Programming Languages Used**

Python is the primary programming language used for this project. It is widely adopted in the fields of machine learning and data science due to its extensive libraries and frameworks that

simplify complex tasks. Python provides robust support for data manipulation, model training, and evaluation.

### **Rationale for Choice**

- i. **Libraries and Frameworks:** Python's rich ecosystem of libraries, such as pandas for data manipulation, scikit-learn for machine learning algorithms, and PyTorch for deep learning models, makes it an ideal choice for developing and deploying machine learning solutions. These tools facilitate efficient development and implementation of various algorithms and models.
- ii. **Community Support:** Python has a large and active community that contributes to a wealth of resources, documentation, and support. This community involvement ensures that developers can access up-to-date solutions and best practices.

### **Additional Tools**

Streamlit is also used in this project to create a user-friendly web interface. As a Python-based framework, Streamlit allows for quick and efficient development of interactive web applications. It provides an easy way to deploy machine learning models and offers real-time feedback to users through a simple and intuitive interface.

### **Advantages**

- i. **Ease of Use:** Python's syntax is straightforward and readable, which accelerates development and reduces the learning curve for new users.
- ii. **Flexibility:** Python's versatility allows it to be used across different stages of the project, from data preprocessing to model training, evaluation, and web interface creation.
- iii. **Integration:** Python seamlessly integrates with various platforms and tools, such as Jupyter Notebook, Google Colab, and Streamlit, enabling efficient workflow management and collaboration.

## **2.2.5 Algorithm**

### **Logistic Regression**

Logistic Regression is a statistical technique used for binary classification tasks. It employs a logistic (or sigmoid) function to convert a linear combination of input features into a probability value between 0 and 1. The primary objective is to determine the optimal weight parameters that minimize the discrepancy between predicted probabilities and actual outcomes. Logistic Regression is widely used in areas such as medical diagnosis, credit scoring, and marketing.

## **Naive Bayes**

Naive Bayes is a straightforward yet powerful classification algorithm that relies on Bayes' Theorem. It operates under the assumption that the features are conditionally independent given the class label, allowing the calculation of joint probabilities by multiplying individual feature probabilities. Due to its simplicity and computational efficiency, Naive Bayes is especially suitable for text classification tasks like spam detection and sentiment analysis. [4][5].

## **Decision Tree**

Decision Trees are a type of algorithm used for both classification and regression tasks, employing a tree-like structure to represent decisions and their possible outcomes. The algorithm divides the dataset into smaller subsets based on feature-based rules until a specific stopping condition is met, such as maximum depth or minimum impurity at the leaf nodes. Each internal node represents a decision rule, each branch represents an outcome, and each leaf node indicates a class label. Decision Trees are valued for their interpretability and are frequently used in applications such as customer segmentation and risk assessment.

## **Long Short-Term Memory (LSTM)**

Long Short-Term Memory (LSTM) networks are a specialized type of Recurrent Neural Network (RNN) designed to capture long-term dependencies in sequential data. LSTMs utilize memory cells along with gating mechanisms, including input, forget, and output gates, to regulate information flow and mitigate the vanishing gradient problem that often occurs in traditional RNNs. These networks are widely applied in fields like time series forecasting, speech recognition, and natural language processing [6].

### **2.2.5.5 Bidirectional Long Short-Term Memory (BiLSTM)**

Bidirectional Long Short-Term Memory (BiLSTM) is an advanced version of the LSTM network that processes data in both forward and backward directions along the time sequence. By analyzing input from both directions, BiLSTM provides a more comprehensive context, significantly enhancing performance in various sequential tasks such as machine translation and sentiment analysis. [7].



## **Convolutional Neural Network-Long Short-Term Memory (CNN-LSTM)**

The CNN-LSTM model integrates the strengths of Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN). While LSTM networks are adept at capturing long-term dependencies in sequential data, CNNs excel at extracting local patterns from the data using their convolutional structures. By combining these capabilities, the CNN-LSTM architecture is highly effective for text classification tasks, as it can learn both sequential dependencies and local features in textual data. [8].

### **2.2.6 Summary of the Technologies Review**

This section reviews the various technologies utilized in this project, highlighting their contributions and roles in achieving the project's objectives.

- i. **Hardware Platform:** The project relies on two main hardware platforms, a local Jupyter Notebook environment and Google Colab. The local environment, running on a CPU, is suitable for developing and testing machine learning models with lower computational demands, while Google Colab provides access to powerful GPUs and TPUs, significantly speeding up deep learning model training. The use of both platforms allows for flexibility and resource optimization, ensuring efficient model development and training.
- ii. **Firmware/OS:** The choice of operating systems, including Windows for the local environment and Linux for the cloud-based environment (Google Colab), directly impacts the project's computational efficiency and scalability. Windows supports initial model development and experiments, while the Linux-based Google Colab offers the necessary computational power for deep learning tasks, benefiting from its advanced hardware capabilities.
- iii. **Database:** Data management is crucial to the project's success. The dataset, stored in CSV format, is managed consistently across local and cloud environments using the pandas library. This approach allows seamless data handling, manipulation, and preprocessing, ensuring a streamlined workflow and minimizing potential issues related to data transfer and versioning between different platforms.

- iv. **Programming Language:** Python is the primary programming language used throughout the project due to its extensive libraries and frameworks, such as pandas, scikit-learn, and PyTorch, which facilitate efficient development and deployment of machine learning models. The choice of Python enhances ease of use, flexibility, and integration with other tools like Streamlit, enabling the creation of a user-friendly web interface for real-time feedback.
  
- v. **Algorithms:** A diverse set of algorithms was employed to tackle the hate speech detection task. Logistic Regression serves as a straightforward yet effective approach for binary classification problems, making it a strong baseline model. Naive Bayes is particularly useful for text classification tasks due to its simplicity and efficiency. Decision Trees offer interpretability and are effective for both classification and regression tasks. For deep learning models, various architectures were explored to handle the complexity and nuances of text data. LSTM (Long Short-Term Memory) networks are designed to capture long-term dependencies in sequence data, which is crucial for understanding the context in text. BiLSTM (Bidirectional Long Short-Term Memory) extends the LSTM by processing input sequences in both forward and backward directions, allowing the model to gain a more comprehensive understanding of the context. The CNN-LSTM model combines the strengths of Convolutional Neural Networks (CNNs) and LSTMs, enabling it to capture both local patterns and sequential dependencies in text, making it particularly well-suited for text classification tasks.

# Chapter 3

## System Methodology/Approach

### 3.1 CRISP-DM Methodology Overview

The Cross-Industry Standard Process for Data Mining (CRISP-DM) is a widely recognized framework for conducting data mining projects. This methodology consists of six key phases, as illustrated in the diagram below:

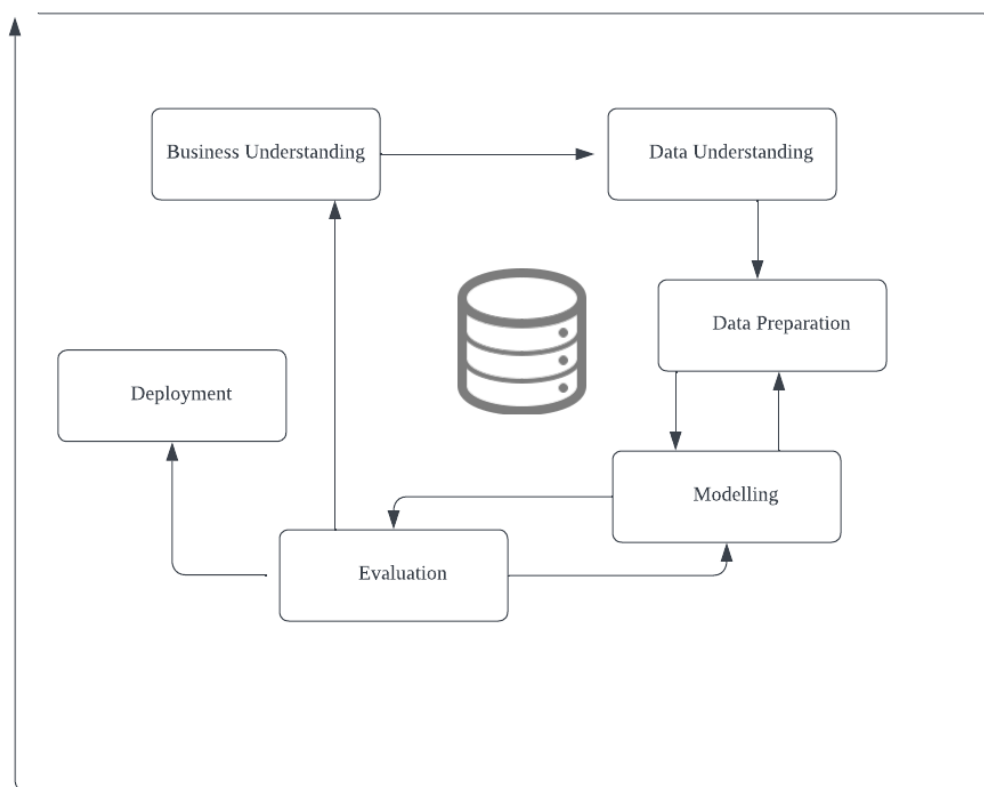


Figure 3.1.1 CRISP-DM Methodology

### 3.2 Description of CRISP-DM Phases

#### Business Understanding

The main goal of this project is to create a hate speech detection tool that is both accessible and user-friendly, catering to a diverse audience, including social media users, content moderators, educators, and organizations. Hate speech refers to any form of communication that demeans an individual or group based on attributes such as race, color, ethnicity, gender, sexual orientation, nationality, religion, or other traits, and has become an increasing concern on social

media platforms. The spread of hate speech can lead to serious consequences, such as psychological harm, social division, and, in severe cases, incitement to violence.

**Context and Need for the Tool:** With billions of users worldwide, social media platforms offer spaces for connection and free expression, but they also serve as hotspots for hate speech. Traditional moderation techniques, which often rely on manual review, cannot cope with the vast amount of content generated each day. Automated detection tools are therefore essential to maintain a safer and more positive online environment. However, many existing tools are either too complicated for the general public to use or lack the transparency necessary to build trust in their results.

### **Goals and Success Criteria:**

To tackle these issues, the project aims to develop a hate speech detection tool that:

- **Is accessible and easy to use:** The tool will be designed with a user-friendly interface that requires minimal technical knowledge, allowing people from different backgrounds to use it effectively.
- **Provides real-time detection:** The tool will use advanced machine learning models to deliver quick feedback on the likelihood of hate speech in text inputs.
- **Ensures model transparency and interpretability:** To build trust in its decisions, the tool will incorporate SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) to help users understand why specific content is flagged.
- **Achieves a validation accuracy of at least 0.8:** The tool's machine learning models will be trained to reach a minimum validation accuracy of 0.8, ensuring consistent and reliable performance in distinguishing hate speech from non-hate speech.

### **Data Understanding**

In this phase, we perform an initial analysis of the collected dataset to assess its quality and relevance for the project. The dataset consists of social media posts labeled as hate speech or non-hate speech. To ensure the data is suitable for building effective models, several key steps are undertaken:

- i. **Duplicate Data Analysis:** The dataset is checked for duplicate entries. Duplicate records are identified and removed to maintain the independence of each data point and ensure the model is not biased by repeated information.

- ii. **Missing Values Handling:** The dataset is examined for any missing values. Missing data points can affect model performance and lead to inaccurate predictions, so they are addressed by either removing them or applying appropriate imputation techniques.
- iii. **Label Distribution Analysis:** The distribution of labels (i.e., hate speech or non-hate speech) is analyzed to understand the balance of the dataset. A skewed distribution can result in biased models, so any imbalance will be identified and considered for potential data augmentation or weighting adjustments during the modeling phase.
- iv. **Text Length Analysis:** The length of each text entry is analyzed to check for any single-character texts (e.g., "c," "a") that may have been incorrectly labeled as hate speech (label 1). Texts of insufficient length are likely to lack meaningful content and could negatively impact the model's ability to learn and generalize. Such entries will be examined and appropriately handled to improve data quality.
- v. **Text Content Visualization:** Data visualization techniques, such as word clouds, are used to gain insights into the most frequently occurring words and phrases within the dataset. This helps identify common themes, patterns, or potential biases in the data. Word clouds provide a visual representation of word frequency, allowing for a quick overview of prominent words and their relevance to hate speech detection.

