Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques

BY

THONG WEI XIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: **Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques**

Academic Session: **Jan 2024**

I **THONG WEI XIN**

(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,

(Author's signature)

(Supervisor's signature)

Address:
No. 46, Taman Wira Damai 3
Taman Wira Damai
Kampung Koh
32000 Sitiawan, Perak

Ts. Dr. Vikneswary Jayapal

Supervisor's name

Date: **April 10, 2024**

Date: 25/04/24

FACULTY/INSTITUTE* OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: April 10, 2024

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ***Thong Wei Xin*** (ID No: ***20ACB02627***) has completed this final year project/ dissertation/ thesis* entitled **"Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques"** under the supervision of **Ts Dr. Vikneswary a/p Jayapal (Supervisor)** from the Department of Computer and Communication Technology, Faculty/Institute* of Information and Communication Technology.

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,

(*Thong Wei Xin*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques**" 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 : 

Name : **THONG WEI XIN**

Date : **April 10, 2024**

ACKNOWLEDGEMENTS

I would like to express thanks and appreciation to my supervisor, Ts Dr. Vikneswary a/p Jayapal and my moderator, Dr. Lim Jia Qi who have given me a golden opportunity to involve in the Deep Learning 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

This project was conducted to develop hate speech detection models using several deep learning techniques with different word embedding techniques to detect English hate speech tweets on X (Twitter) with the goal of enhancing the online communication environment and reducing the suicide rate due to cyberbullying. Several deep learning techniques were utilised in this project, such as CNN, BiLSTM, a pretrained DistilBERT model named 'distilbert/distilbert-base-uncased', and a pretrained RoBERTa model named 'facebook/roberta-hate-speech-dynabench-r4-target'. The word embedding techniques utilised in this project can be classified into two groups: those utilising a single word embedding technique such as GloVe (Global Vectors for Word Representation), Word2Vec, or word embedding vectors provided by DistilBERT and RoBERTa itself, and those combining two different word embedding techniques by stacking, averaging, and taking the root mean square of them. In comparison to the old trend models that utilised word-based tokenisation in the preprocessing of data, subword tokenisation is utilised in this project to tokenise the tweets in the dataset.

Several papers on cyberbullying or hate speech detection models using deep learning were reviewed, outlining the strengths and weaknesses of the models developed by various authors. In addition to detailing the architectures of these models used in this project, the paper also explains the model development process, techniques employed to address class imbalance issues or hyperparameter tuning, which were visualised or explained to provide newcomers in text classification with a comprehensive understanding of how models were developed. The most significant focus was on the performance evaluation and analysis of the DistilBERT, RoBERTa transformer models, as well as those CNN and BiLSTM models utilising single word embedding techniques and combining different word embedding techniques.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

σ	Lowercase sigma (Sigmoid function)

LIST OF ABBREVIATIONS

CCDH	Centre for Countering Digital Hate
NLP	Natural Language Processing
DL	Deep Learning
CNN	Convolutional Neural Network
PCNN	Pronunciation-based Convolutional Neural Network
CNN-CB	Convolutional Neural Network
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
LSTM	Long Short-Term Memory
BiLSTM	Bidirection Long Short-Term Memory
GCN	Graph Convolutional Network
DQE	Dynamic Query Expansion
GloVe	Global Vectors for Word Representation
HTML	Hypertext Markup Language
BERT	Bidirectional Encoder Representations from Transformers
RNN	Recurrent Neural Network
NSP	Next Sentence Prediction
RoBERTa	Robustly optimized BERT approach
MCC	Matthews Correlation Coefficient

Chapter 1
Introduction

In this chapter, we present the problem statement and motivation of our project, our contributions, objectives to achieve the project goal, project direction and scope, as well as the report organisation.

1.1 Problem Statement and Motivation

A study by the Centre for Countering Digital Hate (CCDH), using over-time data from Brandwatch, found that after Elon Musk took over X (Twitter), there was a 202% increase in daily tweets mentioning the racist term (now at 3,876). Homophobic term usage rose by 58% to 3,964 daily tweets, misogynist term mentions increased by 33% to 17,937 daily tweets, and transphobic term usage surged by 62% to 5,117 daily tweets [1]. Therefore, this underscores the urgent need for an effective hate speech detection model to identify hate speech on X (Twitter) and enhance the online communication environment. However, most of the detection models in the market primarily focused on the broader term of cyberbullying. Only a minority of detection models were developed to specifically target certain types of cyberbullying on social media platforms, such as body shaming, hate speech, sexism, racism, and others.

Besides, there is a lack of evidence in model evaluation demonstrating that combining multiple word embedding techniques outperformed using a single word embedding technique. This highlighted a gap in model evaluation within the field of cyberbullying/ hate speech detection, which this project aimed to address. Furthermore, while most cyberbullying/ hate speech detection models currently utilised word-based tokenisation, there was a recent emergence of subword tokenisation as a new trend in Natural Language Processing (NLP) tasks. It is believed that subword tokenisation may be the optimal tokenisation technique in NLP tasks, as it breaks down unknown words into recognised components and assists in handling words that are combined without clear spacing.

Moreover, hate speech is an attack aimed at harming, and harassing the victim, it might bring about a series of adverse impacts such as depression, anxiety, and unhealthy psychology, or even suicidal behaviour. According to the study conducted by [2], cyberbullying is a

significant key factor for suicidal behaviour among adolescents. All these issues contributed to the formation of an unhealthy social environment in the future. The motivation of this project was to develop a deep learning model to detect hate speech on X (Twitter) effectively and foster a healthier online communication environment by specifically targeting instances of cyberbullying. Such a model could play a crucial role in mitigating the impact of cyberbullying victimisation, thereby contributing to the reduction of suicidal behaviour associated with these harmful online experiences.

1.2 Objectives

The project aimed to develop a hate speech detection model that could be use to identify hate speech on X (Twitter) using deep learning techniques, with the goal of improving the online communication environment. To achieve this goal, the following outlined objectives needed to be accomplished:

1. Investigated deep learning models capable of detecting hate speech on X (Twitter) and different word embedding techniques. Conducted a literature review to comprehend models proposed and identified the strengths and weaknesses of each model. This provided comprehensive insights for selecting deep learning algorithms for development.
2. Developed deep learning classifiers with different word embedding techniques capable of being used for detecting hate speech on X (Twitter), based on the findings from the investigation. Decided on the deep learning techniques to be used for the hate speech detection model based on the literature review and applied it to develop a deep learning model for detecting hate speech on X (Twitter).
3. Evaluate the performance of the developed deep learning models by employing various evaluation metrics, including precision, recall, and F1 score. This evaluation is essential, as it is used to demonstrate that the developed models work effectively in detecting hate speech on X (Twitter).

1.3 Project Scope and Direction

At the end of this project, an enhanced hate speech detection model leveraged deep learning techniques will be delivered. The primary focus was on identifying instances of hate speech in the English language on the X (Twitter) platform. To enhance the model's efficacy, combination of word embedding techniques was employed. Unlike the conventional approach

of using a single word embedding, this technique involved combining two-word vectors, thereby enriching the model's understanding of context and relationships within the text.

A word embedding vector served as a word representation that captured intricate dependencies and relationships between words. By utilising these word vectors, it enhanced the deep learning model's ability to comprehend the nuances of a sentence. Combining distinct word embedding vectors further enriched the model's understanding, facilitating improved performance. Essentially, this approach allowed the model to leverage a broader and more naunced context from multiple word embeddings, leading to enhanced comprehension and consequently, superior overall performance.

1.4 Contributions

The key contribution of this project was in proposing a substantial improvement to the performance of a hate speech detection model. This enhancement was achieved through the application of advanced combined word embedding techniques, specifically tailored for the detection of English language hate speech on the X (Twitter) platform. The primary aim was to positively influence the online communication environment on X (Twitter), ultimately playing a role in reducing the cyberbullying victimisation rate among X (Twitter) users. This project aspired to make a tangible and positive impact on the online experience for users, fostering a safer and more respectful digital space.

Additionally, the implementation of an effective hate speech detection model held the potential to significantly diminish the human resources required for identifying suspicious hate speech tweets. This not only amplified overall work efficiency but also concurrently diminished the financial expenditure associated with the recruitment of examiners dedicated to the identification of hate speech. By automating the initial phase of content moderation, organisations could streamline their processes, allowing human moderators to concentrate their efforts on cases that demanded nuanced judgement and contextual understanding.

1.5 Report Organization

This report was structured into 7 chapters: Chapter 1 - Introduction, Chapter 2 - Literature Review, Chapter 3 - System Methodology/ Approach, Chapter 4 - System Design, Chapter 5 - System Implementation, Chapter 6 - System Evaluation and Discussion, and Chapter 7 - Conclusion and Recommendation.

Chapter 1 served as the introduction to the project, encompassing the problem statement and motivation behind it. It outlined the objectives aimed at achieving the project's goals, defined the project scope, highlighted the contributions made by this project, and provided an overview of the report's organisation.

Chapter 2 serves as an investigation on the research papers that related to hate speech detection or cyberbullying detection using deep learning tehcniques. In this chapter, it contained the summarisation of the investigated research papers and the comparison of strengths and weaknesses among different papers.

Chapter 3 described the model architecture utilised in this project, the hardware and software specifications utilised in this project were discussed in Chapter 4. While Chapter 5 served as a chapter to introduce the system block diagram of this project and the specific components in the system block diagram.

In Chapter 6, we evaluated the performance of the developed deep learning hate speech detection models using appropriate evaluation metrics. We then analysed the evaluation results. Finally, Chapter 7 summarised the findings, and provided recommendations for future work.

Chapter 2

Literature Review

2.1 Previous works on Deep Learning Hate Speech/ Cyberbullying Detection Models

2.1.1 Cyberbullying Detection with a Pronunciation Based Convolutional Neural Network [4]

X. Zhang et al. [4] introduced a technique that used a pronunciation-based convolutional neural network (PCNN) for the purpose of cyberbullying detection. This paper aimed to address the issue of identifying instances of cyberbullying in text-based communication, which could be challenging due to the informal and often abbreviated nature of online text. The main technique introduced in this paper involved the convertion of words into a phonetic representation using eSpeak as a feature in the CNN model. Figure 2.1.1.1 illustrated a table exemplifying word-to-pronunciation conversion, adapted from X. Zhang et al. [4].

Word and phrases	Phonetic code	Effect on the data
"fuck, fuc, fuk"	*f'Vk*	Positive
"fuckk"	*f'Vkk*	Neutral
"shitt, shit"	*S'It*	Positive
"suck, suk, suc"	*s'Vk*	Positive
"dik, dic, dick"	*d'Ik*	Positive
"guesss, guess"	*g'Es*	Positive
"bitchh, bitch, bich"	*b'ItS*	Positive
"cool, cooll"	*k'u:l*	Positive
"cum, come"	*k'Vm*	Negative

Figure 2.1.1. 1 Examples of the word-to-pronunciation conversion. Adapted from [4].

It achieved an accuracy of 0.989, a precision rate of 0.989, a recall rate of 0.972, and an F1 score of 0.980 with the utilisation of threshold moving to solve class imbalance issue.

Besides, it achieved an accuracy of 0.990, a precision rate of 0.991, a recall rate of 0.972, and an F1 score of 0.981 with the utilisation of cost function adjustment to solve class imbalance issue. Moreover, it achieved an accuracy of 0.990, a precision rate of 0.991, a recall rate of 0.975, and an F1 score of 0.983 with the utilisation of threshold moving and cost function adjustment to solve class imbalance issue.

This paper offered several advantages. Firstly, it leveraged Word-to-Pronunciation conversion to enhance the model's ability to comprehend misspelled words, resulting in improved perfromance. Additionally, the paper introduced innovative approaches to address class imbalance issues beyond traditional oversampling and undersampling techniques. These solutions contributed to a more effective system performance.

This paper had several limitations. Firstly, there was a class imbalance issue in the dataset, which could impact the model's performance. Additionally, the word-to-pronunciation conversion technique employed may have introduced some level of noise into the data. For example, a bad word 'cum' would have a same phonetic representation as 'come'. Moreover, the model proposed in this paper could not convert words with numbers to their respective phonetic representations, such as 'ugl1' and 'nigg13', designed to evade detection by the algorithm, as they are not proper alphabetic words.

2.1.2 Deep Learning Algorithm for Cyberbullying Detection [5]

M. A. Al-Ajlan and M. Ykhlef [5] introduced a novel algorithm named Convolutional Neural Network for CyberBullying (CNN-CB) for the purpose of detecting cyberbullying. CNN-CB utilised word embeddings to comprehend the meaning of words and their semantics in cyberbullying contexts. This approach eliminated the complexities associated with feature engineering shown in figure 2.1.2.1 and aimed for improved detection accuracy. CNN-CB was built upon a Convolutional Neural Network (CNN) framework, integrating the benefits of word embeddings. Experimental results indicated that the CNN-CB algorithm surpassed the performance of traditional content-based cyberbullying detection models, achieving an impressive accuracy rate of 95%.

The proposed algorithm offered several advantages. It leveraged word embeddings to enable the model to comprehend sentence meanings, moving beyond classifying sentences solely based on individual words. This approach eliminated the need for extensive feature engineering, simplifying the model's complexity. Additionally, the proposed model outperformed traditional content-based cyberbullying detection models.

However, there are some drawbacks in this paper. The dataset used in the study exhibits a class imbalance issue, which could potentially impact the model's performance by making it more likely to predict instances as non-cyberbullying. Furthermore, the recall rate of the proposed model was 73%, which means it is not very effective in detecting the actual positive cases of cyberbullying.

2.1.3 Offensive Language Detection using Artificial Neural Network [6]

M. Susanty, Sahrul, A. F. Rahman, M. D. Normansyah, and A. Irawan [6] proposed an artificial neural network (ANN) model to detect offensive language on an online platform. The model classified the results using a sigmoid function, with '1' signifying content of an offensive nature and '0' denoting content that is devoid of any offensive undertones. The proposed model achieved an accuracy of 99.18% for training, 94.28% for validation, and 96.8% for testing.

Eliminating duplicate characters in a word during the preprocessing phase was a strength of this model. This process was useful in correcting the word to its original form, reducing confusion for the model, and ensuring the word exists in a dictionary.