## **Data Preparation**

The data preparation phase involves cleaning, transforming, and formatting the raw data to make it suitable for effective machine learning and deep learning modeling. This phase includes several critical steps to ensure the dataset's quality and relevance for training the models.

- i. **Removing Duplicates:** Duplicate entries are removed to maintain the integrity of the dataset. Retaining only unique records ensures that the models do not learn redundant patterns, which could lead to overfitting or biased predictions.
- ii. **Removing Non-ASCII Characters:** Any non-ASCII characters present in the dataset are removed. This step ensures that the text data is compatible with various processing tools and libraries that might not support extended character sets, and also helps to standardize the input text for consistent analysis.

- iii. **Filtering Text Length:** Texts shorter than 5 characters are removed, as they are unlikely to contain meaningful content that contributes to hate speech detection. For texts longer than 20 characters, an additional check is performed to ensure that the content is in English. Non-English texts are filtered out to maintain consistency, as the models are specifically trained to detect hate speech in English.
- iv. **Removing Stopwords:** Common stopwords (such as "is," "and," "the") that do not contribute to the meaning or sentiment of the text are removed. This reduces the noise in the data and allows the model to focus on more relevant words that are likely to be indicators of hate speech.
- v. **Lemmatization:** The text is lemmatized to reduce words to their base or root form. For example, "running" becomes "run." This step helps standardize the text data by consolidating different forms of a word, thus improving the model's ability to generalize across variations.
- vi. **Tokenization:** The cleaned text is tokenized, breaking it down into individual words or tokens. This step prepares the text for further processing and numerical conversion, allowing it to be effectively utilized by machine learning algorithms.
- vii. **Text-to-Numeric Conversion:** Different methods are employed to convert the textual data into numerical representations:
  - a. **Machine Learning Approach:** For traditional machine learning models, CountVectorizer and TFIDF (Term Frequency-Inverse Document Frequency) are used to transform the text into vectors. These methods provide a structured numerical representation of the text data, capturing word frequency and importance.
  - b. **Deep Learning Approach:** For deep learning models, advanced word embedding techniques such as Word2Vec and BERT (Bidirectional Encoder Representations from Transformers) are used to represent words in a dense vector space that captures contextual meaning and relationships. If Word2Vec or BERT is not used, the tokenized text is directly fed into the first Embedding Layer of the neural network, which learns the word representations during the training process.

## Modeling

The modeling phase involves selecting and training various machine learning and deep learning algorithms to detect hate speech in social media text effectively. Given the nature of this project as a classification task, both traditional machine learning algorithms and advanced deep

learning neural networks are employed to explore different approaches to achieving optimal performance.

i. **Machine Learning Algorithms:**

Since this is a binary classification task, several machine learning algorithms are selected for their effectiveness in handling text data and classification problems:

- a. **Logistic Regression:** A linear model that is widely used for binary classification problems. Logistic Regression is chosen for its simplicity and interpretability, as well as its effectiveness in scenarios with a clear decision boundary.
- b. **Naive Bayes:** A probabilistic classifier based on Bayes' theorem, particularly suitable for text classification tasks due to its assumption of feature independence. It is efficient and often performs well with large datasets.
- c. **Decision Tree:** A non-linear model that splits the data into branches based on feature values. It is chosen for its ability to handle complex decision boundaries and its interpretability. Decision trees can capture non-linear patterns in the data, making them useful for this classification task.

ii. **Deep Learning Neural Networks:**

In addition to traditional machine learning models, deep learning techniques are employed to capture complex patterns in text data and better understand the nuances of language in the context of hate speech detection. The following neural network architectures are utilized:

- a. **LSTM (Long Short-Term Memory):** An advanced form of recurrent neural network (RNN) that is capable of learning long-term dependencies in sequential data. LSTM is chosen for its ability to capture contextual information and the order of words, which is critical for understanding the semantics of text in hate speech detection.
- b. **BiLSTM (Bidirectional Long Short-Term Memory):** An extension of LSTM that processes the input sequence in both forward and backward directions. This bidirectional approach allows the model to capture context from both directions, enhancing its ability to understand the text comprehensively.

- c. **CNN\_LSTM (Convolutional Neural Network with LSTM):** A hybrid architecture that combines the strengths of CNNs (Convolutional Neural Networks) and LSTMs. The CNN layers are used to extract local features from the text (such as phrases and n-grams), which are then fed into LSTM layers to capture the sequential dependencies and context. This combination is particularly useful for capturing both local and global features in the text.

By employing a combination of machine learning and deep learning models, the project aims to explore multiple approaches to hate speech detection, comparing their performance and selecting the most effective model for deployment.

## Evaluation

The evaluation phase is dedicated to measuring the performance of the models developed in the preceding phase. A variety of metrics are used to thoroughly assess the models' ability to detect hate speech:

- **Accuracy:** Represents the ratio of correctly predicted instances (both hate speech and non-hate speech) to the total instances in the dataset. Although accuracy provides an overall view of model performance, it might not accurately reflect the model's effectiveness in detecting hate speech, particularly when the dataset is imbalanced.
- **Recall (Sensitivity or True Positive Rate):** Indicates the model's capability to correctly identify all instances of hate speech. A high recall suggests that the model successfully detects most cases of hate speech, thereby reducing the number of false negatives. This metric is particularly important in situations where failing to identify hate speech can have serious consequences.
- **Precision:** Measures the proportion of instances that are actually hate speech among those predicted as hate speech. A high precision score indicates that the model is usually correct when it identifies content as hate speech, thereby minimizing false positives. This is essential to avoid incorrectly flagging non-hate speech content.
- **F1-Score:** The harmonic mean of precision and recall, providing a balanced metric that accounts for both false positives and false negatives. The F1-score is especially useful in cases where the dataset is imbalanced, or when both types of errors have significant consequences.



Beyond these performance metrics, SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) are used to interpret and explain the decision-making process of the model:

- **SHAP:** Offers insights into how much each feature contributes to a specific prediction by assigning Shapley values. These values explain the impact of individual features on the model's output, helping to understand how various words or phrases influence the classification of text as hate speech or non-hate speech.
- **LIME:** Builds locally interpretable models around each prediction to clarify the decisions made by complex models, such as deep learning networks. By providing understandable explanations for individual predictions, LIME helps to determine if the model's reasoning aligns with human intuition and understanding.

By combining these quantitative evaluation metrics with interpretability tools, the evaluation phase ensures not only that the models achieve high performance but also that their decision-making processes are transparent, understandable, and trustworthy to users.

## Deployment

The deployment phase involves integrating the developed hate speech detection models into a user-facing application and making them available for use in real-world scenarios. This phase ensures that the models are accessible, functional, and reliable in practical settings. Key aspects of the deployment phase include:

- i. **Deployment Environment:** The hate speech detection system is deployed on a web-based platform using Streamlit. This choice allows for easy accessibility and user interaction through a web interface, which is accessible from various devices and platforms.
- ii. **Integration:** The trained models are integrated into the Streamlit application, allowing users to input text and receive real-time feedback on hate speech detection. This involves setting up the necessary backend infrastructure to handle user requests, run the models, and return predictions.
- iii. **Testing and Validation:** Prior to full deployment, the system undergoes extensive testing to ensure it operates correctly in the production environment. This includes unit testing of individual components, integration testing to ensure all parts work together seamlessly, and user acceptance testing to verify that the system meets the needs and expectations of its users.

## **Iteration and Refinement**

The CRISP-DM methodology is inherently iterative and flexible, allowing for continuous refinement and improvement throughout the data mining process. After completing the Evaluation phase, it is often necessary to revisit earlier phases to enhance the overall effectiveness of the project. This iterative approach ensures that the models and data preparation processes remain aligned with the project's objectives and performance goals.

### **Iteration and Refinement Process:**

#### **i. Revisiting Data Understanding:**

- Based on the results from the Evaluation phase, the understanding of the data may need to be revisited. This could involve further analysis of the data distribution, detecting new patterns or anomalies, or addressing issues that were initially overlooked. Enhanced data insights can lead to improved data preparation and more effective feature engineering.

#### **ii. Adjusting Data Preparation:**

- The data preparation steps may require adjustments based on feedback from the Evaluation phase. For example, if certain features are found to be less relevant, they may be refined or replaced. New preprocessing techniques or feature engineering methods might be introduced to better capture the characteristics of the data.

#### **iii. Exploring Different Algorithms:**

- The Evaluation phase may reveal that certain algorithms perform better or worse than anticipated. In response, it may be beneficial to experiment with alternative algorithms or models. This could involve trying different machine learning techniques, adjusting hyperparameters, or exploring new deep learning architectures to achieve better performance.

#### **iv. Re-evaluating Model Performance:**

- After making changes to the data or models, a new round of evaluation is conducted to assess the impact of these adjustments. This helps ensure that the improvements align with the project's performance criteria and goals.

#### **v. Continuous Improvement:**

- The iterative process supports ongoing refinement and enhancement of the hate speech detection system. By continuously cycling through these phases, the project can adapt to new insights, address emerging challenges, and achieve higher levels of accuracy and reliability.

In summary, the CRISP-DM methodology supports a dynamic and iterative approach, enabling the project to evolve and improve through repeated cycles of data understanding, preparation, modeling, and evaluation. This iterative refinement process is crucial for developing a robust and effective hate speech detection system.

# Chapter 4

## System Design

### 4.1 System Block Diagram

Figure 4.1.1 provides a detailed overview training process of the machine learning approach used in this project. The diagram visually represents the complete workflow from data collection through model training and evaluation to the creation of a user interface. Each component of the process is described below:

#### 1. Data Collection:

- **Source:** Kaggle dataset.
- **Description:** The dataset used for training and evaluating the models is sourced from Kaggle. It contains labelled social media posts classified as hate speech or non-hate speech.

#### 2. Data Preprocessing:

- Prepare the raw data for model training by cleaning and transforming it into a suitable format.
- **Clean Data:**
  - **Drop Duplicated Entries:** Identifies and removes duplicate records to ensure that each data point is unique. This step is crucial for avoiding biased model training.
  - **Handle Missing Values:** Detects and addresses missing values using appropriate techniques, such as imputation or removal, to prevent skewed results.
  - **Filter Out Short Sentences:** Removes sentences shorter than 5 characters, as they are unlikely to provide meaningful information for hate speech detection.
- **Text Processing:**
  - **Remove Stopwords:** Common words like "is," "and," "the" are removed because they do not contribute significant meaning to the text and can add noise to the model.
  - **Lemmatization:** Reduces words to their base or root form (e.g., "running" to "run"), which helps in normalizing the text data and improving model consistency.

- **Tokenization:**
  - **Description:** Breaks down the text into individual words or tokens, preparing it for numerical representation.
  - **Purpose:** Tokenization is essential for transforming text into a format that can be processed by machine learning algorithms.
- **Vectorization:**
  - **CountVectorizer:**
    - Converts tokens into a numerical matrix where each row represents a document, and each column represents a word's frequency in the document.
  - **TFIDF (Term Frequency-Inverse Document Frequency):**
    - Converts tokens into a numerical matrix where each row represents a document, and each column represents the importance of a word in the document relative to its frequency in the entire dataset.

### 3. Model Training:

- Train various machine learning models using the vectorized data to classify text as hate speech or non-hate speech.
- **Logistic Regression:**
  - A linear model used for binary classification tasks. It is trained to predict the probability of text being hate speech based on the vectorized input.
- **Naive Bayes:**
  - A probabilistic classifier based on Bayes' theorem, assuming feature independence. It is trained to estimate the likelihood of a text being hate speech.
- **Decision Tree:**
  - A non-linear model that splits the data into branches based on feature values. It is trained to create a decision tree that classifies text into hate speech or non-hate speech.

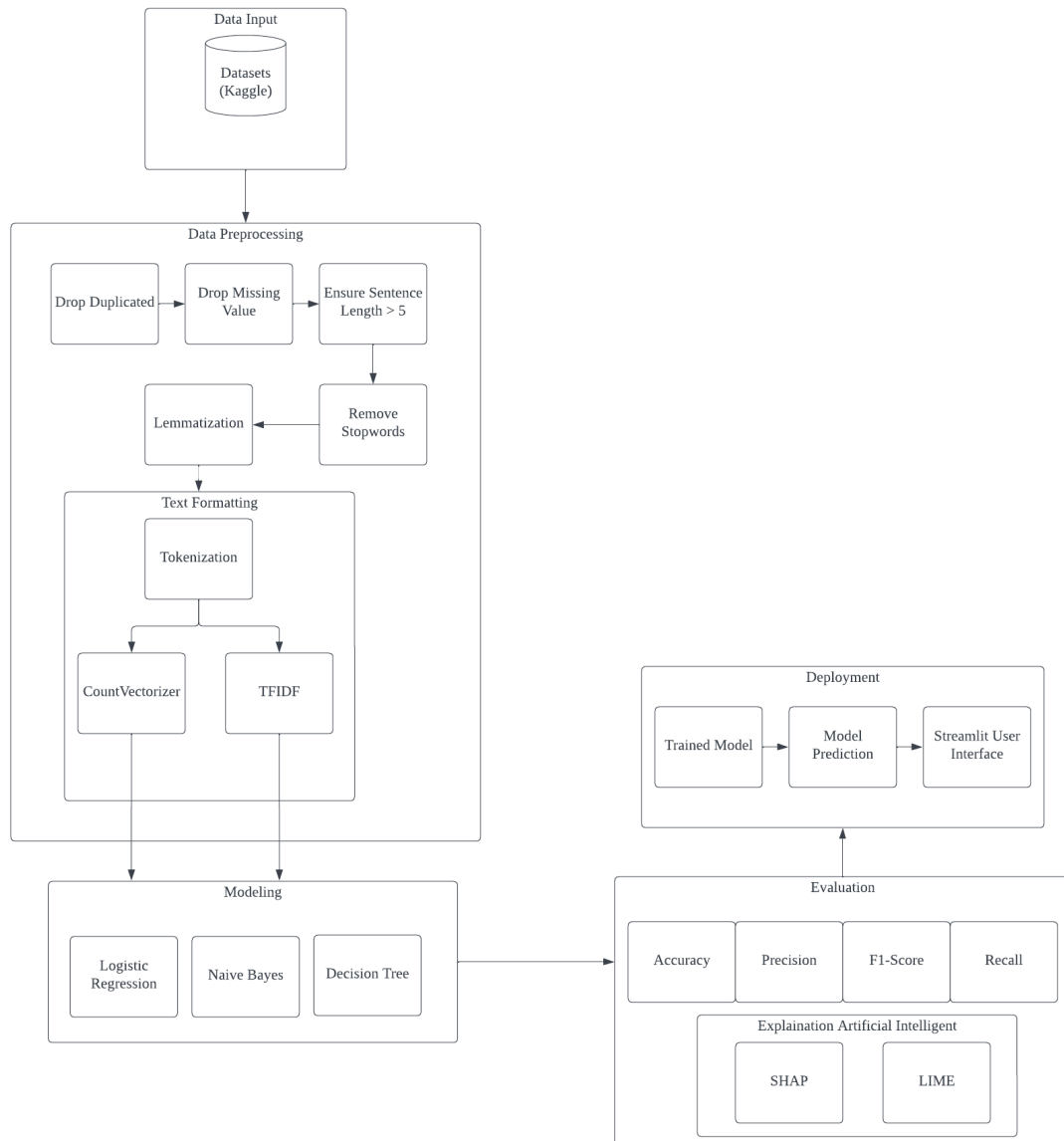
### 4. Model Evaluation:

- Assess the performance of each trained model using various metrics.
- **Confusion Metrics:**

- **Description:** Evaluates the model's performance using accuracy, precision, recall, and F1-score. These metrics provide insights into how well the model distinguishes between hate speech and non-hate speech.
- **SHAP and LIME:**
  - **SHAP (SHapley Additive exPlanations):**
    - **Description:** Provides explanations for model predictions by calculating Shapley values, which attribute the contribution of each feature to the final prediction.
  - **LIME (Local Interpretable Model-agnostic Explanations):**
    - **Description:** Creates interpretable explanations for individual predictions by generating locally approximated models around each prediction.

## 5. User Interface:

- Develop a user-friendly interface to facilitate the use of the hate speech detection models.
- **Streamlit:**
  - A web-based application framework used to create an interactive and intuitive user interface. Users can input text and receive real-time feedback on the likelihood of hate speech.
  - Makes the models accessible and easy to use for individuals without technical expertise.



*Figure 4.1.1 Training Process for Machine Learning Approach*

Figure 4.1.2 provides a detailed overview training process of the deep learning approach used in this project. The diagram visually represents the complete workflow from data collection through model training and evaluation to the creation of a user interface. Each component of the process is described below:

### 1. Dataset Collection

- **Source:** Downloaded from Kaggle.
- **Description:** A collection of social media posts labeled as either hate speech or non-hate speech.

## 2. Data Preprocessing

- **Data Cleaning:**
  - i. **Remove Duplicates:** Identify and eliminate duplicate entries to ensure each data point is unique.
  - ii. **Handle Missing Values:** Detect and address any missing values by either removing records with missing data or applying imputation techniques.
  - iii. **Filter Out Short Sentences:** Remove text entries with fewer than 5 characters to eliminate entries that are likely not meaningful for analysis.
- **Text Processing:**
  - i. **Remove Stopwords:** Filter out common, non-informative words (e.g., "the", "is") that do not contribute to the meaning of the text.
  - ii. **Lemmatization:** Normalize words to their base or root form (e.g., "running" to "run") to standardize text data and improve model consistency.
  - iii. **Tokenization:** Break down text into individual tokens (words or phrases) to prepare it for vectorization.
- **Word Embedding:**
  - i. **Word2Vec:** Convert tokens into dense vector representations based on their context within the text. These vectors capture semantic meanings and relationships between words.
  - ii. **BERT:** Generates contextual embeddings for tokens using the BERT model. BERT produces vectors that reflect the context of words within a sentence, providing rich and nuanced representations.
  - iii. **Embedding Layer:** Converts tokens directly into embeddings if neither Word2Vec nor BERT is used. This layer learns the representation of words during training.

## 3. Modeling

- **Word2Vec + Double LSTM:** A two-layer LSTM network that processes the vectors obtained from Word2Vec. This setup helps capture long-term dependencies and contextual information in the text.



- BERT + BiLSTM: A Bidirectional LSTM network that processes the contextual embeddings from BERT. This model reads text in both forward and backward directions to enhance context understanding.
- Embedding Layer + Deep Learning Models:
  - i. LSTM: A single-layer LSTM network that processes embeddings generated by the embedding layer. This model captures sequential dependencies and contextual patterns in the text.
  - ii. CNN\_LSTM: A hybrid model combining Convolutional Neural Network (CNN) layers with LSTM layers. CNN layers extract local features from the text, while LSTM layers capture sequential dependencies and context.

#### **4. Evaluation**

- Confusion Matrix: Evaluate model performance using metrics such as accuracy, precision, recall, and F1-score.
  - i. Accuracy: Proportion of correctly classified instances out of the total instances.
  - ii. Precision: Proportion of true positives out of all instances predicted as positive.
  - iii. Recall: Proportion of true positives out of all actual positives.
  - iv. F1-Score: Harmonic mean of precision and recall, providing a balanced measure of model performance.
- Explainability:
  - i. SHAP: Provides insights into how each feature (word or phrase) contributes to the model's prediction.
  - ii. LIME: Creates interpretable models around individual predictions to explain the decisions made by complex models.

#### **5. Deployment**

- User Interface: Develop a user interface that allows users to interact with the trained models, input text, and receive predictions about hate speech. The interface provides a practical tool for end-users to leverage the model's capabilities.

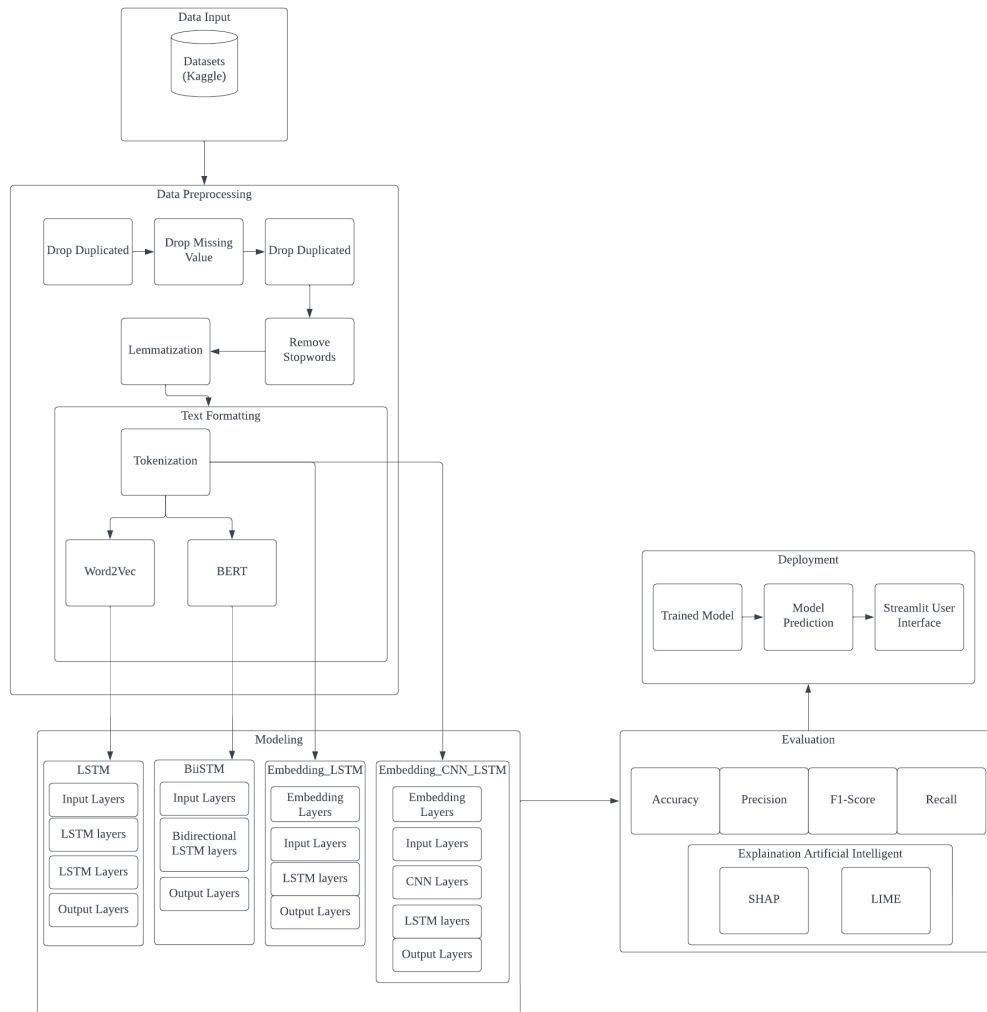


Figure 4.1.2 Training Process for Deep Learning Approach

Figure 4.1.3 is the use case diagram illustrates the various functionalities of the hate speech detection system built using Streamlit, along with the relationships between these functionalities. The system enables users to select a model, choose an explainer, input text, and generate predictions with explanations.

### Use Cases and Their Relationships:

#### 1. Model Selection:

- The user can select a model from the list of trained models available in the system.
- This use case **includes** the Prepare Model use case, indicating that once a model is selected, the system automatically proceeds to prepare the model for prediction.