However, there were limitations. Human labeling could indeed introduce variability and potential inaccuracies due to personal cultural perspectives and biases. Different annotators may interpret offensive language differently based on their individual cultural backgrounds, experiences, and beliefs. This subjectivity could impact the consistency and accuracy of the labeled dataset. Additionally, the dataset used to train the model is too small, containing only 504 data. The small size of dataset could lead to overfitting to the training dataset, making the model less effective at detecting offensive language in other datasets.

2.1.4 Sexism Identification using BERT and Data Augmentation – EXIST2021 [7]

S. Butt [7] tested several machine learning models and deep learning models to identify sexism on social media by using a multilingual dataset containing English tweets and Spanish tweets. They augmented the identified classes. According to their results, BERT achieved the best F1 score of 78.02% for sexism identification and 49.08% for sexism categorization among deep learning models. The best result for a machine learning algorithm was Random Forest, which achieved an F1 score of 63.66% on the sexism identification task and an F1 score of 45.43% on the sexism categorization task.

There were several strengths of the model proposed by the authors. First, it could identify sexist sentences in English and Spanish. Moreover, it classified the identified sexism cases into specific types of sexism. Additionally, the proposed model took into consideration the emojis and repeated special characters used in a sentence.

The limitation of the model proposed by the authors is that translating the original text to German and then translating it back to the original language might cause the model to overfit to the dataset, as it might produce the same text as the original.

2.1.5 Deep Learning for Detecting Cyberbullying Across Multiple Social Media Platforms [8]

Agrawal and A. Awekar [8] conducted an experiment to provide a systematic analysis of cyberbullying detection on the topics of bullying, sexism, attack, and racism across multiple social media platforms. They utilised CNN, LSTM, BiLSTM, and BiLSTM with attention, along with transfer learning, to test the effectiveness of the developed model on different datasets.

Dataset	Label	P				R				F1			
		M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4
F+	Bully	0.93	0.91	0.91	0.90	0.90	0.85	0.81	0.91	0.91	0.88	0.86	0.91
T+	Racism	0.93	0.91	0.92	0.90	0.94	0.80	0.95	0.96	0.93	0.85	0.93	0.93
	Sexism	0.92	0.84	0.88	0.88	0.92	0.93	0.94	0.92	0.92	0.88	0.92	0.90
W+	Attack	0.92	0.70	0.90	0.87	0.83	0.54	0.81	0.86	0.88	0.61	0.85	0.87

Figure 2.1.5. 1 Performance comparison of various DNN models. Adapted from [8].

Figure 2.1.5.1 illustrated the performance of the deep learning models adapted from S. Agrawal and A. Awekar [8]. M1, M2, M3, M4 represented the CNN model, LSTM, BiLSTM, and BiLSTM with attention, respectively.

One strength of this paper was that it showed the comparison of the performance of deep learning models and the machine learning models on the same dataset. Besides, the utilisation of t-SNE visualised words that are most relevant to specific topics, providing a clear understanding of words related to those topics.

A limitation of this model was that it was trained using a class-imbalanced dataset, with non-cyberbullying instances outnumbering cyberbullying instances. This might cause the

model to preferentially predict instances as non-cyberbullying, thereby reducing the model's performance.

2.1.6 SOSNet: A Graph Convolutional Network Approach to Fine-Grained Cyberbullying Detection [9]

J. Wang, K. Fu, and C.-T. Lu [9] compiled 6 datasets from different authors and utilised Dynamic Query Expansion to expand the size of the dataset, forming a new twitter cyberbullying dataset with 6 classes: age, religion, ethnicity, gender, other cyberbullying, and non-cyberbullying. Then, they randomly sampled 8,000 data from each class to form a 47,000 tweets dataset. They also proposed a Graph Convolutional Network (GCN) model named SOSNet, which used several word embedding techniques to generate the word embedding for each tweet, acting as a node. The edges between the nodes are identified by using the thresholded cosine similarities between tweets embeddings (node).

Accuracy	LR	NB	KNN	SVM	XGB	MLP	SOSNet
SBERT	0.8900	0.8420	0.8240	0.9200	0.8950	0.9160	0.9270
BERT	0.8330	0.6820	0.7060	0.8480	0.8280	0.8600	0.8580
DistilBERT	0.8900	0.7320	0.7990	0.8840	0.8830	0.9030	0.8890
GloVe	0.8520	0.7000	0.7320	0.9060	0.8970	0.8870	0.8830
W2V	0.8280	0.7140	0.7230	0.9050	0.8930	0.8860	0.8710
FastText	0.8280	0.6090	0.6740	0.8710	0.8710	0.8680	0.8850
TF-IDF	0.8330	0.7890	0.2350	0.6460	0.9030	0.7340	0.7170
BOW	0.8350	0.7750	0.4180	0.6580	0.9180	0.7600	0.7670

Figure 2.1.6. 1 Test Accuracies—4,000 Tweets. Adapted from [9].

F1 Score	LR	NB	KNN	SVM	XGB	MLP	SOSNet
SBERT	0.8876	0.8403	0.8103	0.9190	0.8936	0.9142	0.9258
BERT	0.8305	0.6803	0.7034	0.8471	0.8264	0.8586	0.8552
DistilBERT	0.8878	0.7287	0.7953	0.8815	0.8805	0.9009	0.8876
GloVe	0.8480	0.6887	0.6916	0.9050	0.8954	0.8829	0.8825
W2V	0.8221	0.7038	0.6717	0.9028	0.8908	0.8821	0.8697
FastText	0.8245	0.6005	0.6427	0.8697	0.8690	0.8645	0.8833
TF-IDF	0.9298	0.7612	0.1559	0.6602	0.9028	0.7327	0.7113
BOW	0.8362	0.7408	0.3975	0.6721	0.9177	0.7636	0.7598

Figure 2.1.6. 2 Test F1 Scores—4,000 Tweets. Adapted from [9].

Figure 2.1.6.1 and figure 2.1.6.2 illustrated the systematic analysis of different word embedding techniques, achieving different accuracy rates and F1 scores. The highest accuracy rate achieved by the model was with the use of SBERT embedding techniques, reaching an accuracy rate of 0.9270 and an F1 score of 0.9258.

The strength of this paper lay in its use of the Dynamic Query Expansion (DQE) technique, a data mining technique employed to address the class imbalance issue. This technique gathers more natural data instead of generating the same data. Besides, the authors' compilation of six datasets from different authors contributed significantly to the field of data science by providing a diverse and comprehensive benchmark dataset.

However, labeling the instances obtained from the DQE technique could have raised concerns about incorrect labeling, as the labels were assigned by humans, which could have been influenced by human culture or religion.

2.1.7 An Application to Detect Cyberbullying Using Machine Learning and Deep Learning Techniques [10]

M. Raj, S. Singh, K. Solanki, and R. Selvanambi [10] proposed a hybrid deep learning-based cyberbullying detection model that combined CNN and BiLSTM. The model utilised stacked word embedding techniques and was applied to a cyberbullying detection application on X (Twitter) to detect cyberbullying. The stacked embedding techniques is a combination of GloVe and FastText word embeddings. The proposed model outperformed standalone deep learning models in terms of performance. The proposed hybrid deep learning model achieved an accuracy of 0.9135 before hyperparameter tuning and 0.9512 after hyperparameter tuning.

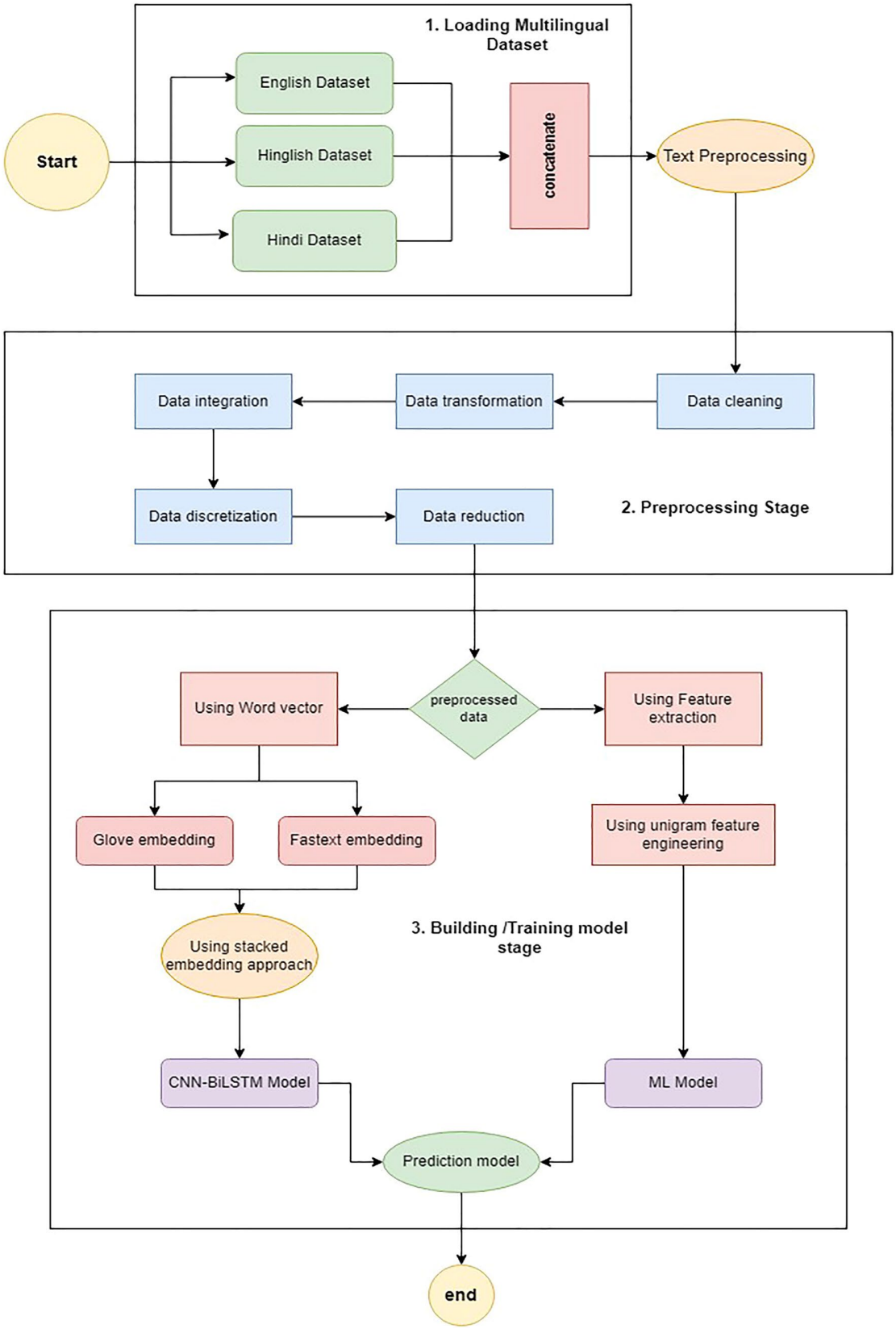


Figure 2.1.7. 1 Proposed Hybrid deep learning model with stacked word embedding techniques. Adapted from [10].

The model excelled in detecting cyberbullying across multiple languages. The hybrid deep learning model demonstrated superior performance compared to its standalone counterpart. Stacked embeddings play a crucial role in enhancing the model's overall performance.

A limitation of this paper is that the authors do not provide an analysis of the performance with the use of a single word embedding technique compared to the use of multiple word embedding techniques. It should provide a view to demonstrate that combining word embedding techniques would have better performance compared to using a single technique.

2.1.8 Multilingual Hate Speech Detection: Comparison of Transfer Learning Methods to Classify German, Italian, and Spanish Posts [11]

J. Fillies, M. P. Hoffmann and A. Paschke [11] conducted an investigation using seven transformer-based deep learning models to identify the most suitable model for detecting hate speech across monolingual, joint monolingual, and translated to English datasets, employing transfer learning techniques.

Three types of BERT-based transformer models were utilised to identify hate speech in German, Italian, and Spanish texts. Specifically, 'bert-base-german-cased' targeted German-based hate speech, 'dbmdz/bert-base-italian-cased' for Italian-based, and BETO for Spanish-based detection. Subsequently, two BERT-based models analysed the joint multilingual content of German, Italian, and Spanish to identify hate speech. These models included mBERT and XLM-RoBERTa.

Furthermore, the DistilBERT transformer model was applied to translated datasets, with translations conducted separately using GoogleTrans and DeeplTrans. Evaluation of all BERT-based models will be based on Accuracy, F1-Score, and Matthews Correlation Coefficient (MCC), with comparisons drawn against corresponding SVM models.

Model	Dataset	Size	Accuracy	F1-Scor	MCC
GermanBERT	**German**	**4811**	**0.82**	**0.55**	**0.44**
SVM	German	4811	0.81	0.29	0.30
ItalianBERT	**Italian**	**6837**	**0.81**	**0.76**	**0.60**
SVM	Italian	6837	0.78	0.72	0.53
BETO	**Spanish**	**5851**	**0.84**	**0.67**	**0.56**
SVM	Spanish	5851	0.81	0.53	0.44

Figure 2.1.8. 1 Results for the Monolingual approach. Adapted from [11].

Corpus	Model	Accuracy	F1-Scor	MCC
Multilingual	mBERT	0.81	0.66	0.53
Multilingual	**XLM-RoBERTa**	**0.82**	**0.69**	**0.56**
Multilingual	SVM	0.79	0.61	0.48

Figure 2.1.8. 2 Results for the Multilingual Detection Task. Adapted from [11].

Data Corpus	Model	ACC	F1Score	MCC
GoogleTrans	**DistilBert**	**0.80**	**0.67**	**0.53**
GoogleTrans	SVM	0.79	0.58	0.46
DeeplTrans	**Distillbert**	**0.81**	**0.63**	**0.51**
DeeplTrans	SVM	0.79	0.54	0.43

Figure 2.1.8. 3 Results for the Translated-based approach. Adapted from [11].