#### 2. Prepare Model (Included Use Case):

- This use case is automatically invoked when the user selects a model. It involves loading the model's parameters, initializing the necessary libraries, and ensuring that the model is ready for use.
3. **Explainer Selection:**
- The user selects an explainer tool, such as SHAP or LIME, to interpret the model's predictions.
  - This use case **includes** the Prepare Explainer use case, which prepares the selected explainer for interpreting predictions.
4. **Prepare Explainer (Included Use Case):**
- Once an explainer is selected, the system initializes the explainer, loads relevant libraries, and sets up the necessary configuration to use the explainer with the chosen model.
5. **Enter Text:**
- The user inputs a text snippet to analyze for potential hate speech. This is the initial step in the prediction process.
6. **Predict Button:**
- After entering the text, the user clicks the 'Predict' button to start the analysis.
  - This use case **includes** several subsequent use cases that are part of the prediction workflow:
    - **Check Text:** The system checks the input text for errors.
      - This use case **extends** to Text Error if any errors are detected, providing feedback to the user to correct the input.
    - **Text Preprocessing:** If no errors are found, the text is preprocessed, which involves tokenization, removal of stop words, lemmatization, and vectorization.
    - **Model Prediction:** The preprocessed text is fed into the selected model to generate a prediction.
    - **Explain Prediction:** The system uses the chosen explainer tool to interpret the model's output and provides insights on which words or phrases influenced the decision.
7. **Check Text (Included Use Case):**

- This use case is invoked by the Predict Button and checks the input text for errors, such as invalid characters or inappropriate length.
  - It **extends** to Text Error when errors are found.
8. **Text Error** (Extended Use Case):
- If the system detects errors during the Check Text process, it provides feedback to the user, suggesting corrections for the input text.
9. **Text Preprocessing** (Included Use Case):
- Prepares the text for model prediction by converting it into a suitable format.
10. **Model Prediction** (Included Use Case):
- Takes the preprocessed text and generates a prediction using the selected model.
11. **Explain Prediction** (Included Use Case):
- Uses the selected explainer to provide an interpretation of the model's prediction, offering insights into the decision-making process.

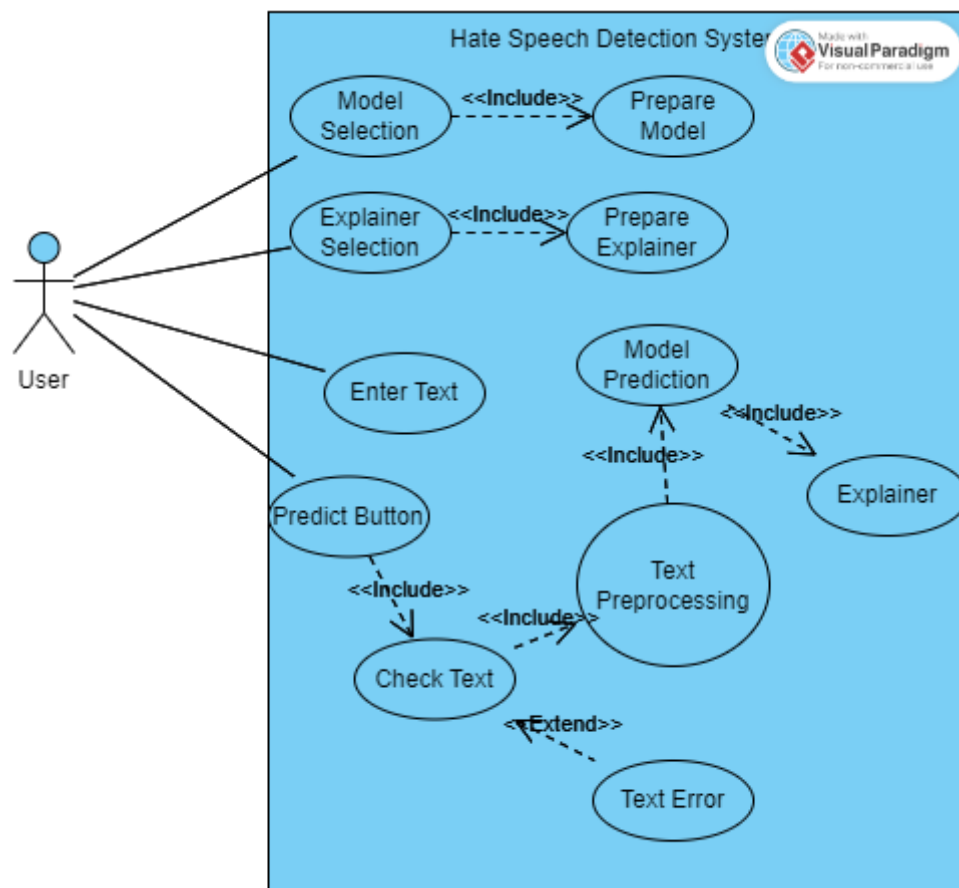


Figure 4.1.3 Hate Speech Detection System Use Case Diagram

## 4.2 Use Case Description

<b>Use Case ID</b>	UC-01
<b>Use Case Name</b>	Model Selection
<b>Actor</b>	User
<b>Preconditions</b>	The system is running, and the user has accessed the model selection interface.
<b>Postconditions</b>	The selected model is prepared and loaded for use.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The user selects a desired model from the list of available models.</li> <li>2. The system retrieves the selected model.</li> <li>3. The system prepares the model for use.</li> </ol>
<b>Alternate Flow</b>	The default model will be selected
<b>Include</b>	Prepare Model

*Table 4.2.1 Model Selection Use Case Description*

<b>Use Case ID</b>	UC-02
<b>Use Case Name</b>	Explainer Selection
<b>Actor</b>	User
<b>Preconditions</b>	The system is running, and the user has accessed the explainer selection interface.
<b>Postconditions</b>	The selected explainer is prepared and loaded for use.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The user selects an explainer (SHAP or LIME).</li> <li>2. The system retrieves the selected explainer.</li> <li>3. The system prepares the explainer for use.</li> </ol>
<b>Alternate Flow</b>	The default explainer will be selected
<b>Include</b>	Prepare Explainer

*Table 4.2.2 Explainer Selection Use Case Description*

<b>Use Case ID</b>	UC-03
<b>Use Case Name</b>	Enter Text
<b>Actor</b>	User
<b>Preconditions</b>	The system is running, and the user is on the text input interface.
<b>Postconditions</b>	The text is entered and ready for further processing.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The user inputs text into the designated field.</li> <li>2. The system accepts and displays the entered text.</li> </ol>

*Table 4.2.3 Enter Text Use Case Description*

<b>Use Case ID</b>	UC-04
<b>Use Case Name</b>	Predict Button
<b>Actor</b>	User
<b>Preconditions</b>	The system is running, the user has selected a model and explainer, and text has been entered.
<b>Postconditions</b>	The system provides a prediction and an explanation for the input text.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The user clicks the 'Predict' button.</li> <li>2. The system checks the entered text for errors (Include: Check Text).</li> <li>3. If no errors, the system preprocesses the text (Include: Text Preprocessing).</li> <li>4. The system uses the selected model to make a prediction (Include: Model Prediction).</li> <li>5. The system uses the selected explainer to provide an explanation for the prediction (Include: Explainer).</li> </ol>
<b>Alternate Flow</b>	If text errors are found, the system displays an error message (Extend: Text Error).

*Table 4.2.4 Predict Button Use Case Description*

<b>Use Case ID</b>	UC-05
<b>Use Case Name</b>	Check Text
<b>Actor</b>	System
<b>Preconditions</b>	The user has clicked the 'Predict' button, and the system checks whether text has been entered.
<b>Postconditions</b>	The text is either accepted for preprocessing or an error is reported if no text or invalid input is detected.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The system verifies that the user has entered text.</li> <li>2. The system checks the entered text for errors, such as inappropriate characters or overly short text.</li> <li>3. If the text is valid, the system proceeds to text preprocessing.</li> </ol>
<b>Extend</b>	Text Error (if no text is entered or errors are found in the text).

*Table 4.2.5 Check Text Use Case Description*

<b>Use Case ID</b>	UC-06
<b>Use Case Name</b>	Text Preprocessing
<b>Actor</b>	System
<b>Preconditions</b>	Text has been checked and confirmed to be error-free.
<b>Postconditions</b>	The text is tokenized and vectorized, ready for model prediction.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The system removes stopwords and applies lemmatization.</li> <li>2. The system tokenizes and vectorizes the text using the chosen method (e.g., CountVectorizer, TFIDF, Word2Vec, BERT).</li> </ol>

*Table 4.2.6 Text Preprocessing Use Case Description*

<b>Use Case ID</b>	UC-07
<b>Use Case Name</b>	Model Prediction
<b>Actor</b>	System
<b>Preconditions</b>	Preprocessed text is available.
<b>Postconditions</b>	The system generates a prediction for the entered text.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The system uses the selected model to predict whether the input text contains hate speech.</li> <li>2. The prediction result is generated.</li> </ol>

*Table 4.2.7 Model Prediction Use Case Description*

<b>Use Case ID</b>	UC-08
<b>Use Case Name</b>	Explainer
<b>Actor</b>	System
<b>Preconditions</b>	Model prediction is complete.
<b>Postconditions</b>	The system provides an explanation for the prediction.
<b>Main Flow</b>	<ol style="list-style-type: none"> <li>1. The system uses the selected explainer (SHAP or LIME) to interpret the prediction.</li> <li>2. The explanation is generated and displayed to the user.</li> </ol>

*Table 4.2.8 Explainer Use Case Description*



### 4.3 System Components Specifications

Specifications	Local Computer	Cloud Google Colab
<b>Hardware Specification:</b>		
Processor	Intel i5-10300H CPU@ (2.50GHz)	-
RAM	16GB	TPU: 334.6 GB T4 GPU: 12.7 GB
Storage	465 GB HDD	-
GPU/TPU	NVIDIA GTX 1650	T4 GPU, TPU v2
<b>Software Specification:</b>		
Operating System	Windows 11	Linux
Programming Language	Python 3.12.4	
Libraries and Frameworks	Data Manipulation: NumPy, Pandas Data Visualization: Matplotlib, Seaborn Deep Learning: TensorFlow, Keras Machine Learning: Scikit-learn Natural Language Processing: NLTK (Natural Language Toolkit) Feature Extraction: Keras Tokenizer, Scikit-learn's TfidfVectorizer Interpretability: LIME (Local Interpretable Model-agnostic Explanations) Other Tools: WordCloud (for visual representation)	
Development Environment	Jupyter Notebook Visual Studio Code	Google Colab

Table 4.3.1 System Components Specifications

# Chapter 5

## System Implementation

### 5.1 Implementation of CRISP-DM Methodology

#### 1. Business Understanding

In this project, the business understanding phase focused on defining clear objectives and success criteria to guide the development and design of a hate speech detection tool. By thoroughly analyzing the complexities of hate speech on social media and understanding user needs, the following goals were established:

- **Goal 1: Create an Accessible Hate Speech Detection Tool**

To ensure that the tool is easy to use for all individuals, regardless of their technical background, the project utilized the Streamlit framework to build an intuitive user interface. The design includes simple features such as model selection and text input, allowing users to quickly get started and input text for analysis without needing complex operations.

- **Goal 2: Provide Real-Time Detection**

To meet the need for real-time detection, the project employed various machine learning and deep learning models to efficiently process user input. Using a combination of optimized models such as Logistic Regression, Naive Bayes, Decision Trees, and LSTM networks, the system can quickly provide feedback on whether the input contains hate speech.

- **Goal 3: Ensure Model Transparency and Interpretability**

Trust and understanding of the model are crucial to the success of the tool. Therefore, the project integrated SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) tools. These tools help explain why each text input is classified as hate speech or non-hate speech, providing users with insights into the model's decision-making process and increasing transparency and interpretability.

- **Goal 4: Achieve a Minimum Validation Accuracy of 0.8**

The project set a minimum validation accuracy benchmark of 0.8. To achieve this, multiple strategies were employed in data preprocessing, feature extraction, and model selection. These included data cleaning, noise removal, using various feature extraction methods like CountVectorizer, TF-IDF, Word2Vec, and BERT, and iterative training and optimization of different models.

By setting and implementing these goals, the business understanding phase laid the foundation for the subsequent system implementation, ensuring that the hate speech detection tool meets practical requirements while being efficient, transparent, and user-friendly.

## 2. Data Understanding + Data Preparation

### i. Initial Data Understanding

The original dataset used for this project consisted of two columns: "Content" (textual data) and "Label" (binary value: 0 or 1). The initial exploration of the dataset revealed the following:

- **Total Records:** 726,119
- **Duplicated Values:** 25,046 entries were identified as duplicates.
- **Null Values:** No null values were found in the dataset.

The distribution of labels in the initial dataset was:

- **Label 1 (Hate Speech):** 364,525 entries
- **Label 0 (Non-Hate Speech):** 361,594 entries

This shows a fairly balanced dataset, with a slight majority of entries labeled as hate speech.

### ii. Initial Data Preprocessing

To clean and standardize the dataset, several preprocessing steps were performed:

- a. **Remove Duplicate Entries:** Duplicates were removed based on the "Content" column to ensure each piece of text is unique.
- b. **Text Cleaning:** The `clean_text` function was applied to normalize the text and remove unwanted characters, including:
  - Converting Unicode characters to ASCII.
  - Keeping only letters, numbers, spaces, and some punctuation.

- Reducing repeated characters while preserving at least three (to avoid removing emphasis).

```
def clean_text(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('ascii')
    text = re.sub(r'^\w\s?!.,]', '', text)
    text = re.sub(r'(\w)\1{3,}', r'\1\1\1', text)
    return text.strip()
```

*Figure 5.1.2.2.1 clean\_text function*

- Remove Short Texts:** Texts shorter than 5 characters were removed to avoid noise in the data.
- Language Detection:** Only English texts were retained, based on language detection using the langdetect library. Shorter texts (less than 20 characters) were kept without a language check to avoid misclassification due to lack of context.
- Save Cleaned Dataset:** The cleaned dataset was saved as cleaned\_dataset.csv for future use in training and evaluation.

### iii. New Data Understanding for Cleaned Dataset

After preprocessing, the cleaned dataset was re-evaluated:

- **Duplicated Values:** New duplicates were identified, likely due to text normalization and cleaning steps that caused different original texts to become identical after cleaning. These were subsequently removed.
- **Null Values:** No null values were present in the cleaned dataset.

The updated distribution of labels in the cleaned dataset was:

- **Label 1 (Hate Speech):** 335,703 entries (50.3%)
- **Label 0 (Non-Hate Speech):** 331,051 entries (49.7%)

The dataset remained balanced, which is favorable for training models.

### iv. Frequency Analysis of Cleaned Data

A frequency analysis was conducted to identify the most common words in the dataset, separated by hate speech and non-hate speech categories:

- **Top 5 Words in All Comments:** ['slut', 'article', 'like', 'page', 'would']
- **Top 5 Words in Hate Speech Comments:** ['like', 'people', 'fuck', 'fucking', 'get']
- **Top 5 Words in Non-Hate Speech Comments:** ['article', 'slut', 'page', 'would', 'wikipedia']





- **CountVectorizer:** This method converts the text into a matrix of token counts, capturing the frequency of each word within the dataset.
- **TF-IDF (Term Frequency-Inverse Document Frequency):** This technique measures the importance of a word in a document relative to the entire dataset, reducing the weight of commonly used words while highlighting more distinctive words.

Only the training set was fit-transformed using these vectorizers, ensuring that the test set was not influenced by the training data. The test set was subsequently transformed using the vectorizers fitted on the training data.

### **Deep Learning Approach**

For deep learning models, the preparation process differed depending on the type of word embedding technique used. Three primary approaches were implemented:

#### **i. Word2Vec Embedding:**

- **Tokenization:** The cleaned text data was tokenized to convert words into sequences of integers, where each integer represents a unique word in the vocabulary.
- **Word2Vec Vectorization:** Using the Word2Vec algorithm, words were embedded into a dense vector space where semantically similar words have closer vector representations.
- **Padding Sequences:** To handle varying text lengths, the sequences were padded to a fixed length using `pad_sequences` to ensure uniform input size for the model.
- **Train-Test Split:** The dataset was split into training and test sets post-vectorization, maintaining the integrity of the embeddings.

#### **ii. BERT Embedding:**

- **BERT Tokenizer:** Text data was tokenized using the BERT tokenizer, which handles sub-word tokenization and provides special tokens for sentence classification tasks.
- **Padding within Tokenizer:** During tokenization, padding was applied to ensure all sequences were of the same length, as required by BERT.
- **BERT Embedding Extraction:** The tokenized data was passed through the BERT model to obtain the `last_hidden_state` — a representation of

the input that encodes contextual information for each word. This state was used as input features for subsequent layers of the deep learning model.

### iii. **Neural Network Embedding Layer:**

- **Tokenization:** Similar to the Word2Vec approach, the text was tokenized into sequences of integers.
- **Padding Sequences:** The tokenized data was padded using pad\_sequences to ensure uniform sequence lengths.
- **Train-Test Split:** The dataset was split into training and test sets before feeding the tokenized data into the neural network's Embedding Layer. This layer learns word embeddings during the model training process, which are optimized for the specific task of hate speech detection.

By employing these various data preparation techniques, the dataset was made ready for both machine learning and deep learning models, ensuring that each model had appropriately formatted inputs for optimal performance.

## **3. Modeling**

The modeling phase involved developing and training a variety of machine learning and deep learning models to detect hate speech in social media text. This section provides a detailed overview of the different approaches and their specific configurations, ensuring a robust evaluation of model performance.

### **i. Machine Learning Approach**

For the machine learning models, two different text vectorization techniques — **CountVectorizer** and **TF-IDF** — were used to represent the text data. These vectorized datasets were then used to train three different machine learning algorithms:

#### **a. Logistic Regression:**

- Logistic Regression is a popular algorithm for binary classification tasks due to its simplicity and effectiveness. It was configured using the following parameters:
  - **Penalty:** 'l2' (Ridge Regularization)
  - **Inverse Regularization Strength (C):** [0.01, 0.1, 1]



- To ensure the robustness of the model, **GridSearchCV** with **5-fold cross-validation** (cv=5) was applied to identify the optimal hyperparameters. The model was evaluated using **accuracy** as the scoring metric.
- b. **Naive Bayes:**
- The Naive Bayes classifier, specifically the Multinomial Naive Bayes variant, is suitable for text classification tasks due to its ability to handle high-dimensional data efficiently. Since Naive Bayes has fewer hyperparameters, no specific grid search parameters were set (param\_NB = {}).
  - Similar to Logistic Regression, **GridSearchCV** with **5-fold cross-validation** (cv=5) was utilized to ensure consistent and reliable model performance, using **accuracy** as the scoring metric.
- c. **Decision Tree:**
- Decision Trees were employed to explore the hierarchical structure of decision-making in the text data. The model was configured with the following parameters:
    - **Maximum Depth** (max\_depth): [5, 8]
    - **Minimum Samples Split** (min\_samples\_split): [1, 2, 5]
  - Due to the complexity of the model, **GridSearchCV** with **2-fold cross-validation** (cv=2) was used to determine the optimal parameters, with **accuracy** as the scoring metric.
  - Bagging Classifier: To improve the performance and stability of the Decision Tree model, a BaggingClassifier was used with the following parameters:
    - Estimator: DecisionTreeClassifier
    - Number of Estimators (n\_estimators): 5
    - Random State: 42

By employing GridSearchCV, all models were trained iteratively across different parameter combinations to identify the best-performing configurations.

## ii. Deep Learning Approach

For the deep learning approach, four distinct models were implemented using various word embedding techniques and neural network architectures. Each model was evaluated to determine its effectiveness in detecting hate speech. The following sections provide details on the configurations and parameters for each deep learning model used in this project.

### a. Stacked LSTM with Word2Vec Embeddings

#### Architecture:

- **Input Layer:** Takes Word2Vec embeddings as input vectors.
- **LSTM Layer 1:** 100 units, return\_sequences=True to pass the sequence to the next LSTM layer.
- **LSTM Layer 2:** 100 units, processes the sequence output from the first LSTM layer.
- **Dense Layer:** 1 unit with sigmoid activation to output the prediction.

### b. BiLSTM with BERT Embeddings

#### Architecture:

- **BERT Layer:** Pre-trained BERT model used to obtain contextual embeddings for the input text.
- **BiLSTM Layer:** Bidirectional LSTM with dropout for regularization.
- **Fully Connected Layer:** Linear layer for classification output.

### c. Single-Layer LSTM with Embedding Layer

#### Architecture:

- **Embedding Layer:** Converts input tokens to dense vectors with an output dimension of 130.
- **LSTM Layer:** Includes a single LSTM layer with 100 units and dropout for regularization.
- **Dense Layer:** Output layer with a sigmoid activation function for binary classification.

#### Parameters:

- **Embedding Dimension:** 130
- **LSTM Units:** 100
- **Dropout Rate:** 0.2

#### d. CNN-LSTM with Embedding Layer

##### Architecture:

- **Embedding Layer:** Converts input tokens to dense vectors with an output dimension of 100.
- **CNN Layers:** Uses multiple convolutional filters for feature extraction:
  - **Number of Filters:** 100
  - **Filter Sizes:** 3, 4, 5
- **LSTM Layer:** Processes combined features from CNN layers with 64 units.
- **Dense Layer:** Output layer with a sigmoid activation function for binary classification.

##### Parameters:

- **Embedding Dimension:** 100
- **Number of Filters:** 100
- **Filter Sizes:** 3, 4, 5
- **LSTM Units:** 64
- **Dropout Rate:** 0.3
- **Learning Rate:** 0.0001

#### 4. Evaluation

The evaluation of both machine learning and deep learning models was conducted using four key metrics: **Accuracy**, **Precision**, **Recall**, and **F1-score**. These metrics provide a comprehensive understanding of each model's performance, particularly in the context of binary classification tasks such as hate speech detection.

#### Machine Learning Models

The performance of the machine learning models trained with two different text vectorization techniques, CountVectorizer and TF-IDF, is summarized in Table 5.1.4.1.1 below.

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression (CountVectorizer)	0.84	0.84	0.84	0.84
Naive Bayes (CountVectorizer)	0.80	0.74	0.92	0.82
Decision Tree (CountVectorizer)	0.66	0.61	0.93	0.73
Bagging Decision Tree (CountVectorizer)	0.83	0.82	0.84	0.83
Logistic Regression (TF-IDF)	0.84	0.84	0.84	0.84

Naive Bayes (TF-IDF)	0.82	0.78	0.89	0.83
Decision Tree (TF-IDF)	0.66	0.61	0.93	0.73
Bagging Decision Tree (TF-IDF)	0.83	0.82	0.85	0.83

*Table 5.1.1 Evaluation of Machine Learning Approach*

**Analysis:**

- **Logistic Regression:** Both with CountVectorizer and TF-IDF, Logistic Regression achieved relatively high performance, with accuracies of 0.84 and 0.84, respectively. Precision, recall, and F1-scores were also balanced, indicating its effectiveness in handling the text classification task.
- **Naive Bayes:** The Naive Bayes classifier showed a higher recall (0.92 and 0.89 for CountVectorizer and TF-IDF) but lower precision (0.74 and 0.78). This suggests that the model is more inclined towards correctly identifying the positive class but at the cost of a higher false positive rate.
- **Decision Tree:** The Decision Tree model performed poorly, especially in terms of accuracy (0.66 and 0.66) and precision (0.61 and 0.61). However, its recall was high, which indicates that the model was also biased towards identifying positive cases.
- **Bagging Decision Tree:** By applying the Bagging technique, the Decision Tree's performance improved significantly. The Bagging Decision Tree achieved better accuracy (0.83 and 0.83) and F1-scores (0.83 and 0.83) for both vectorization methods, demonstrating enhanced stability and robustness in model predictions.

Overall, **Logistic Regression** and **Bagging Decision Tree** were the best-performing machine learning models. Logistic Regression maintained a balanced performance across all metrics, while Bagging Decision Tree showed a good trade-off between precision and recall after boosting the base Decision Tree.

**Deep Learning Models**

The performance of the deep learning models with different architectures and embedding methods is summarized in Table 5.1.4.2.1 below.

<b>Model</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>
Word2Vec + Stacked LSTM	0.80	0.79	0.83	0.81
BERT + BiLSTM	0.82	0.83	0.81	0.82
Embedding + CNN-LSTM	0.88	0.87	0.89	0.88
Embedding + Single-Layer LSTM	0.85	0.84	0.87	0.85

*Table 5.1.2 Evaluation of Deep Learning Approach*

#### **Analysis:**

- **Word2Vec + Stacked LSTM:** The model achieved an accuracy of 0.80 and an F1-score of 0.81, indicating that it performed reasonably well in both precision (0.79) and recall (0.83). However, it did not outperform the other deep learning models, especially in terms of accuracy and F1-score.
- **BERT + BiLSTM:** Utilizing BERT embeddings and a bidirectional LSTM, this model achieved an accuracy of 0.82 and an F1-score of 0.82. The precision (0.83) and recall (0.81) scores were balanced, demonstrating its capability to handle more complex contextual information from the text data.
- **Embedding + CNN-LSTM:** This model performed the best among all the deep learning architectures, achieving the highest accuracy (0.88) and F1-score (0.88). The combination of CNN for feature extraction and LSTM for sequence modeling proved highly effective in capturing both local and global dependencies in the text data.
- **Embedding + Single-Layer LSTM:** The Single-Layer LSTM model achieved an accuracy of 0.85 and an F1-score of 0.85. It also maintained a good balance between precision (0.84) and recall (0.87), making it a solid choice for binary classification tasks, though slightly less effective than the CNN-LSTM model.