Figure 2.1.8.1 depicts the testing outcomes of respective BERT-based transformer models on their corresponding monolingual dataset including German, Italian, and Spanish datasets, along with the corresponding SVM model. Figure 2.1.8.2 illustrats the testing results for mBERT, XML-RoBERTa and SVM on the joint multilingual dataset while Figure 2.1.8.3 presents the testing results for DistilBERT and SVM on the separate translation techniques via GoogleTrans and DeeplTrans. Across Figures 2.1.8.1 to Figure 2.1.8.3, it is evident that BERT-based transformer models consistently outperform the SVM model in terms of accuracy, F1-score and MCC across all dataset types, including monolingual, joint multilingual, and translated to English dataset.

The authors raised several challenges, including the potential for incorrect annotations due to misclassification or annotator biases during dataset creation. Additionally, concerns were raised regarding potential information loss during preprocessing and the risk of mistranslation or omission of samples by translation tools, potentially leading to non-hate texts being misclassified as offensive or hateful.

The strengths of this paper were the developed model's multilingual capability and its investigation of various BERT-base transformer models. Investigating different BERT-base transformer models could have provided insights into models to be utilised.

2.1.9 A Scalable Hate Speech Detection System for Vitnamese Social Media using Real-time Big Data Processing and distributed Deep Learning [12]

V. -C. Dinh, T. -D. Vo, M. -P. T. Nguyen and T. -H. Do [12] developed a system to detect hate speech and offensive comments on social networks in real-time, presenting the results on

a dashboard using big data processing and distributed deep learning technology. Apache Kafka and Apache Spark were utilised for big data processing; Apache Kafka was used to partition and store Facebook comments, which were gathered using the Selenium framework, across multiple partitions. This approach enabled diverse consumers, including various devices, to access the comments in different manners, enhancing scalability. Following this, the comments underwent preprocessing and analysis task within Apache Spark.

Multiple machine learning models (Multinomial Naïve Bayes, Logistic Rgression, Decision Tree) and deep learning models (BiLSTM, BiGRU, text-CNN) were trained on the imbalanced Vi-HSD dataset, which contained comments from Facebook posts and YouTube videos. The dataset had significantly more comments labeled as CLEAN compared to those labeled as Hate or Offensive. The models with the highest F1 score and accuracy were selected to be integrated into Spark Structured Streaming in Spark for preprocessing and analysis of real-time hate speech detection. Among these models, text-CNN, a CNN model utilising word embedding techniques, achieved the best performance. It converted 1-D text input to 2-D, where rows represented words in a sentence and columns represent vectors extracted from the FastText word embedding technique. The text-CNN model achieved an F1 score of 61.76% and an accuracy of 86.84%.

The text-CNN model was integrated into Spark Structured Streaming to preprocess the partitioned Vietnamese comments received from Kafka. Next, Spark predicted labels for these comments, identifying whether they are categorized as CLEAN, Hate or Offensive. It then proceeded to analyse the comments, extracting insights such as the date of the comments, user information, and the predicted class of the comments. These analysis results were promptly updated on a dashboard, providing real-time statistics on various metrics including the number of users and the distribution of comments across categories like CLEAN, Hate and Offensive classes.

The strength of this paper was that they integrated several techniques and produced a system that was able to classify the comments in real-time. Additionally, the classified comments were further analysed and visualised in a dashboard, allowing users to easily analyse them. The limitation of this paper was the imbalanced dataset used to train the model, which could have led to better performance on the class with larger instances.

2.1.10 Hate Speech Detection using CNN and BiGRU with Attention Mechanism for Twitter [13]

Q. Sifak and E. B. Setiawan [13] created a hate speech detection model to identify hate speech in the Indonesian language. They integrated CNN and BiGRU with an attention mechanism and utilised the IndoBERT transformer model to extract word embedding vectors. The attention mechanism, introduced with the transformer model, was crucial for focusing the model on important words.

They explored six combinations of integrating CNN and BiGRU with attention: CNN-BiGRU-Attention, CNN-Attention-BiGRU, CNN-Attention-BiGRU-Attention, BiGRU-CNN-Attention, BiGRU-Attention-CNN, and BiGRU-Attention-CNN-Attention. Performance evaluation was conducted using Accuracy, Precision, Recall, and F1-Score.

For the dataset, they crawled Indonesian language tweets form Twitter using Twitter Developer API, employing keywords related to various topics. These tweets were manually labeled as Hate or Non-Hate by three individuals, with the majority vote determining the label. After preprocessing, they obtained a balanced and cleaned dataset consisting of 53, 589 tweets (26,496 Non-Hate and 27,093 Hate).

The dataset was split into training and testing sets using different ratios (7:3, 8:2, and 9:1) to determine the optimal model performance. Additionally, the number of attention heads in the attention mechanism was tuned ro 1, 2, 3, and 6 for optimal results. The testing result of the BiGRU-Attention-CNN-Attention showed an accuracy of 88.12%, surpassing the performance of other models.

The strength of this paper lay in its integration of attention mechanism into the deep learning model, resulting in improved performance. However, a limitation of this paper could have been the potential for incorrect annotations due to misclassification or annotator biases during dataset creation

2.2 Strengths and Weaknesses

Paper No.	Authors	Papers name	Types	Features	Dataset	Techniques	Strengths	Weaknesness
1	X. Zhang et al. [4]	Cyberbullying Detection with a Pronunciation Based Convolutional Neural Network	Cyberbullying	Word pronunciation conversion (eSpeak speech synthesizer)	Twitter, Formspring.me	Pronunciation-based CNNs	Word-to-Pronunciation conversion to enhance the model's ability to comprehend misspelled words.	lack of an established benchmark dataset.
							innovative approaches to address class imbalance issues beyond traditional oversampling and undersampling techniques.	Cannot convert leetspeak word such as 'ugl1'
								pronunciation conversion technique employed may introduce some level of noise into the data

2	M. A. Al-Ajlan and M. Ykhlef [5]	Deep Learning Algorithm for Cyberbullying Detection	Cyberbullying	Word embedding	Twitter by canlling API	CNN-CB	Eliminate feature extraction and apply word embeddings to enable the model to comprehend sentence meanings.	dataset used in the study exhibits a class imbalance issue.
							Outperforms traditional content-based cberbullying detection model	Low in recall between the two algorithms CNN-CB, 73%
3	M. Susanty, Sahrul, A. F. Rahman, M. D. Normansyah, and A. Irawan [6]	Offensive Language Detection using Artificial Neural Network	Offensive language		Web scrabing on several websites	ANN		Human labeling introduces variability and potential inaccuracies.
								Too small dataset to train model

4	Sabur Butt, Noman Ashraf, Grigori Sidorov, and Alexander Gelbukh [7]	Sexism Identification using BERT and Data Augmentation – EXIST2021	Sexism	n-gram, GloVe pretrained word embeddings	Twitter Dataset	Logistic Regression (LR), Multilayer perceptron (MLP), Random Forest (RF), Support Vector Machine (SVM), 1 Dimensional Convolutional Neural Network (1D-CNN), Long short-term memory (LSTM) and BERT	multilingual Taking consideration emojis and repeated special characters in text	Translate from origin to another language and translate back to origin might cause the model to overfit to the dataset

							Further classify sexism into specific type	
5	S. Agrawal and A. Awekar [8]	Deep Learning for Detecting Cyberbullying Across Multiple Social Media Platforms	Bully, Racism, Sexism, Attack	Random, GloVe, SSWE word embedding, t-SNE	Formspring, Twitter, Wikipedia	CNN, LSTM, BiLSTM, BiLSTM with attention	comparison of the performance of deep learning models and the machine learning models t-SNE visualizes words that are most relevant to specific topics	Class imbalance dataset, might reduce model performance
6	J. Wang, K. Fu, and C.-T. Lu [9]	SOSNet: A Graph Convolutional Network Approach to Fine-Grained	Age, Ethnicity, Gender, Religion, Other	Bag of Words (BoW), TF-IDF, word2vec, Glove, fastText, BERT,	6 datasets from different authors and utilise Dynamic Query Expansion using keyword search	SOSNet (Graph Convolutional Network)	Solve class imbance issue with the use of Dynamic Query Expansion	

		Cyberbullying Detection		DistilBERT, Sentence BERT	to address class imbalance issue		Compile 6 dataset from different authors	
7	M. Raj, S. Singh, K. Solanki, and R. Selvanambi [10]	An application to Detect Cyberbullying Using Machine Learning and Deep Learning Techniques	Cyberbullying	GloVe FastText stack, adam optimizer	Hindi, English & Hinglish from open sources	CNN-BiLSTM	multilingual	do not provide an analysis of the performance with the use of a single word embedding technique compared to the use of multiple word embedding techniques
							Hybrid models perform with strength of both model	
							Stack word embedding enhance performance	
8	J. Fillies, M. P. Hoffmann and A. Paschke [11]	Multilingual Hate Speech Detection: Comparison of Transfer Learning Methods to	Hate Speech	GoogleTrans, DeeplTrans	German, Italian, Spanish, Joint dataset, and English translated datasets.	Bert-base-german-cased, bert-base-italian-cased, BETO, mBERT, XLM-	multilingual	Concern of incorrect annotations
							Investigate different bert-base transformer models	

		Classify German, Italian, and Spanish Posts				RoBERTa, DistilBERT		
9	V. -C. Dinh, T. -D. Vo, M. -P. T. Nguyen and T. -H. Do [12]	A Scalable Hate Speech Detection System for Vietnamese social media using Real-time Big Data Processing and distributed Deep Learning	Hate Speech	Apache Kafka, Apache Spark, Selenium, FastText word embedding technique	Vi-HSD	Multinomial Naïve Bayes, Logistic Regression, Decision Tree, BiLSTM, BiGRU, text-CNN	Develop a system to detect hate speech and offensive comments on social media networks in real-time	Imbalanced dataset
							Predicted result visualize in a dashboard that easier for user to perform analysis.	
							Utilise Kafka, Spark to develop a system that can predict hate and offensive on social media in real-time	

10	Q. Sifak and E. B. Setiawan [13]	Hate Speech Detection using CNN and BiGRU with Attention Mechanism for Twitter	Hate Speech	Attention Mechanism, IndoBERT embedding vectors	Indonesian Twitter dataset crawled using Twitter Developer API	CNN-BiGRU-Attention, CNN-Attention-BiGRU, CNN-Attention-BiGRU-Attention, BiGRU-CNN-Attention, BiGRU-Attention-CNN, and BiGRU-Attention-CNN-Attention	Integrate attention mechanism to deep learning models to allow model to know which word is important to focus. Balanced dataset Hybrid deep learning model	Human labeling introduces variability and potential inaccuracies.

Table 2.2. 1 Literature Strengths and Weaknesses comparison.

2.3 Summary on Literature Review

Upon reviewing the literature [4] [5] [7] [8] [10], it was evident that 1-Dimensional Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), and Bidirectional Long Short-Term Memory (BiLSTM) were the most frequently employed deep learning algorithms in the development of cyberbullying detection models. However, these algorithms were predominantly used for detecting a broad spectrum of cyberbullying, and there existed a notable gap in addressing specific types of cyberbullying such as hate speech, body shaming, sexism, and other downstream manifestations. This identified gap underscored the need for further research to enhance the maturity of cyberbullying detection models by specifically targeting and addressing these nuanced forms of harmful online behaviour.

M. Raj, S. Singh, K. Solanki, and R. Selvanambi [10] proposed a hybrid deep learning model that combined CNN and BiLSTM. They utilised the stacking of two word embedding techniques as input for embedding layers, introducing interesting ideas. However, the study solely presented the outcomes of model evaluation using stacked word embedding techniques, omitting the results of the model utilising single word embedding techniques. This omission hindered the illustration of distinctions between stacking and single word embedding techniques, impeding the demonstration that stacking surpassed the use of a single technique. Consequently, there existed a gap in the performance evaluation of single word embedding vectors and the combination of two word embedding vectors. This project aimed to address this gap by comparing single word embedding techniques with the combination of two word embedding techniques.

Moreover, the transformer model had garnered considerable interest across diverse domains of Natural Language Processing (NLP), encompassing text classifications, text generation, and multi-label prediction tasks in recent years. Its popularity was rooted in its ability to grasp the context of individual words within varied sentences, enabling it to simulate human-like cognitive processes.

As previously mentioned, there was a limited number of hate speech detection models developed using transformer architectures. In this specific instance, several pretrained transformer models would be fine-tuned to contribute to the field of hate speech detection, harnessing the capabilities of the transformer model for improved performance in this domain.

While many cyberbullying detection models traditionally use word-based tokenisation, which divides sentences by spaces or punctuation marks, a newer approach has emerged: the subword tokeniser. This tokeniser broke down words into smaller, recognizable units based on patterns learned from the text corpus used to train it. This technique was considered advantageous in NLP tasks because it addressed the challenge of handling unknown or illegible words within sentences. Such words may arise from combining multiple words without clear spacing. Therefore, in this project, a subword tokeniser was chosen for sentence tokenization to enhance the clarity and effectiveness.

Chapter 3

System Methodology/Approach

3.1 System Design Diagram/ Equation

3.1.1 System Architecture Design (CNN)

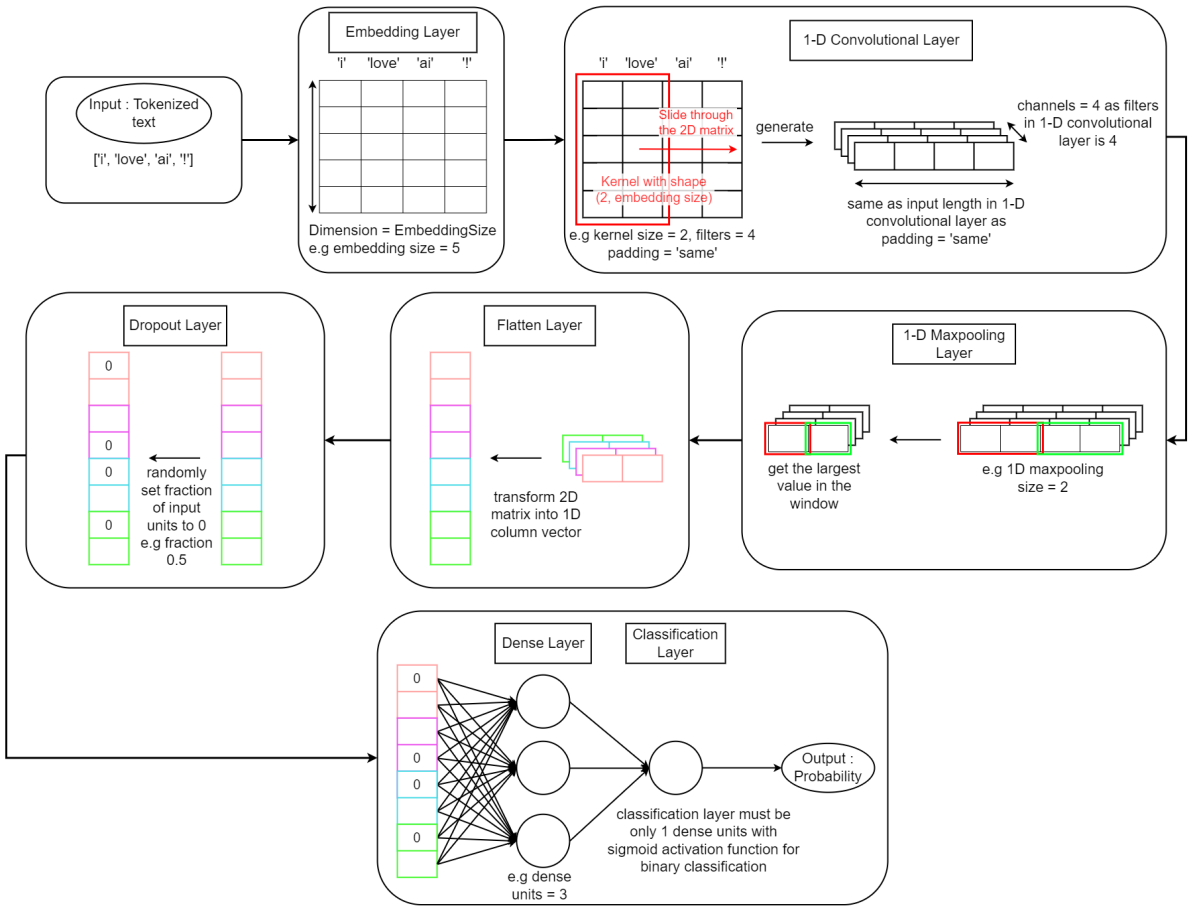


Figure 3.1.1. 1 CNN architecture for text classification.

In Figure 3.1.1.1, the CNN architecture for text classification was illustrated. Within this architecture, the text underwent preprocessing and tokenization into a list of tokens. These tokens were then passed to an Embedding layer, which converted the 1D input data into a 2D matrix with specific dimensionality before undergoing convolution with the kernel to generate features or filters.

The Embedidng layer acted as a dictionary containing vector representations for all unique words in the dataset. These vectors could be obtained from various pretrained word embedding techniques such as Word2Vec or GloVe. The Embedding layer assigned the

available vectors, stored in the embedidng layer with specific dimensionality, to each token in a text, thereby converting the 1D shaped list of tokens into a 2D matrix.

The output of the Embedding layer underwent convolution using a kernel size of 2 with padding = 'same', for example. Padding = 'same' ensured that the output had the same length as the input to the convolutional layer, with a channel size equal to the filter size. The output from the convolutional layer was then subjected to downsampling and feature extraction among filters using a 1D maxpooling layer. The 1D maxpooling layer extracted important features among filters by using a kernel window with a specified size, selecting only the maximum value features in the kernel, and sliding the kernel to the next position without overlapping with the previous kernel window.

The output of the 1D maxpooling layer was then flattened into a 1D column vector using a Flatten layer. This ensured that the shape of the input was compatible with the fully connected layer, also known as the dense layer. After flattening, a dropout layer was applied t avoid overfitting to the training dataset. The dropout layer randomly set a fraction of input units to 0, thereby ignoring some of the extracted features and improving model generalisation. The fraction of dropout was determined by the developer.

Subsequently, the data was fully connected to dense layers, with only a single neuron in the last dense layer using a sigmoid activation function for binary classification. These layers aimed to learn the weights and biases of the neurons to perform well on the training dataset. The last layer with a single neuron generated probabilities for binary classification, and classification was based on the probability threshold. For example, if the threshold was 0.6 and the probability was 0.7, the classification was 1 (positive); otherwise, it is 0 (negative).

$$Sigmoid\ function,\ \sigma(x) = \frac{1}{1+e^{-x}} \qquad (1)$$

where

- x is the weighted input
- e is the Euler's number

3.1.2 System Architecture Design (BiLSTM)

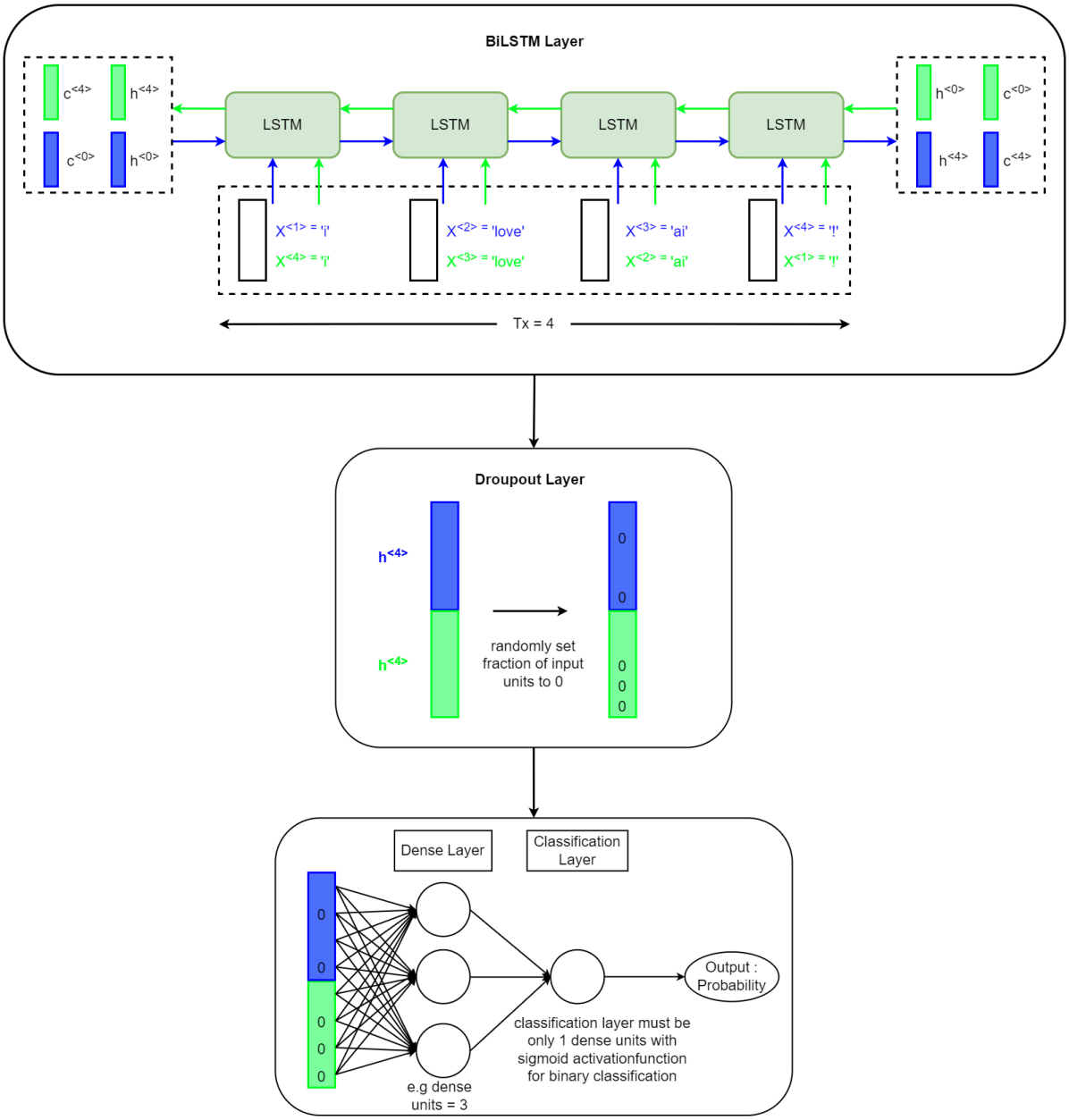


Figure 3.1.2. 1 BiLSTM model architecture.

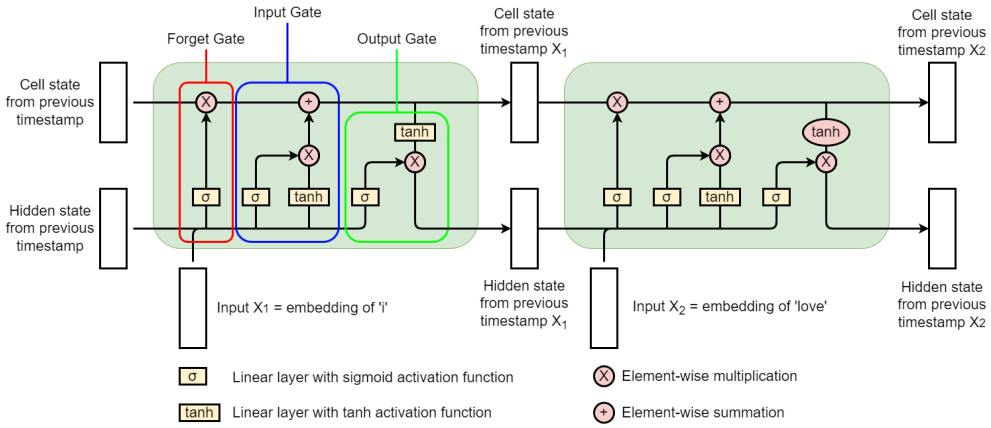


Figure 3.1.2. 2 LSTM cell.

The BiLSTM (Bidirectional Long Short-Term Memory) architecture consisted of an LSTM layer that computed the hidden output of a sequence of input in both forward and backward directions. The LSTM layer sequentially processed the hidden output of the input sequence, moving from the current to the next in a series of LSTM cells, as illustrated in Figure 3.1.2.1. Within each LSTM cell, as depicted in Figure 3.1.2.2, three essential components were present: the forget gate, input gate, and output gate.

The forget gate determined the extent to which information from previous cell states should be retained or forgotten. It achieved this through a linear function with a sigmoid activation function, computing the probability of retention. The input gate controlled the incorporation of information into the cell state from the current input. Meanwhile, the output gate generated the hidden state for the current cell by regulating the information flow from the updated cell state.

Linear layers with sigmoid activation function were used to control the flow of information, while those with tanh activation function ensured that inputs remained within the range of [-1], avoiding infinity values. In Natural Language Processing (NLP) tasks, the input comprised the embedding vector of a token (word), which was then utilised in the forget gate, input gate, and output gate to generate the cell state and hidden state for the current cell. These states were then passed to the next LSTM cell which handled the next sequence within the input sequence.

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (2)$$

where

- x is the weighted input
- e is the Euler's number

The hidden output of the BiLSTM consisted of the stacked hidden outputs in both forward and backward directions, effectively doubling the hidden output of the LSTM layer. Subsequently, the hidden output of the BiLSTM layer was fully connected to dense layers, with the last layer comprising a single neuron with a sigmoid activation function for a binary classification task. This final layer computed the probability of the text belonging to a particular class (hate or non-hate) and made the classification decision based on a specified threshold. If the computed probability exceeds the threshold, the text is classified as hate; otherwise, it is classified as non-hate.

3.1.3 System Architecture Design (DistilBERT)

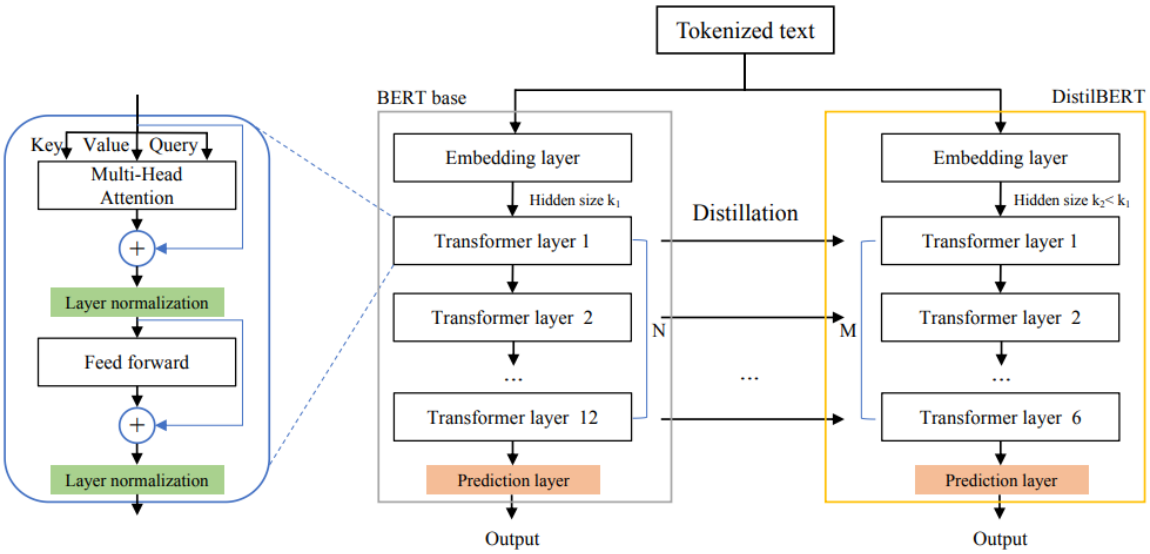


Figure 3.1.3. 1 DistilBERT model architecture comparison with BERT model architecture. Adapted from [14]

DistilBERT is a distilled version of the BERT model designed to reduce computational costs without significantly sacrificing performance. It achieved this by removing certain components such as segment embedidngs and the pooler, and by halving the number of Transformer encoder layers by factor of 2. Despite these modifications. [15] DistilBERT remained a smaller, faster alternative to BERT, maintaining approximately 97% of its language understanding capabilities while being 40% smaller in parameters and 60% faster. Moreover, it was particularly efficient for mobile question-answering applications, being 71% faster than BERT and requiring only 207 MB of weight.