#### **Comparative Analysis**

Comparing both approaches, it is evident that the **deep learning models** generally outperformed the **machine learning models** in terms of all evaluated metrics:

- **CNN-LSTM** emerged as the top-performing model overall, achieving the highest scores in all metrics, including accuracy, precision, recall, and F1-score.
- **BERT + BiLSTM** and **Single-Layer LSTM** also demonstrated strong performance, suggesting that deep learning models, especially those incorporating advanced

embeddings (like BERT) or hybrid architectures (like CNN-LSTM), can handle the complexities of text data more effectively.

- Among machine learning models, **Logistic Regression** and **Bagging Decision Tree** performed well, but they were still outpaced by the best deep learning models, highlighting the benefits of using neural networks for tasks requiring deeper contextual understanding and feature extraction.

### **CNN-LSTM Model Performance and Overfitting Mitigation**

The CNN-LSTM model demonstrated strong performance during training, with the training accuracy increasing from 70.18% in Epoch 1 to 94.17% in Epoch 10. However, signs of potential overfitting were observed as the validation accuracy plateaued, while the validation loss started to increase after Epoch 7. For example, the validation loss decreased until Epoch 7 (0.3011) but began to rise from Epoch 8 (0.3078) onwards.

To address this overfitting issue, the EarlyStopping mechanism was applied with the following settings:

- **Monitor:** Validation loss (val\_loss)
- **Patience:** 3 epochs (the model stops training if the validation loss does not improve for 3 consecutive epochs)
- **Restore Best Weights:** True (the model restores weights from the epoch with the lowest validation loss)

In this case, EarlyStopping selected the weights from Epoch 7, where the validation loss was at its minimum (0.3011) before it began to increase. This approach effectively prevented overfitting by restoring the best model state from the epoch with the optimal balance between training and validation performance. As a result, the model maintained robust generalization to new data, preventing it from learning noise and irrelevant patterns from the training set.

By applying EarlyStopping, the CNN-LSTM model's performance was optimized for both accuracy and generalization, ensuring that it did not overfit to the training data.

```
Epoch 5/20  
2110/2110 ————— 142s 52ms/step - accuracy: 0.8948 - loss: 0.2512 - val_accuracy: 0.8714 - val_loss: 0.3043  
Epoch 6/20  
2110/2110 ————— 112s 53ms/step - accuracy: 0.9063 - loss: 0.2267 - val_accuracy: 0.8752 - val_loss: 0.3024  
Epoch 7/20  
2110/2110 ————— 140s 52ms/step - accuracy: 0.9166 - loss: 0.2030 - val_accuracy: 0.8773 - val_loss: 0.3011  
Epoch 8/20  
2110/2110 ————— 142s 52ms/step - accuracy: 0.9260 - loss: 0.1833 - val_accuracy: 0.8801 - val_loss: 0.3078  
Epoch 9/20  
2110/2110 ————— 142s 52ms/step - accuracy: 0.9345 - loss: 0.1637 - val_accuracy: 0.8816 - val_loss: 0.3116  
Epoch 10/20  
2110/2110 ————— 142s 52ms/step - accuracy: 0.9417 - loss: 0.1470 - val_accuracy: 0.8831 - val_loss: 0.3228
```

*Figure 5.1.4 CNN\_LSTM Training Phase*

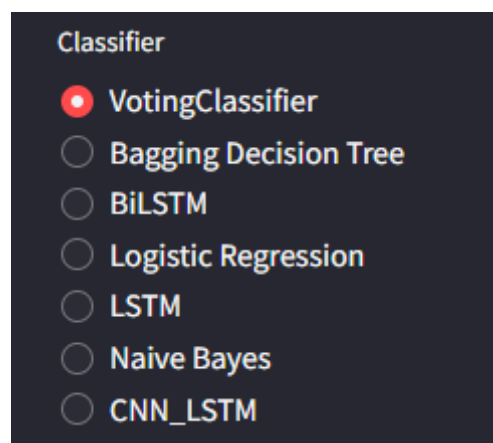
The evaluation shows that deep learning models, particularly the CNN-LSTM, outperformed traditional machine learning models in detecting hate speech, achieving the highest scores across all metrics. While Logistic Regression and Bagging Decision Tree were the best among machine learning approaches, they were still outpaced by deep learning methods that effectively capture complex text patterns. The use of EarlyStopping in the CNN-LSTM model also helped mitigate overfitting, ensuring robust generalization to new data. Overall, deep learning models, especially those with advanced embeddings and hybrid architectures, proved to be more effective for this task.

## 5. Deployment

To deploy the trained models and facilitate user interaction, a Streamlit web application was developed. The main features and functionalities of the application are as follows:

### 1. Model Selection:

- Users can choose from a variety of classifiers, including specific models or an ensemble option labeled “VotingClassifier” which included CNN\_LSTM, BiLSTM, Bagging Decision Tree. If no specific classifier is selected, the default is the “VotingClassifier” option, which aggregates predictions from all available models using a voting classifier approach.



*Figure 5.1.4 Model Selection Interface*

### 2. Explainer Selection:

- The application provides options to choose between SHAP and LIME for model explanation. Note that when the “VotingClassifier” classifier is selected, explainer options are disabled.

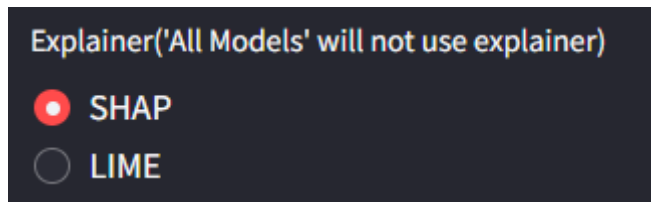


Figure 5.1.5 Explainer Selection Interface

### 3. Text Input and Prediction:

- Users can input text into a prompt and click the “Result” button to generate predictions. The chosen model will then provide probabilities for both non-hate and hate classifications.

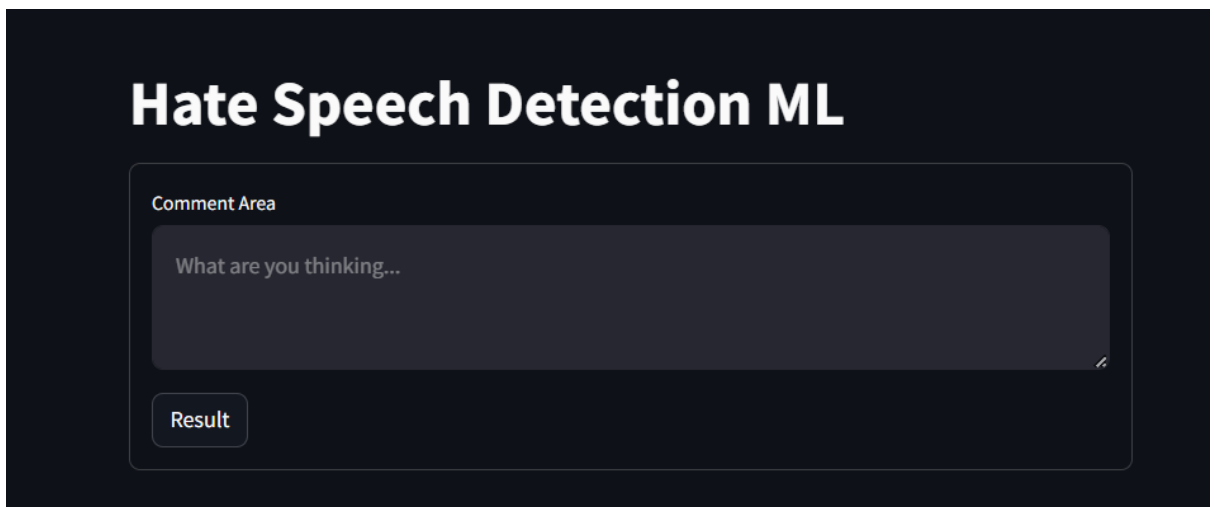
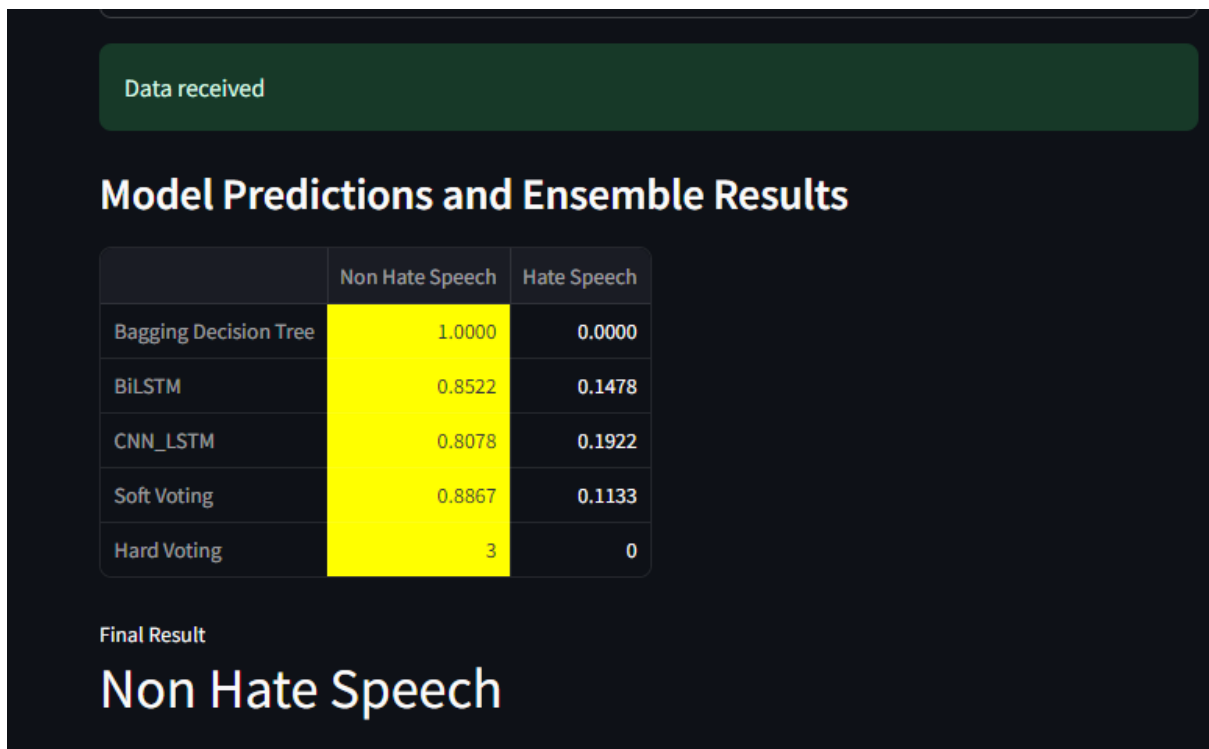


Figure 5.1.6 Text Input and Result Button Interface

### 4. Output Display:

- For “VotingClassifier”:
  - A table displays probabilities for non-hate and hate classifications from CNN\_LSTM, BiLSTM, Bagging Decision Tree models.
  - It includes results from hard voting and soft voting. Hard voting typically determines the final result.





*Figure 5.1.7 Output Display of Voting Classifier*

- **For Specific Models:**

- The selected model's probabilities for non-hate and hate classifications are displayed.
- Additionally, the selected explainer (SHAP or LIME) provides insights into the model's decision-making process.

This deployment ensures an interactive and user-friendly interface for real-time hate speech detection and model interpretation.