In the DistilBERT model, the input tokens first underwent word embedding and position embedding processes, with the resulting vectors summed in the Embedidng layer. One notable advantage of DistilBERT over models like CNN and BiLSTM was its ability to process input sequences in parallel, leading to faster training. The output of the Embedding layer was then passed through a stack of six transformer encoder layers to extract features for the input tokens. Each transformer encoder comprises components such as the multi-head self-attention block and the feed-forward block.

In the multi-head self-attention block, the model enriched a token's feature by considering the features of other tokens in the same sentence. This was achieved through a computation involving query (Q), key (K), and value (V) vectors, where the dot product of Q and the tranpose of K was divided by the square root of the model's input dimensionality

(d_model) and passed through a softmax function. The resulting attention scores were used to compute a weighted sum of the value vectors, generating an attention feature (A).

$$Attention\ feature, A = Softmax(\frac{QK^T}{\sqrt{d\_model}})V \quad (3)$$

where

- Q is a matrix that contains query vectors for all tokens in an input sentence with a shape of (length of input text, input dimentionality of a model).
- K^T is the transpose of matrix K, which is used as an index for the value vectors V for tokens. Its shape is (input dimentionality of a model, length of input text)
- V is a matrix that contains value vectors representing the information of tokens in a sentence with a shape of (length of input text, input dimentionality of a model)
- d_model represents the input dimensionality of a model

The dot product of Q and the transposed of K computed the similarity between the query (Q) of a token and the key (K) of tokens in a sentence. Dividing with the square root of the model's input dimentionality aimed to normalise the values to avoid large differences in value when the sequence of a token was getting larger. Subsequently, the softmax function applied to the resulting values, computing the probabilities of the current token's relevance to other tokens, with higher relevance receiving higher probabilities. These probabilities were then used to perform multiplication and summation with the value (V) of each token in a sentence and passed through dense layers resulting in a weighted feature. Different attention heads produced different weighted features, which were then concatenated and passed through dense layers to further enrich the token's feature representation, denoted as A.

The feed-forward block further extracted features from the multi-head self-attention layer to produce higher-level representations of tokens. The output of the last encoder layer was then used in dense layers for predictions or classification of a task.

3.1.4 System Architecture Design (RoBERTa)

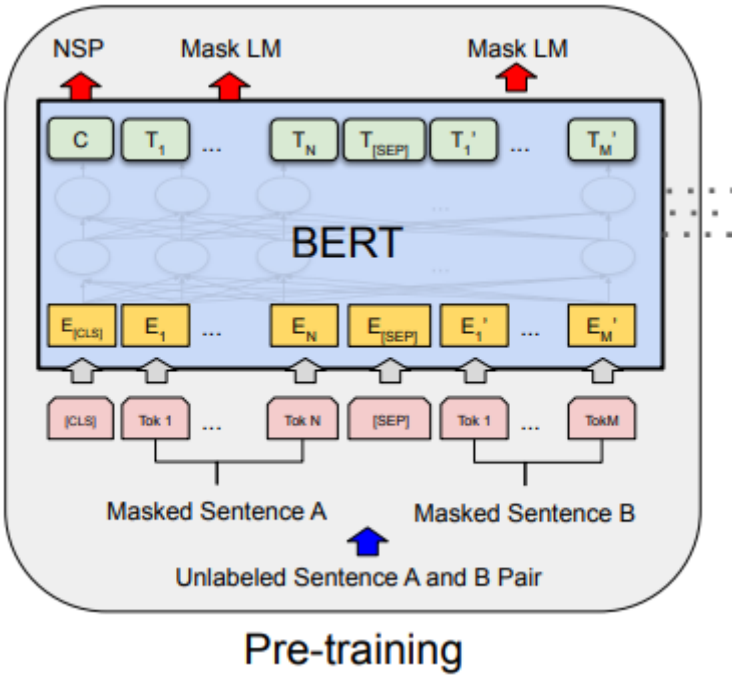


Figure 3.1.4. 1 BERT model architecture. Adapted from [16]

RoBERTa (Robusty optimized BERT approach) is a transformer-based model that builds upon the BERT architecture. It consists of 12 Transformer Encoder Layers, a Hidden Size of 768, and 6 Attention Heads, resulting in a total of 110 million parameters. RoBERTa is designed to improve the performance of BERT models through several key modifications. These include training with a dynamic masking instead of the static masking, utilising full-sentences without the Next Sentence Prediction (NSP) loss, employing large mini-batches, removing the NSP objective, and conducting longer pretraining steps compared to the BERT model. These enhancements aim to improve the model's performance across various natural language processing (NLP) tasks.

'facebook/roberta-hate-speech-dynabench-r4-target' [17] is a specific RoBERTa model designed to provide a high quality, robust and generalised hate speech detection model. It utilises a human-and-model-in the loop process for collecting datasets and training the model. In this process, humans iteratively create datasets and train the model. An important aspect of this process is the role of annotators, who create realistic synthetic data to exploit any weaknesses identified in the model through real-time feedback. They perturb half of the dataset created in each iteration, introducing noise, and train the model with these perturbed samples to enhance its robustness and generalisation in detecting hate speech.

Chapter 4

System Design

4.1 System Block Diagram

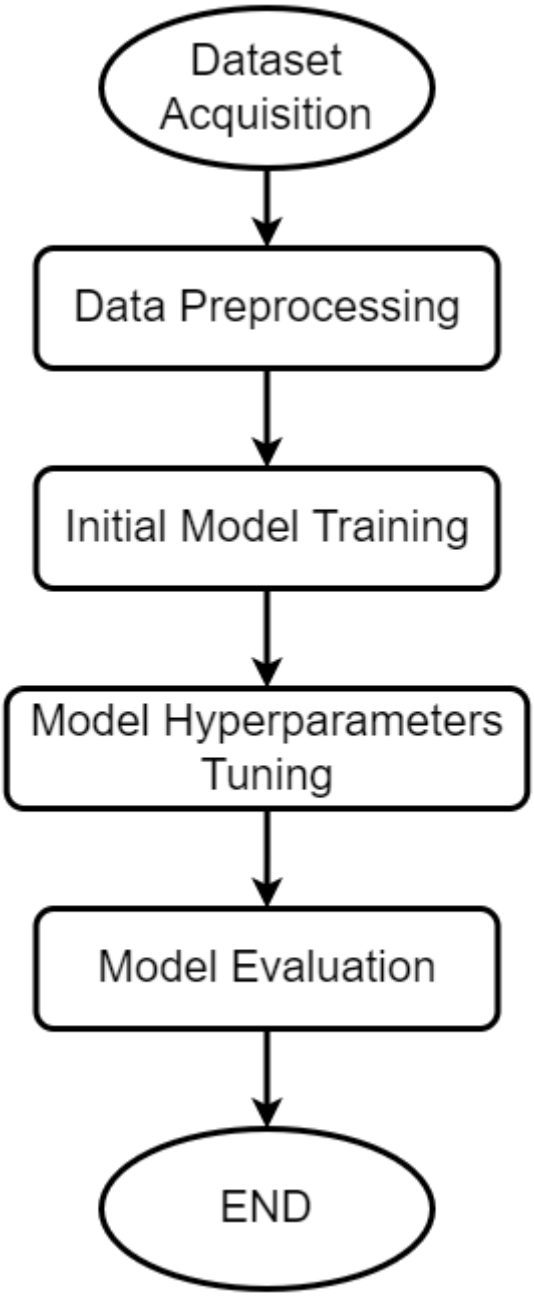


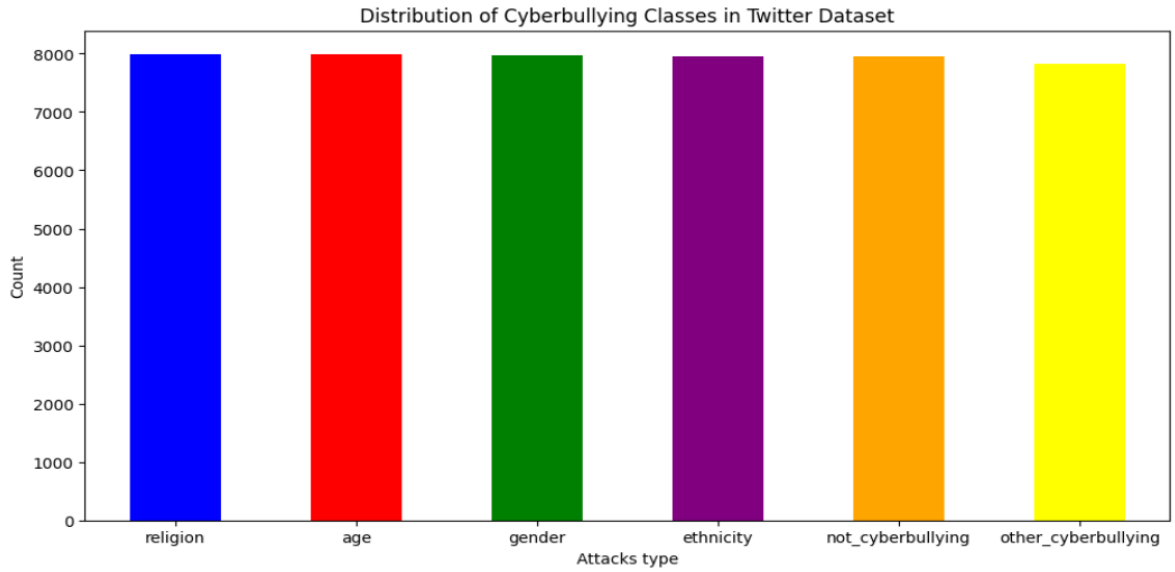
Figure 4.1. 1 System Block Diagram for this project.

Figure 4.1.1 illustrated the system block diagram for this project. There were 5 main steps included in this project: Step 1 Dataset Acquisition, Step 2 Data Preprocessing, Step 3 Initial Model Training, Step 4 Model Hyperparameters Tuning, and Step 5 Model Evaluation. In Step 1, the dataset used was introduced and acquired. In Step 2, the data in the dataset was preprocessed to remove useless information. In Step 3, the initial model was trained with the preprocessed data using randomly initialised hyperparameters. In Step 4, the initial model was hypertuned to improve the model's performance, and Step 5 was to evaluate the hypertuned model's performance. All the details of these steps were discussed in Chapter 4.2.1, 4.2.2, 4.2.3, 4.2.4, and 4.2.5 repsectively.

4.2 System Components Specifications

4.2.1 Dataset Acquisition

The dataset used in this project was downloaded from Kaggle.com, consisting of English tweets with 6 classes, which were cyberbullying based on Age, Ethnicity, Gender, Religion, Other Cyberbullying, and Not Cyberbullying. Each class contained around 8,000 tweets, resulting in a total of 47,000 tweets. This dataset was curated by J. Wang, K. Fu, and C.-T. Lu [9], who compiled 6 datasets from different authors to form a new Twitter cyberbullying dataset with a class-imbalanced dataset. The dataset was expanded using Dynamic Query Expansion, a data mining technique that utilised the GetOldTweets3 library to expand the size of the dataset. Subsequently, around 8,000 tweets were randomly selected from each class to create a class-balanced dataset. The dataset was downloaded as a CSV file and loaded into a pandas dataframe. The distribution of cyberbullying classess in the dataset was illustrated in Figure 4.2.1.1.




```
Total number of text that label as religion attack = 7998
Total number of text that label as age attack = 7992
Total number of text that label as gender attack = 7973
Total number of text that label as ethnicity attack = 7961
Total number of text that label as not cyberbullying = 7945
Total number of text that label as other attack = 7823
```

Figure 4.2.1. 1 Distribution of Cyberbullying Classes in Twitter Dataset.

4.2.2 Dataset Preprocessing

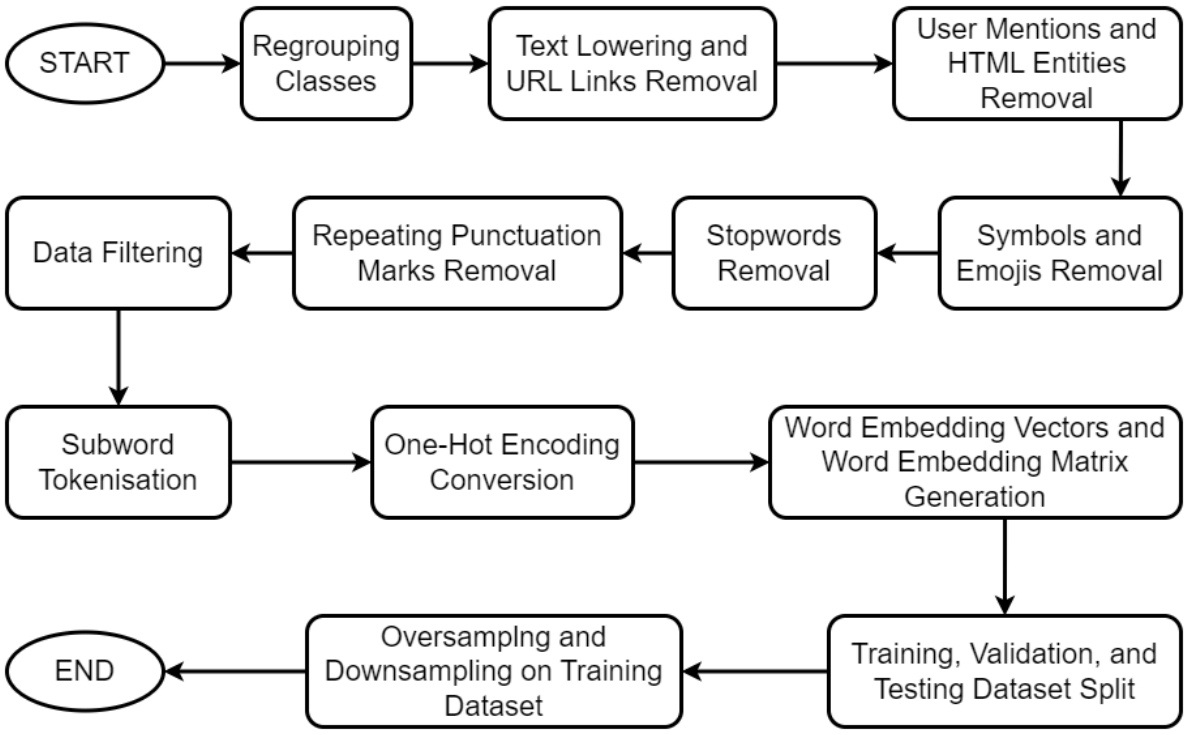
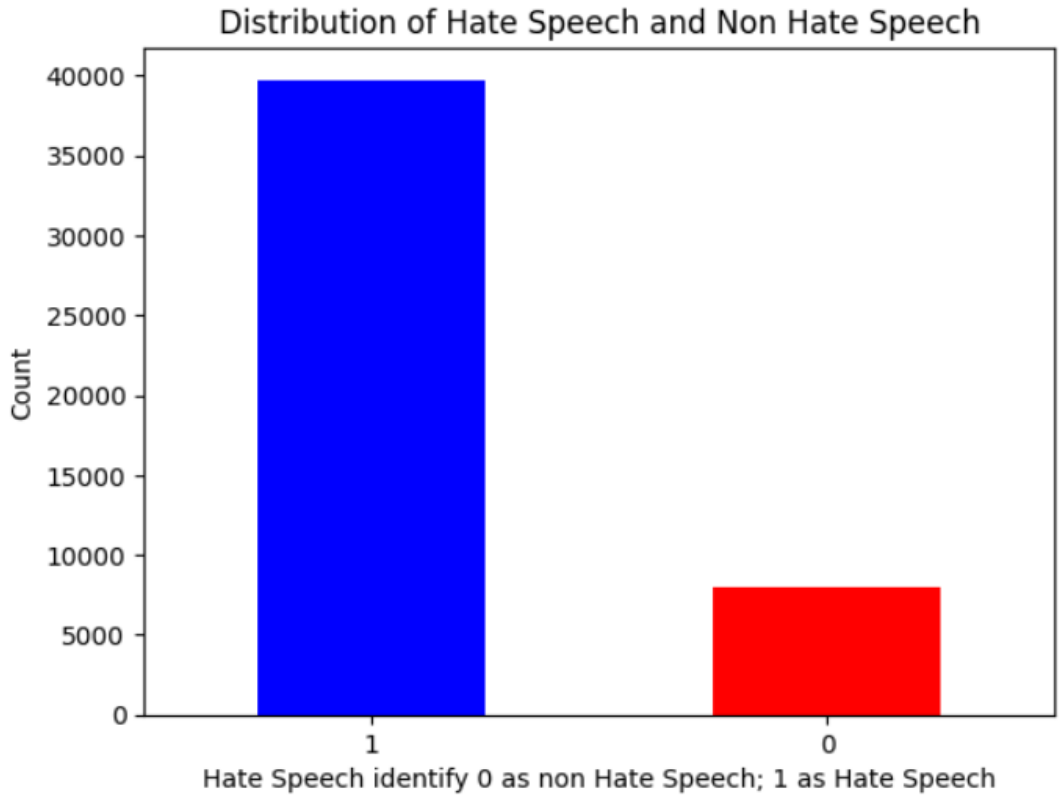


Figure 4.2.2. 1 The flow of Data Preprocessing.

Several preprocessing techniques were conducted, as shown in Figure 4.2.2.1. These include regrouping classes, converting text to lowercase and removing URL links, user mentions and Hypertext Markup Language (HTML) entities removal, symbols and emojis removal, stopwords removal, and repeating punctuation marks removal, as well as data filtering. Following this, subword tokenisation was performed to tokenise the text into smaller subwords. The unique words in the tokenised tweets were identified, and one-hot encoding conversion was applied. Subsequently, embedding vectors for the unique tokenised words and the word embedding matrix were generated. Finally, the dataset was split into training, validation, and testing datasets. Then, oversampling or downsampling techniques were applied only to the instances of the training dataset.

4.2.2.1 Regrouping Classes

In this project, tweets labelled as attacks based on Age, Gender, Religion, Ethnicity, and Other Cyberbullying were grouped as Hate Speech (labelled '1'), while tweets labelled as Not Cyberbullying were classified as Non-Hate Speech (labelled '0'). This classification was based on the understanding that these attacks may evoke animosity from the victims towards the aggressor. In contrast, Not Cyberbullying tweets were considered as normal text that is unlikely to elicit negative emotions in the receiver. The distribution of hate speech and non hate speech is illustrated in Figure 4.2.2.1.1.




```
Total number of text that label as non hate = 7945
Total number of text that label as hate = 39747
```

Figure 4.2.2.1. 1 The distribution of Hate Speech and Non-Hate Speech.

4.2.2.2 Text Lowering and URL Links Removal

Firstly, all the text in the tweet underwent text lowering to convert all the text to lowercase. Following that, URLs in the tweets were removed since they do not contribute much to indicating hate speech. This was achieved using a regular expression. The pattern used in the regular expression identified words in tweets that started with ‘http://’, ‘https://’, ‘www.’, or ‘bit.ly’. Words that matched this pattern were replaced with an empty space, effectively removing the URLs from the tweets. An example of URL removal is shown in Figure 4.2.2.2.1, and the highlighted region in the figure demonstrated the changes made after performing URL removal.

Sample data before URL removal

	clean_text	hate_speech
10	@jord_is_dead http://t.co/usqinyw5gn	0
11	the bully flushes on kd http://twitvid.com/a2tnp	0
12	ughhhh #mkr	0
13	rt @kurdsnews: turkish state has killed 241 ch...	0
14	love that the best response to the hotcakes th...	0

Sample data after URL removal

	clean_text	hate_speech
10	@jord_is_dead	0
11	the bully flushes on kd	0
12	ughhhh #mkr	0
13	rt @kurdsnews: turkish state has killed 241 ch...	0
14	love that the best response to the hotcakes th...	0

Figure 4.2.2.2. 1 Example of URL removal. E.g. 'http://t.co/usqinyw5gn' and 'http://twitvid.com/a2tnp' were removed.

4.2.2.3 User Mentions and HTML Entities Removal

To remove user mentions and HTML entities from the tweets, regular expressions are also employed. To achieve this, words that matched the patterns '@\w+' or '&\w+;', were removed. User mentions typically start with '@', as shown in the example highlighted in yellow in Figure 4.2.2.3.1. Additionally, HTML entities usually follow a pattern of starting with '&' and ending with ';', as demonstrated in the example of HTML entity removal highlighted in blue in Figure 4.2.2.3.1.

Sample data before user mention and html entity removal

	clean_text	hate_speech
20	@halalcunty @biebervalue @liamxkiwi @greenline...	0
21	kids love😘❤ @ mohamad bin zayed city 0	مدينة محم...
22	i still have jack, amsterdam, ciroc, crown, bu...	0
23	@scottyswaggod men are the ones that are going...	0
24	wishing my arena partner was on. >.> re...	0

Sample data after user mention and html entity removal

	clean_text	hate_speech
20	i know saudis chased girls into a burning...	0
21	kids love😘❤ @ mohamad bin zayed city 0	مدينة محم...
22	i still have jack, amsterdam, ciroc, crown, bu...	0
23	men are the ones that are going to push the r...	0
24	wishing my arena partner was on. . really wan...	0

Figure 4.2.2.3. 1 Example of user mention and HTML entity removal. E.g. '@halalcunty', '@biebervalue', '>' were removed.

4.2.2.4 Symbols and Emojis Removal

Moreover, emojis and other symbols within the tweet, such as thumbs up, tree symbols, moon symbols and others, were removed using a regular expression designed to capture a wide range of emoji and symbol in Unicode. The removal of emojis and other symbols aids in standardising the text, allowing for a focus on linguistic content. This ensures that the deep learning detection model will not be misled by the presence of these elements. An example of emojis and other symbols removal is illustrated in Figure 4.2.2.4.1 with highlighted elements. The blowing kiss emoji and black heart symbol in the example were removed after applying the regular expression designed for emojis and symbols removal.

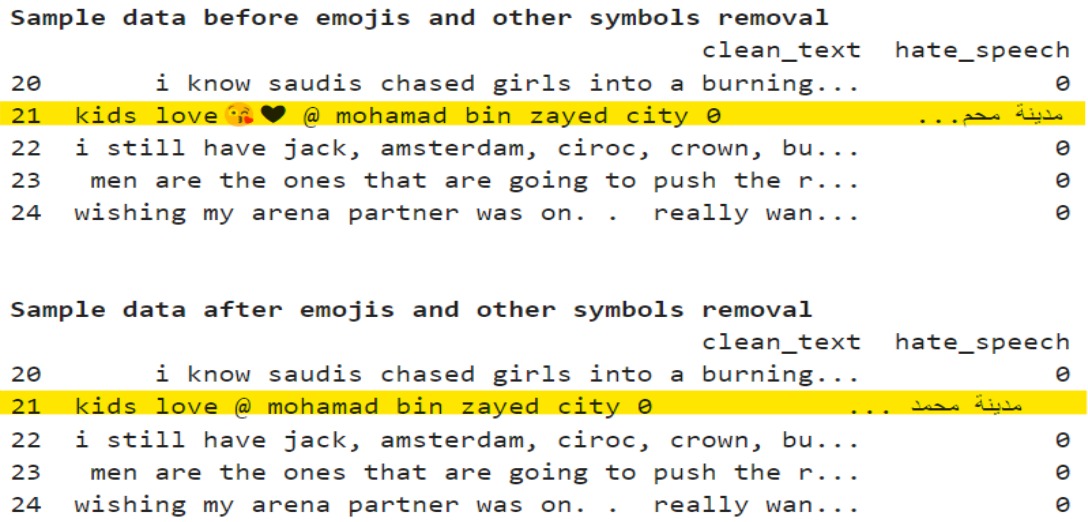


Figure 4.2.2.4. 1 Example of emojis and other symbols removal. E.g. The blowing kiss emoji and black heart symbol were removed.

4.2.2.5 Stopwords Removal

After removing emojis and symbols, the tweet text underwent stopwords removal, a process aimed at eliminating words that contribute little meaning to reduce the text length. To achieve this, the tweet text needed to be tokenised into a list of words using the 'split()' function. Subsequently, the words in the list were compared with the stopwords dictionary provided by the 'nltk.corpus' library. If a word was identified as a stopword, no action is taken. Conversely, if the word is not a stopword, it was included in a new list, and the words in the new list were joined to form a new string, which was then returned.

Figure 4.2.2.5.1 illustrated the results of stopwords removal, highlighting the removal of common stopwords such as 'i', 'into', and 'a' from the samples after the stopwords removal process.

```
Sample data before stopwords removal
                                                clean_text  hate_speech
20        i know saudis chased girls into a burning...                0
21  kids love @ mohamad bin zayed city 0              ... مدينة محمد
22  i still have jack, amsterdam, ciroc, crown, bu...                0
23   men are the ones that are going to push the r...                0
24  wishing my arena partner was on. .  really wan...                0


Sample data after stopwords removal
                                                clean_text  hate_speech
20          know saudis chased girls burning building.                0
21  kids love @ mohamad bin zayed city 0              ... مدينة محمد
22  still jack, amsterdam, ciroc, crown, bud light...                0
23  men ones going push real change. ones power, g...                0
24  wishing arena partner on. . really want get pv...                0
```

Figure 4.2.2.5. 1 Example of stopwords removal. E.g. 'i', 'into', and 'a' were removed from the highlighted text.

4.2.2.6 Repeating Punctuation Marks Removal

Following stopwords removal, the tweet text underwent a process of removing repeating punctuation marks to eliminate redundant characters within the text. This step aimed to prevent excessive text length caused by the repeated characters. To accomplish this, a regular expression 'r'([^\w\s])\1+' was employed to identify repeating punctuation marks. The function of this regular expression was to detect instances where non-space characters were repeated, and these repetitions were then removed, retaining only a single character, as demonstrated in Figure 4.2.2.6.1.

```
Sample data before removing repeating punctuation mark
This is a sample.... of repeating^^^^ punctuation mark$$$*****


Sample data after removing repeating punctuation mark
This is a sample. of repeating^ punctuation mark$*
```

Figure 4.2.2.6. 1 Example of repeating punctuation mark removal.

4.2.2.7 Data Filtering

Furthermore, the tweet text underwent a regular expression process to ensure it only contains alphanumeric characters in words. Non-alphanumeric characters were replaced with

an empty space during this step. Subsequently, the text length of the cleaned tweets was calculated by splitting the string text into a list of words and determining the length of the list. Tweets with a length of 0 are removed. The text length in the dataset was then visualised for further analysis to select a suitable maximum length for the text. Figure 4.2.2.7.1 illustrated the visualisation of the cleaned text lengths in the dataset.

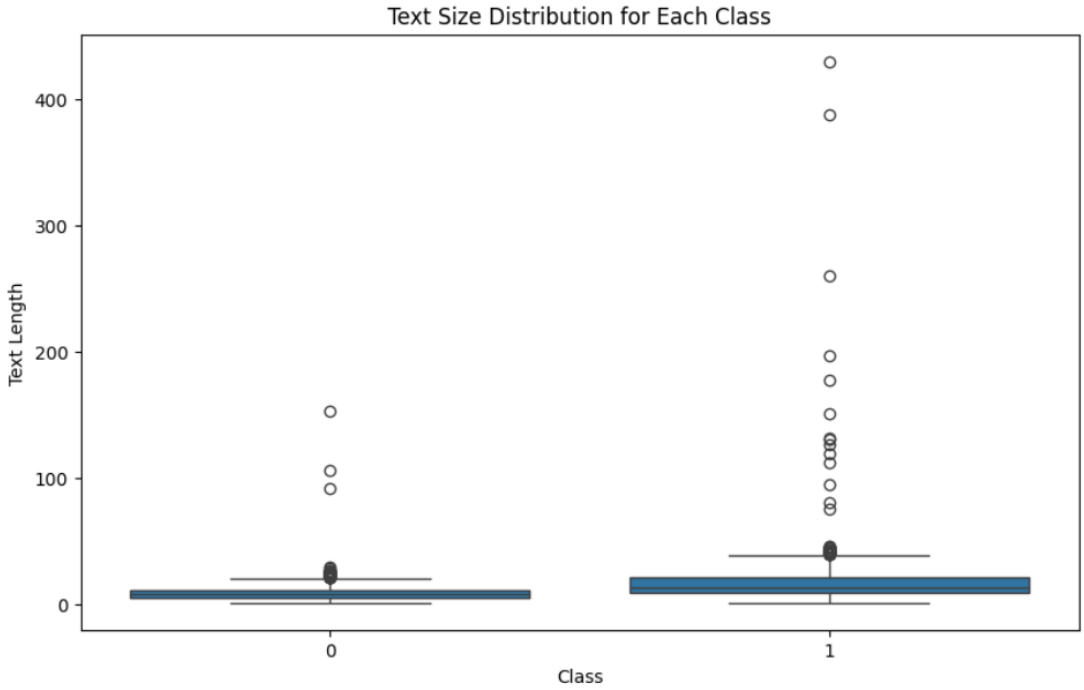


Figure 4.2.2.7. 1 Text length distribution for each class.