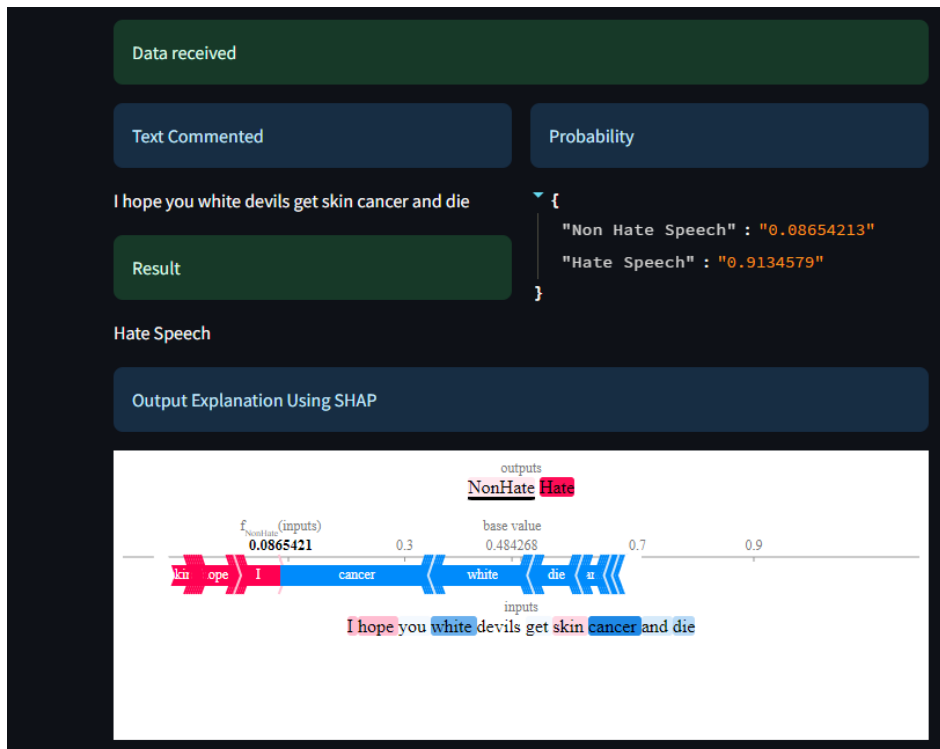


Figure 5.1.8 Output Display for Specific Models

# Chapter 6

## System Evaluation and Discussion

### 6.1 Testing Cases

Test Case ID	Description	Expected Result	Actual Result (Pass/Fail)
TC-001	Streamlit able to load trained model	No error when starting the web application	Pass
TC-002	User able to prompt text in text area and see the result	Display result after clicking "Result" button	Pass
TC-003	System able to check the text error	When user click "Result" button: If no text: "No Data received" If all digits: "Digits dont have hate element"	Pass
TC-004	Every Classifier should be able to predict	Classifier "All Models" display the table have all classifiers' output	Pass
TC-005	Classifier "All Models" should be able to make final output	Classifier able to determine final decision through hard voting and soft voting	Pass
TC-006	Input hate speech to "All Models"	Final predicts "hate" class with high voting	Pass
TC-007	Input valid non-hate text to "All Models"	Final Predicts "non-hate" class with hight voting	Fail
TC-008	Input valid non-hate text to CNN-LSTM model	Model predicts "non-hate" class with high probability	Pass
TC-009	Input valid hate text to CNN-LSTM model	Model predicts "hate" class with high probability	Pass
TC-010	Input valid hate text to Logistic Regression model	Model predicts "hate" class with high probability	Pass

TC-011	Input valid non-hate text to Logistic Regression model	Model predicts "non-hate" class with high probability	Fail
TC-012	Input valid hate text to Naive Bayes model	Model predicts "hate" class with high probability	Pass
TC-013	Input valid non-hate text to Naive Bayes model	Model predicts "non-hate" class with high probability	Fail
TC-014	Input valid hate text to Bagging Decision Tree model	Model predicts "hate" class with high probability	Pass
TC-015	Input valid non-hate text to Bagging Decision Tree model	Model predicts "non-hate" class with high probability	Pass
TC-016	Input valid hate text to LSTM model	Model predicts "hate" class with high probability	Pass
TC-017	Input valid non-hate text to LSTM model	Model predicts "non-hate" class with high probability	Fail
TC-018	Input valid hate text to BiLSTM model	Model predicts "hate" class with high probability	Pass
TC-019	Input valid non-hate text to BiLSTM model	Model predicts "non-hate" class with high probability	Pass

## 6.2 Model Evaluation

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression (CountVectorizer)	0.84	0.84	0.84	0.84

Naive Bayes (CountVectorizer)	0.80	0.74	0.92	0.82
Decision Tree (CountVectorizer)	0.66	0.61	0.93	0.73
Bagging Decision Tree (CountVectorizer)	0.83	0.82	0.84	0.83
Logistic Regression (TF-IDF)	0.84	0.84	0.84	0.84
Naive Bayes (TF-IDF)	0.82	0.78	0.89	0.83
Decision Tree (TF-IDF)	0.66	0.61	0.93	0.73
Bagging Decision Tree (TF-IDF)	0.83	0.82	0.85	0.83
Word2Vec + Stacked LSTM	0.80	0.79	0.83	0.81
BERT + BiLSTM	0.82	0.83	0.81	0.82
Embedding + CNN-LSTM	0.88	0.87	0.89	0.88
Embedding + Single-Layer LSTM	0.85	0.84	0.87	0.85

The evaluation of the models reveals that the deep learning models generally outperformed the machine learning models in all key metrics. Among the machine learning approaches, Logistic Regression and Bagging Decision Tree achieved the highest scores, with consistently strong performance across accuracy, precision, recall, and F1-score. In contrast, the Decision Tree models, both with CountVectorizer and TF-IDF, showed lower accuracy and precision, although they excelled in recall.

The deep learning models, particularly the Embedding + CNN-LSTM architecture, demonstrated superior performance with the highest accuracy and F1-score. This model effectively captured both local and global dependencies in the text data, making it the most effective for the hate speech detection task. The other deep learning models, such as BERT + BiLSTM and Embedding + Single-Layer LSTM, also performed well but did not surpass the CNN-LSTM model.

Overall, the results highlight the effectiveness of advanced deep learning techniques in handling complex text classification tasks compared to traditional machine learning methods.

## 6.3 Project Challenges

Several challenges were encountered during the development of this project, requiring careful problem-solving strategies.

The first challenge was managing memory usage when using BERT embeddings. Due to the large size of the dataset and the complexity of the BERT model, the training process consumed an excessive amount of RAM. Even in a high-performance environment such as Google Colab with 334GB of RAM and TPU support, the training could not proceed without running out of memory. To address this issue, PyTorch's DataLoader was implemented with an appropriate batch size to handle the data in smaller, more manageable portions, effectively reducing the RAM usage and allowing the training process to complete successfully.

The second challenge arose with the Decision Tree classifier, which showed signs of underfitting and resulted in poor performance metrics. This was likely due to the high variance and instability associated with Decision Trees, particularly when dealing with complex datasets. To overcome this, a Bagging Classifier was employed, combining multiple Decision Trees to reduce variance and stabilize the model's predictions. This approach significantly improved the accuracy score, providing a more reliable model for classification tasks.

By addressing these challenges, the overall model performance and resource management were enhanced, leading to more effective and efficient machine learning workflows.

# Chapter 7

## Conclusion and Recommendations

### 7.1 Conclusion

This project aimed to develop a comprehensive hate speech detection system utilizing advanced machine learning and deep learning techniques. The objectives outlined in Chapter 1 were:

1. **Create a Model Able to Distinguish Hate and Non-Hate:** This objective was achieved by developing and training various machine learning models (Logistic Regression, Naive Bayes, Decision Trees, Bagging Decision Tree) and deep learning models (Word2Vec + Stacked LSTM, BERT + BiLSTM, Embedding + CNN-LSTM, Embedding + Single-Layer LSTM). Each model was rigorously evaluated for its accuracy, precision, recall, and F1-score, demonstrating the system's capability to effectively distinguish between hate and non-hate speech.
2. **Experiment with Different Algorithms to Build Models:** A variety of algorithms were tested, including both traditional machine learning methods and advanced deep learning architectures. The experimentation revealed that deep learning models, particularly the CNN-LSTM model, outperformed traditional methods in all evaluation metrics. The use of ensemble learning through a voting classifier further enhanced the model's performance.
3. **Enhance Model Interpretability:** To address the need for transparency, interpretability tools such as SHAP and LIME were integrated into the system. This addition allowed users to gain insights into how the models made their predictions, thereby building trust and understanding in the system's decisions.
4. **Develop a User Interface:** A user web interface was created using Streamlit, enabling users to input text, select classifiers, and receive real-time feedback. The interface also supports model explainability, providing users with clear explanations of the prediction results.

In summary, the project successfully met its objectives by developing a robust and interpretable hate speech detection system with a user interface. The integration of various machine learning and deep learning models, coupled with ensemble learning techniques, ensured high accuracy and reliability. Future work could focus on further enhancing model performance through

additional data sources, exploring new algorithms, and refining the user interface for even better user experience.

## **7.2 Recommendations**

### **Implement User Feedback Collection with Reinforcement Learning**

#### **1. Integrate a User Feedback Mechanism:**

- Incorporate a feedback feature into the user interface to allow users to evaluate the model's prediction outcomes. Users can mark whether a prediction is correct or incorrect, helping identify model misclassifications and biases.
- The feedback collection could be implemented simply, such as with a "Correct/Incorrect" button or by allowing users to provide additional details on why a prediction might be wrong.

#### **2. Apply Reinforcement Learning (RL) Techniques:**

- Use user feedback as a reward signal in a reinforcement learning framework. Adjust the model by incorporating RL techniques to optimize its decision-making strategy continuously based on feedback.
- Methods like policy gradient algorithms (e.g., REINFORCE or PPO) could be utilized to train the model, enabling it to perform better with new inputs and adapt to dynamic environments. Positive user feedback (correct classifications) would provide positive reinforcement, while negative feedback would result in penalties, driving the model to improve.

#### **3. Develop a Continuous Learning Pipeline:**

- Establish a continuous learning pipeline that updates model parameters dynamically based on user feedback. This pipeline can operate in the background, fine-tuning the model in real-time to better accommodate new input data and user expectations.
- Such an approach would allow the system to adapt to changes in language and social trends, handling new forms of hate speech and expressions effectively.

#### **4. Monitor and Evaluate Feedback-Driven Learning:**

- Regularly monitor and evaluate the model's performance throughout the feedback-driven learning process to ensure that incorporating user feedback and RL techniques improves the prediction capabilities.



- Set baseline metrics (such as performance improvement curves) to measure the actual benefits brought by reinforcement learning and ensure the effectiveness and reliability of user feedback.

By combining user feedback with reinforcement learning, the model can self-optimize continuously, gradually reducing misclassifications and increasing the accuracy and robustness of hate speech detection. This approach enhances both the practical effectiveness of the model and user trust and engagement, making the system more human-centered and dynamically adaptable.