Figure 4.2.2.7.1 clearly indicates that the majority of tweets have a length of around 100, with only a small number exceeding this threshold. To streamline the max length of tweets, all tweets surpassing a length of 100 were discarded. The decision to drop tweets, rather than truncating them to a size of 100, stemmed from the concern that truncated tweets might omit crucial information, potentially leading to inaccurate representations of hate or non-hate content. The preference for dropping ensured a more accurate and reliable dataset for model training.

4.2.2.8 Subword Tokenisation

After dropping the tweets that have a text length of more than 100, the tweet underwent sub-word tokenisation to break them into smaller word pieces. Sub-word tokenisation involved breaking a word into smaller subwords based on patterns observed in the training corpus. This allowed the model to process words it had never seen before. In this project, a pretrained

BertTokeniser from the Hugging Face Transformers library, named ‘bert-base-uncased’, was used for sub-word tokenisation for the CNN model and BiLSTM model, while the DistilBERT transformer model and ‘facebook/roberta-hate-speech-dynabench-r4-target’ used their own tokenizer. ‘bert-base-uncased’ is a tokeniser by the Bidirectional Encoder Representations from Transformers (BERT) trained on a large corpora of text, and it is capable of handling both uppercase and lowercase tokens.

For example, as shown in Figure 4.2.2.8.1, the word ‘mohamad’ was an unseen word in the pretrained sub-word tokeniser. The sub-word tokeniser then further chunked the word into several well-known subwords, resulting in ‘mo’, ‘##ham’, ‘##ad’. The presence of the ‘##’ prefix typically indicated that the subword followed another subword in the original word. The text length of the tweets was then calculated after performing subword tokenisation. This could be useful when padding the tweets text length to the same length.

Sample data before sub-word tokenization

	tweet_text	hate_speech
20	kids love @ mohamad bin zayed city 0 ... مدينة محمد	
21	i still have jack, amsterdam, ciroc, crown, bu...	0
22	men are the ones that are going to push the r...	0
23	wishing my arena partner was on. . really wan...	0
24	education nation: bullying | turn to 10	0

Sample data after sub-word tokenization

	tweet_text	hate_speech
20	[kids, love, @, mo, ##ham, ##ad, bin, za, ##ye...	0
21	[i, still, have, jack, ,, amsterdam, ,, ci, ##...	0
22	[men, are, the, ones, that, are, going, to, pu...	0
23	[wishing, my, arena, partner, was, on, ., ., r...	0
24	[education, nation, :, bullying, |, turn, to, 10]	0

Figure 4.2.2.8. 1 Example of subword tokenisation.

4.2.2.9 One-Hot Encoding Conversion

After performing subword tokenisation on the dataset, all the tokenised tokens were obtained by iterating through the tokenised lists for each row. The ‘set()’ function was applied to extract all unique tokens, eliminating any duplicates. Subsequently, each unique token in the set was assigned a distinct index, starting from 1, as depicted in Figure 4.2.2.9.1. This index assignment was accomplished using the ‘enumerate()’ function, with the understanding that 0 would be reserved for padding the text to ensure a consistent length.

This process led to the creation of a dictionary containing 19,500 unique tokens, indicating the number of distinct tokens identified after subword tokenisation.

Following that, the dictionary of unique tokens was utilised to perform one-hot encoding conversion to convert the words in the dataset into numerical forms. This transformation involved finding the index of each unique token in the dictionary. As a result, the tweet text was transformed into an array of numerical tokens, as depicted in Figure 4.2.2.9.2.


```
{'##rting': 1, 'mahmoud': 2, 'resolve': 3, 'rivalry': 4, 'serials': 5, '##abas': 6, 'red': 7, 'designer': 8, 'best': 9, 'pairs': 10,
'##eira': 11, '##mba': 12, 'takahashi': 13, '##ka': 14, '3': 15, '##bain': 16, 'scholars': 17, 'stares': 18, 'ngos': 19, '##iman': 2
0, 'actresses': 21, '##bbs': 22, 'shepherd': 23, 'sentences': 24, 'monument': 25, 'roach': 26, '##kt': 27, 'ia': 28, '##yra': 29, 'c
ouch': 30, 'blanchard': 31, 'powder': 32, '##room': 33, 'anymore': 34, 'credibility': 35, '##nia': 36, 'intersection': 37, 'stumble
d': 38, 'layout': 39, 'line': 40, 'intimidating': 41, '##heard': 42, '##dded': 43, 'medicare': 44, 'apologized': 45, '60s': 46, 'den
mark': 47, 'wishes': 48, 'intersecting': 49, 'dollar': 50, 'averaged': 51, 'stayed': 52, 'relations': 53, '##ners': 54, 'sanctione
d': 55, 'behave': 56, '##quist': 57, 'extends': 58, '##jan': 59, 'mca': 60, 'cisco': 61, 'infamous': 62, 'abandonment': 63, 'drippin
g': 64, 'id': 65, 'minutes': 66, 'advertising': 67, 'rehearsal': 68, 'flames': 69, '##gau': 70, 'reflects': 71, 'hunting': 72, '##ho
od': 73, 'body': 74, 'entire': 75, 'accumulated': 76, '##gon': 77, 'convicts': 78, 'labelled': 79, '##wee': 80, '##base': 81, 'blac
k': 82, 'survives': 83, 'easy': 84, 'overdose': 85, 'happiness': 86, 'impacted': 87, 'tanned': 88, 'characters': 89, 'spontaneousl
y': 90, 'maxim': 91, 'localities': 92, 'pending': 93, 'ostensibly': 94, '##chi': 95, '##ady': 96, 'rome': 97, '##ste': 98, 'karach
i': 99, 'commence': 100, 'bounce': 101, '##gh': 102, '##late': 103, '##ries': 104, '1998': 105, 'plot': 106, 'does': 107, '##zo': 10
8, 'everyone': 109, 'earn': 110, 'augustus': 111, 'reason': 112, 'suspect': 113, 'blushing': 114, 'finished': 115, 'senator': 116,
```


Figure 4.2.2.9. 1 Example of creating a dictionary contains all unique tokens and the respective index.


```
   text_length                                      tokenized_text  \
0            5  [words, #, kata, ##nda, ##nd, ##re, ,, food, c...
1           11  [#, aus, ##sie, ##tv, white, ?, #, mk, ##r, #,...
2            5  [class, ##y, whore, ?, red, velvet, cup, ##cak...
3           10  [me, ##h, ., p, thanks, heads, up, ,, concerne...
4            8  [isis, account, pretending, kurdish, account, ...

                                numerical_tokens
0  [4141, 2312, 5372, 5011, 13449, 4534, 3100, 12...
1  [2312, 201, 4359, 9105, 12877, 12665, 2312, 16...
2  [5530, 15362, 3885, 12665, 7, 17146, 10283, 15...
3  [4583, 7976, 11032, 12885, 1589, 2085, 16748, ...
4  [14332, 14280, 14873, 2609, 14280, 11032, 1036...
```


Figure 4.2.2.9. 2 Example of converting the tokens in numerical form.

4.2.2.10 Word Embedding Vectors and Word Embedding Matrix Generation

After acquiring all the unique tokens along with their respective indices, the subsequent step involved obtaining word embedding vectors for each unique token. Various pretrained word embedding techniques were employed, including GloVe, Word2Vec, GloVe+Word2Vec (by stacking both vectors), and GloVe+Word2Vec (by computing the mean of both vectors),

as well as GloVe+Word2Vec (by computing the root mean square of both vectors). However, for the DistilBERT and RoBERTa models utilised in this project, they had their own tokenizer and word embedding vectors for tokenised tokens. Therefore, acquiring unique tokens and their respective indices could be skipped, as well as the word embedding extraction. The additional steps for DistilBERT and RoBERTa models involved extracting the attention mask for each input, attention mask identified the tokens in an input that need to be focused, so that those tokens with spaces or padding will not be considered. The output of the attention mask was shown in Figure 4.2.2.10.1, where '1' in the attention mask indicated tokens that needed to be focused on, while '0' could be ignored.

```
train_inputs[1]

tensor([    0, 20982,   236,   989, 11751,   443,     4,   277,  9802,   531,
          156,     4,  2649,  3537,   203,     4,     2,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1])
```

```
train_masks[1]

tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Figure 4.2.2.10. 1 Attention mask for an input sequence.

- **Word2Vec:**

Word2Vec is a family of models for generating word embeddings, developed by T. Mikolov, Ilya Sutskever, K. Chen, G. S. Corrado, and J. Dean [18]. The model used in this project was "word2vec-google-news-300", a pretrained word2vec model with 300 dimensions. It was trained in a supervised manner on a massive dataset containing news, predicting either current tokens with the context or tokens preceding and succeeding the current tokens.

- **GloVe:**

Global Vectors for Word Representation (GloVe) is an unsupervised learning algorithm developed by J. Pennington, R. Socher, and C. Manning [19]. Its objective is to derive vector representations for words through the analysis of their co-occurrence statistics in large corpora. If two words frequently appeared together, their co-occurrence probability was higher. For instance, in Figure 4.2.2.10.2, 'ice' occurred more frequently with 'solid' compared to its occurrence with 'gas', reflecting the higher probability of 'ice' occurring with 'solid'. The co-occurrence matrix was then utilised to learn word vectors through an optimization process, minimising the difference between predicted and actual co-occurrence probabilities.

The GloVe model used in this project was the pretrained Common Crawl word embedding vectors, which encompassed 42 billion tokens and a vocabulary of 1.9 million. These vectors were available for download from https://nlp.stanford.edu/projects/glove/.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 4.2.2.10. 2 Example probabilities of words. Adapted from [19]

- **GloVe + Word2Vec (stack):**

GloVe + Word2Vec (stack) was a technique that formed larger-dimensional word embedding vectors for a word by concatenating the 300-dimensional word vectors obtained from Word2Vec and GloVe word embedding techniques. This involved obtaining word vectors for a word using Word2Vec and GloVe, resulting in two 300-dimensional word embedding vectors. These vectors were then combined by horizontally appending one to the other, resulting in a larger vector with 600 dimensions. This larger word vector aims to capture more meaningful information and potentially more complex relationships between words.

```
Sample word embedding vectors for GloVe and Word2Vec
GloVe vector for word 'crazy':  [-0.53923  -0.094882 -0.40186  -0.055611 -0.039816]
Word2Vec vector for word 'crazy':  [ 0.03063965 -0.01916504  0.03881836  0.22265625 -0.01745605]


Sample concatenating of GloVe and Word2Vec word embedding vectors
Stacked vector for word 'crazy' by concatenation:  [-0.53923    -0.094882   -0.40186    -0.055611   -0.039816    0.03063965
 -0.01916504  0.03881836  0.22265625 -0.01745605]
```

Figure 4.2.2.10. 3 Example of stacking of GloVe and Word2Vec word embedding techniques.

Figure 4.2.2.10.3 illustrated an example demonstrating the creation of a stacked word embedding vector generated from GloVe and Word2Vec vectors. To simplify the visualisation of the concatenation of the two word embedding vectors, only the first 5 elements of each vector were shown in the figure.

- **GloVe + Word2Vec (mean):**

GloVe + Word2Vec (mean) was a technique that created a 300-dimensional word embedding vector by averaging vectors obtained from two distinct word embedding techniques – GloVe and Word2Vec, each with a size of 300 dimensions. Taking the average was believed to leverage the training on diverse datasets, resulting in a vector that was considered more reliable and versatile. For example, in Figure 4.2.2.10.4, the embedding vector for the word 'canisy' was present in the GloVe word embedding techniques, however, it was not present in the Word2Vec word embedding technique dictionary. This showed that combining different word embedding techniques helped in resolving the issue of non-existent words in a word embedding technique and prevented the occurrence of all-zero vectors. This could help reduce the traning time for fine-tuning the word embedding vectors for words and improve the model performance. This involved obtaining word vectors for a word using GloVe and Word2Vec, resulting in two 300-dimensional word embedding vectors. The average of these two vectors was then calculated using the 'np.mean' in the NumPy library, computing column-wise. This resulted in an averaged word embedding vector with 300 dimensions.

```
Sample word embedding vectors for GloVe and Word2Vec
GloVe vector for word 'canisy':
 [ 0.15982    0.0073894  0.32268   -0.048407   0.025101  -0.14531
  0.70945   -0.34261    0.10779    0.12948  ]

Word2Vec vector for word 'canisy':
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Figure 4.2.2.10. 4 Vectors for word 'canisy' from GloVe and Word2Vec.

```
Sample word embedding vectors for GloVe and Word2Vec
GloVe vector for word 'crazy':  [-0.53923  -0.094882 -0.40186  -0.055611 -0.039816]
Word2Vec vector for word 'crazy':  [ 0.03063965 -0.01916504  0.03881836  0.22265625 -0.01745605]

Sample averaging of GloVe and Word2Vec word embedding vectors
Mean vector for word 'crazy':  [-0.25429517 -0.05702352 -0.18152082  0.08352263 -0.02863603]
```

Figure 4.2.2.10. 5 Example of computing the of GloVe and Word2Vec word embedding techniques.