# REFERENCES

- [1]I. Mollas, Z. Chrysopoulou, S. Karlos, and G. Tsoumakas, “ETHOS: a multi-label hate speech detection dataset,” *Complex & Intelligent Systems*, Jan. 2022, doi: <https://doi.org/10.1007/s40747-021-00608-2>
- [2]S. MacAvaney, H.-R. Yao, E. Yang, K. Russell, N. Goharian, and O. Frieder, “Hate speech detection: Challenges and solutions,” *PLOS ONE*, vol. 14, no. 8, p. e0221152, Aug. 2019, doi: <https://doi.org/10.1371/journal.pone.0221152>
- [3]Wafa Alorainy, *An Integrated Framework for Detecting Online Harms, Modelling and Disrupting of Cyberhate Networks*. Wafa S. Alorainy, 2022.
- [4]D. Soni, “Introduction to Naive Bayes Classification,” *Medium*, May 17, 2018. Available: <https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54>
- [5]A. Awan, “Naive Bayes Classifier Tutorial: with Python Scikit-learn,” *www.datacamp.com*, Mar. 2023. Available: <https://www.datacamp.com/tutorial/naive-bayes-scikit-learn>
- [6]M. Banoula, “Introduction to Long Short-Term Memory(LSTM) | Simplilearn,” *Simplilearn.com*, Apr. 27, 2023. Available: <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm>
- [7]“Create Bidirectional LSTM Layer,” *Mathworks.cn*, 2024. Available: <https://ww2.mathworks.cn/help/deeplearning/ref/nnet.cnn.layer.bilstmlayer.html>. [Accessed: Sep. 11, 2024]
- [8]J. Zhang, Y. Li, J. Tian, and T. Li, “LSTM-CNN Hybrid Model for Text Classification,” *IEEE Xplore*, Oct. 01, 2018. doi: <https://doi.org/10.1109/IAEAC.2018.8577620>. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8577620>
- [9]Jitendra Singh Malik, H. Qiao, and G. Pang, “Deep Learning for Hate Speech Detection: A Comparative Study,” *arxiv.org*, Dec. 07, 2023. Available: <https://arxiv.org/html/2202.09517v2>
- [10]B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee, “HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 17, pp. 14867–14875, May 2021, doi: <https://doi.org/10.1609/aaai.v35i17.17745>
- [11]A. Arango, J. Pérez, and B. Poblete, “Hate speech detection is not as easy as you may think: A closer look at model validation (extended version),” *Information Systems*, p. 101584, Jun. 2020, doi: <https://doi.org/10.1016/j.is.2020.101584>. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0306437920300715>

- [12]S. Zannettou, J. Finkelstein, B. Bradlyn, and J. Blackburn, “A Quantitative Approach to Understanding Online Antisemitism,” Proceedings of the International AAAI Conference on Web and Social Media, vol. 14, pp. 786–797, May 2020, doi: <https://doi.org/10.1609/icwsm.v14i1.7343>
- [13]T. Ranasinghe and M. Zampieri, “An Evaluation of Multilingual Offensive Language Identification Methods for the Languages of India,” Information, vol. 12, no. 8, p. 306, Jul. 2021, doi: <https://doi.org/10.3390/info12080306>
- [14]D. Mody, Y. Huang, and T. E. Alves de Oliveira, “A curated dataset for hate speech detection on social media text,” Data in Brief, vol. 46, p. 108832, Feb. 2023, doi: <https://doi.org/10.1016/j.dib.2022.108832>
- [15]souravkr26, “Hate\_Speech,” Kaggle.com, Jul. 03, 2024. Available: <https://www.kaggle.com/code/souravkr26/hate-speech>. [Accessed: Sep. 11, 2024]
- [16]N. Yancey-Bragg, “Hate crimes reached record levels in 2023. Why ‘a perfect storm’ could push them higher,” USA TODAY, Jan. 05, 2024. Available: <https://www.usatoday.com/story/news/nation/2024/01/05/hate-crimes-hit-record-levels-in-2023-why-2024-could-be-even-worse/72118808007/>

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: T2 Y3</b>	<b>Study week no.:2</b>
<b>Student Name &amp; ID: Chai Yun Wai (22ACB00222)</b>	
<b>Supervisor: Dr Abdulkarim Kanaan Jebna</b>	
<b>Project Title: AI FOR A POSITIVE WEB: ANALYZING HATE IN SOCIAL MEDIA</b>	

## 1. WORK DONE

- Model Training Using Machine Learning

## 2. WORK TO BE DONE

- Model Training Using Deep Learning
- Create User Interface

## 3. PROBLEMS ENCOUNTERED

- Decision Tree Underfitting

## 4. SELF EVALUATION OF THE PROGRESS

- Great

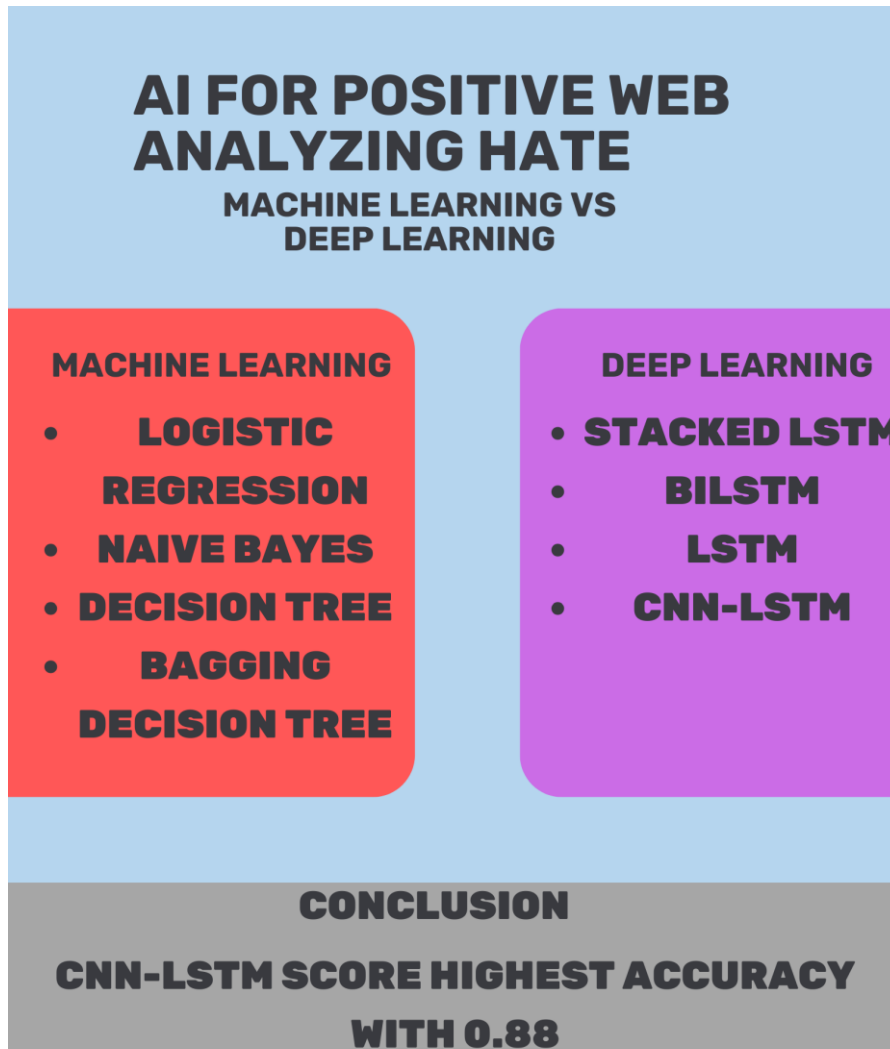


Supervisor's signature



Student's signature

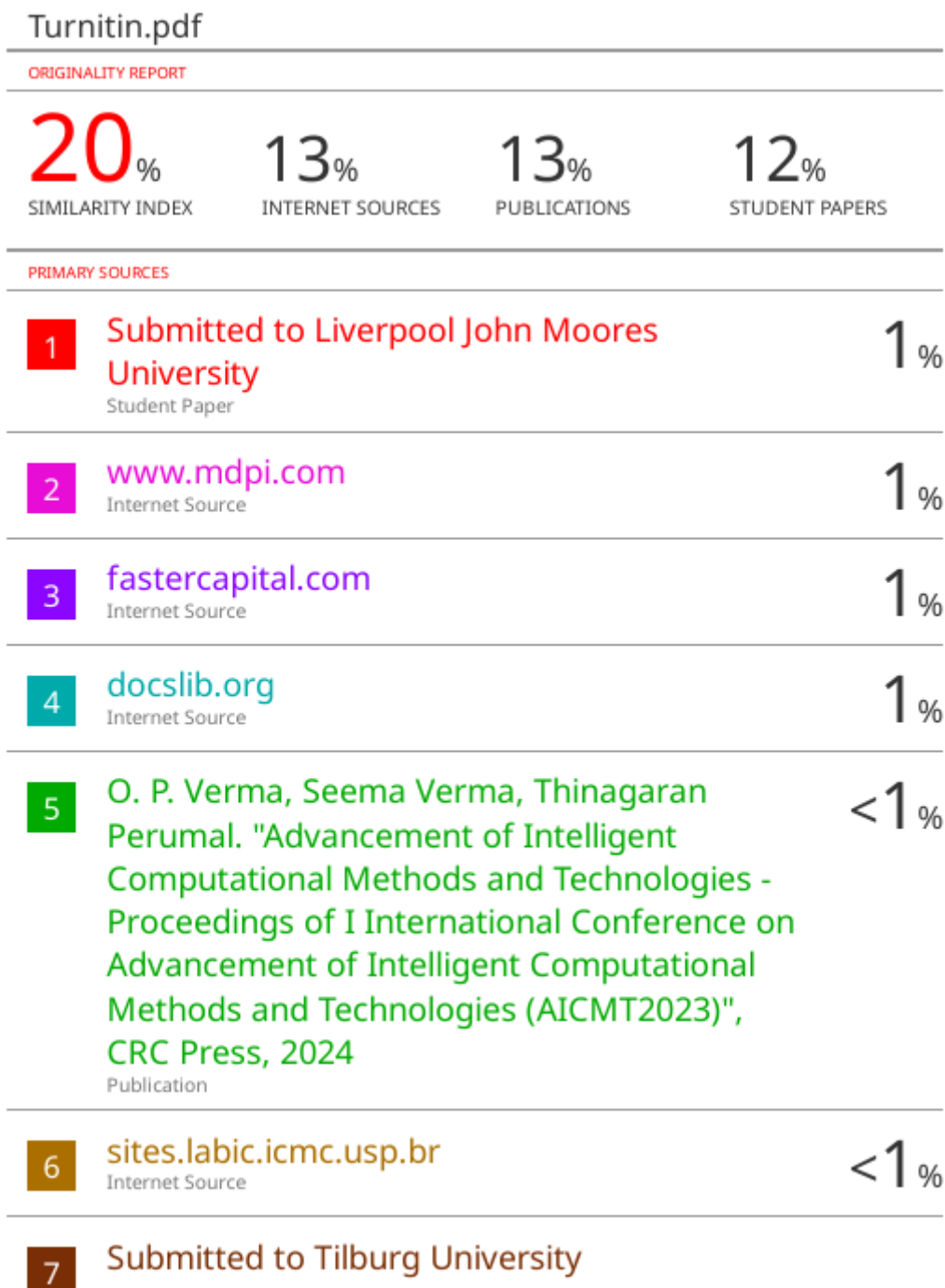
## POSTER



PROJECT DEVELOPER: CHAI YUN WAI

PROJECT SUPERVISOR: DR ABDULKARIM JEBNA

# PLAGIARISM CHECK RESULT



8	Internet Source	<1 %
9	O. P. Verma, Seema Verma, Thinagaran Perumal. "Advancement of Intelligent Computational Methods and Technologies - Proceedings of I International Conference on Advancement of Intelligent Computational Methods and Technologies (AICMT2023)", CRC Press, 2024 Publication	<1 %
10	<a href="https://sites.labic.icmc.usp.br">sites.labic.icmc.usp.br</a> Internet Source	<1 %
11	Submitted to Tilburg University Student Paper	<1 %
12	Submitted to University of Greenwich Student Paper	<1 %
13	<a href="https://internationalhatestudies.com">internationalhatestudies.com</a> Internet Source	<1 %
14	Kamal Malik, Moolchand Sharma, Suman Deswal, Umesh Gupta, Deevyankar Agarwal, Yahya Obaid Bakheet Al Shamsi. "Explainable Artificial Intelligence for Autonomous Vehicles - Concepts, Challenges, and Applications", CRC Press, 2024 Publication	<1 %

<b>Universiti Tunku Abdul Rahman</b>			
<b>Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form ID: IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	Chai Yun Wai
<b>ID Number(s)</b>	22ACB00222
<b>Programme / Course</b>	IB
<b>Title of Final Year Project</b>	<b>AI FOR A POSITIVE WEB: ANALYZING HATE IN SOCIAL MEDIA</b>

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)</b>
<b>Overall similarity index:</b> <u>20</u> %  <b>Similarity by source</b> Internet Sources: <u>13</u> % Publications: <u>13</u> % Student Papers: <u>12</u> %	
<b>Number of individual sources listed of more than 3% similarity:</b> <u>0</u>	
<b>Parameters of originality required and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

***Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.***



\_\_\_\_\_  
Signature of Supervisor

\_\_\_\_\_  
Signature of Co-Supervisor

Name: Dr. Abdulkarim M. Jamal Kanaan Jebna

Name: \_\_\_\_\_

Date: 13/09/2024

Date: \_\_\_\_\_





# UNIVERSITI TUNKU ABDUL RAHMAN

## FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

### CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	22ACB00222
Student Name	Chai Yun Wai
Supervisor Name	Dr Abdulkarim Kanaan Jebna

TICK (√)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
	Table of Contents
	List of Figures (if applicable)
	List of Tables (if applicable)
	List of Symbols (if applicable)
	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

\*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

*chaiyunwai*

(Signature of Student)

Date: 13 September 2024