Figure 4.2.2.10.5 illustrated the process of obtaining a mean word embedding vector from GloVe and Word2Vec vectors. To enhance the clarity, only the first five elements of each vector were presented in the figure. The visualisation simplified the averaging process, where these truncated vectors were averaged elementwise. The resulting mean word embedding vector is 300-dimensional, calculated by averaging corrsponding elements of the two vectors,

$$mean[i] = \frac{GloVe[word_i] + Word2Vec[word_i]}{2} \qquad (4)$$

where

- i is the index of a word
- $word_i$ is the word at index i
- $GloVe[word_i]$ and $Word2Vec[word_i]$ are the vectors obtained from GloVe and Word2Vec embedding techniques

- GloVe + Word2Vec (Rooted Mean Squared):

GloVe + Word2Vec (Rooted Mean Squared) was a technique that created a 300-dimensional word embedding vectors by computing the mean of the squared values of the two vectors obtained from GloVe and Word2Vec, followed by computing the square root of the result. This technique aimed to investigate the model's performance by transforming the vectors into a dimensional space with positive values. The average of the squared values of these two vectors was then calculated using the 'np.mean' in the NumPy library, computing column-wise. This resulted in an averaged word embedding vector with 300 dimensions, with the square root applied to the resulting vector.

$$rms[i] = \sqrt{\frac{(GloVe[word_i])^2 + (Word2Vec[word_i])^2}{2}} \qquad (5)$$

where

- i is the index of a word

- $word_i$ is the word at index i
- $GloVe[word_i]$ and $Word2Vec[word_i]$ are the vectors obtained from GloVe and Word2Vec embedding techniques

To initiate the process, a 2D numpy zeros matrix is created with a size of 19500+1 rows and 300 columns except for stacking of two word embedding techniques utilised 600 columns. The reason for having 19500+1 rows is that the unique tokens' index starts from 1, unlike the 0-based indexing of a 2D matrix. Each row in the matrix corresponds to a unique token, and the matrix has 300 columns to accommodate the 300-dimensional vectors obtained from the pretrained word embedding model.

Next, the index of the unique tokens serves as the index of the 2D matrix, and the corresponding value in that index is the 300-dimensional vector obtained from the pretrained word embedding model. If a unique token is not present in the pretrained word embedding model, the matrix entry for that token will be a 300-dimensional numpy zero vector.

4.2.2.11 Training, Validation and Testing Dataset Split

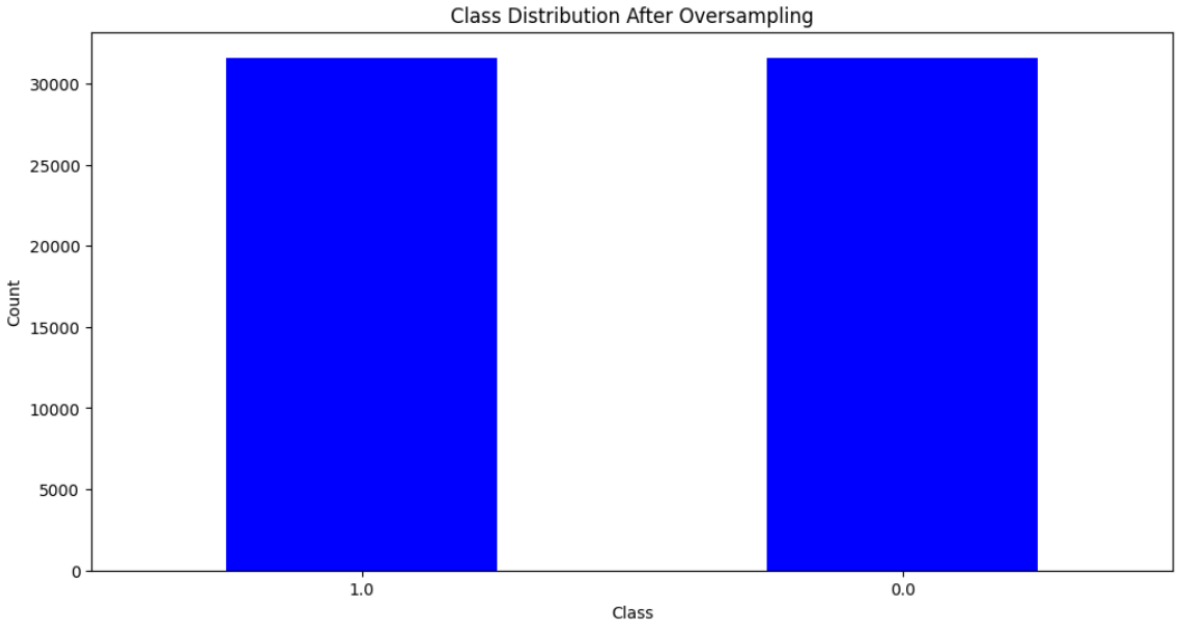
The dataset size was reduced to 47,385 tweet texts, comprising 39,538 instances of Hate Speech and 7,847 instances of Non-Hate Speech, as depicted in Figure 4.2.2.15. Subsequently, the input X, representing numerical values for the tweet texts, and the output y, representing hate speech labels, were established. X and y were then split into training, validation, and testing sets using the 'train_test_split' function from the 'sklearn.model_selection' library, with a split ratio of 0.8 for training, resulting in X_train and y_train, 0.1 for validation, resulting in X_val and y_val, and 0.1 for testing, resulting in X_test and y_test. This resulted in a class-imbalanced dataset with 37,908 tweet texts (31,599 Hate, 6,309 Non-Hate) in the training set, 4739 tweet texts (3,967 Hate, 772 Non-Hate) in the validation set, and 4,738 tweet texts (3972 Hate, 766 Non-Hate) in the testing set.

4.2.2.12 Oversampling and Downsampling on Training Dataset

Grouping multiple attacks into a single class has led to a problem of class imbalance in the dataset. The portion of Hate Speech (labelled '1') is five times greater than Non-Hate Speech (labelled '0'). To address the class imbalance issue in the training set, oversampling and undersampling techniques were applied solely on the training set to prevent bias towards

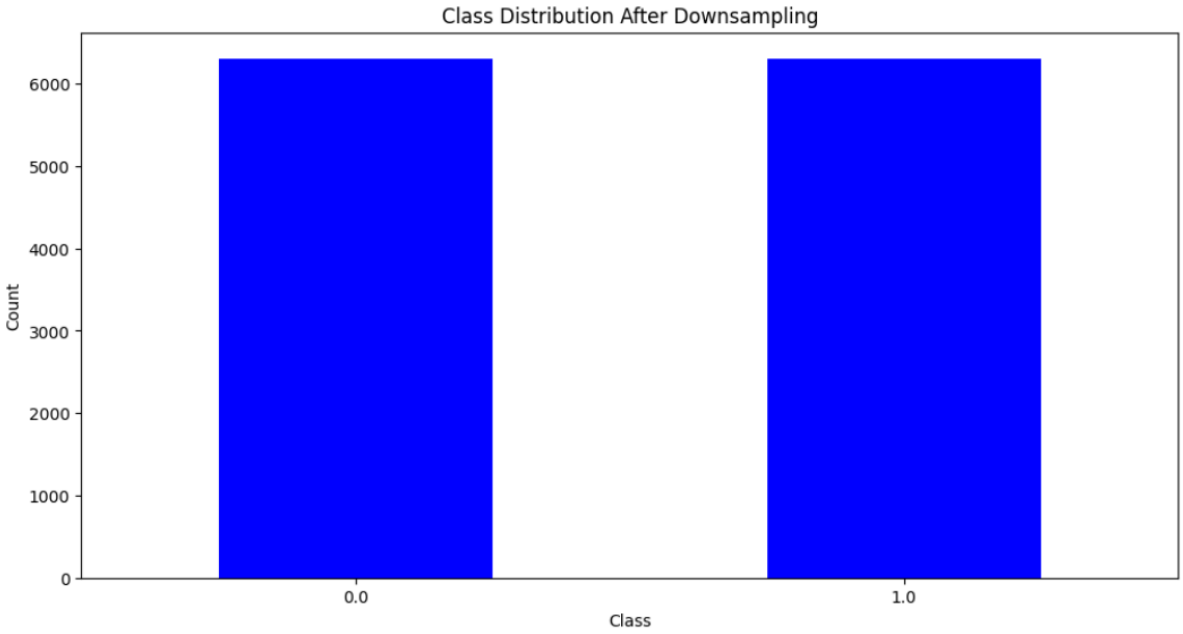
the majority class during model training. Oversampling involved randomly duplicating instances of the minority class to balance class distribution, while downsampling randomly reduced instances in the majority class. However, both techniques had drawbacks; oversampling may lead to overfitting on the oversampled class, while downsampling may result in poor model performance due to reduced instances of the majority class.

The 'RandomOverSampler' and 'RandomUnderSampler' from the 'imblearn' library were imported and utilised for oversampling and undersampling, respectively. The resamplers were fitted using the 'fit_resampled' function on the reshaped 2-D array input training set (X_train) and output training set (y_train). This yielded resampled 2-D array X_train and resampled 1-D y_train. Subsequently, the resampled X_train is flattened to a 1-D array and converted to a Pandas series for consistency, along with the resampled y_train. The oversampled, and undersampled datasets were then used for model hyperparameter tuning and training.




```
Total number of text that label as non hate = 31599
Total number of text that label as hate = 31599
Total unique value in column 'hate_speech':
hate_speech
1.0    31599
0.0    31599
Name: count, dtype: int64
```

Figure 4.2.2.12. 1 Class distribution on training dataset after oversampling.




```
Total number of text that label as non hate = 6309
Total number of text that label as hate = 6309
Total unique value in column 'hate_speech':
hate_speech
0.0    6309
1.0    6309
Name: count, dtype: int64
```

Figure 4.2.2.12. 2 Class distribution on training dataset after downsampling.

4.2.3 Initial Model training

The hate speech detection model utilised four deep learning models: the CNN model, BiLSTM model, DistilBERT model and RoBERTa model, specifically the 'facebook/roberta-hate-speech-dynabench-r4-target'. The architecture of these models was detailed in Chapter 3 System Design. They were trained using oversampled and downsampled training datasets and evaluated during training with the evaluation dataset. The configurations were as follows:

- CNN model with Adam optimizer and a learning rate of 0.0005, a batch size of 32, trained for maximum 30 epochs. Early stopping was implemented with a callback, monitoring the validation F1 score. The training process ceased if the F1 score did not increase for three consecutive epochs, starting from epoch number 5. 'restore_best_weights' was set to True.
  - Embedding layer:
    - Input dimension (size of the vocabulary) = 19500.

        - Output dimension (dimension of the dense embedding) = 300 for GloVe, Word2Vec, GloVe + Word2Vec (mean), and GloVe + Word2Vec (rms); 600 for word2vec + GloVe (stack).
        - Weights = The embedding matrix contained all embedding vectors for unique tokenised words in the dataset, obtained from the word embedding techniques.
        - Trainable = True.
    - Convolutional 1D layer:
        - Filters = 32
        - Kernel = 10
        - Padding = 'same'
        - Activation = 'relu'
        - Kernel regularizer = L2(3)
    - Maxpooling 1D layer:
        - Pool size = 3
    - Flatten layer: Flattened the input from maxpooling 1D layer to form a column vector.
    - Dropout layer:
        - Dropout rate = 0.3
    - Dense layer:
        - Units = 32
    - Dense layer (output):
        - Units = 1
        - Activation = 'sigmoid'

- BiLSTM with Adam optimizer and a learning rate of 0.0005, a batch size of 32, trained for maximum 30 epochs. Early stopping was implemented with a callback, monitoring the validation F1 score. The training process ceased if the F1 score did not increase for three consecutive epochs, starting from epoch number 5. 'restore_best_weights' was set to True.
    - Embedding layer:
        - Input dimension (size of the vocabulary) = 19500.

        - Output dimension (dimension of the dense embedding) = 300 for GloVe, Word2Vec, GloVe + Word2Vec (mean), and GloVe + Word2Vec (rms); 600 for word2vec + GloVe (stack).
        - Weights = The embedding matrix contained all embedding vectors for unique tokenised words in the dataset, obtained from the word embedding techniques.
        - Trainable = True.
    - Bidirectional LSTM layer:
        - Units = 32.
        - Activation = 'relu'
        - Kernel regularizer = L2(1)
        - Recurrent regularizer = L2(1)
    - Flatten layer: Flattened the input from Bidirectional LSTM layer to form a column vector.
    - Dropout layer:
        - Dropout rate = 0.3
    - Dense layer:
        - Units = 32
    - Dense layer (output):
        - Units = 1
        - Activation = 'sigmoid'

- Pretrained DistilBERT model named 'distilbert/distilbert-base-uncased' and RoBERTa model named 'facebook/roberta-hate-speech-dynabench-r4-target'. To train with these models, tensor dataset needed to be created, which included the input with numerical values, the attention mask of the input, and their actual label for each instance in the training, validation, as well as testing dataset. Early stopping was implemented with a callback, monitoring the validation F1 score. The training process ceased if the F1 score did not increase for three consecutive epochs. DistilBERT models will be fine-tuned for all layers, while for the RoBERTa models, only the layers containing 'dense' or 'out_proj' in the parameter name will be fine-tuned. Other layers will freeze.
    - Training Arguments
        - Overwrite output dir = True

- Per device train batch size = 16
- Num train epochs = 20
- Evaluation strategy = ‘epoch’
- Save strategy = ‘epoch’
- Metric for best model = ‘eval_f1’
- Load best model at end = True
- Save total limit = 10
- Learning rate = 1e-5

4.2.4 Model Hyperparameters Tuning

Hyperparameters tuning, also known as hypertuning, is a crucial process in machine learning that aims to identify the optimal set of external configurations for a model training begins. Examples of hyperparameters include the number of features generated by neural networks layers, learning rate, and the number of neurons in hidden layers. Finding the best values for these hyperparameters is essential for achieving optimal performance in deep learning models.

- CNN and BiLSTM models underwent hyperparameter tuning using GridSearch provided by keras-tuner library, with the objective of maximising the validation F1 score. The configurations of the hyperparameters to be tuned are shown as follows:
  - Filters (CNN), Units (BiLSTM) = [16, 32, 64, 128]
  - Dense units = [16, 32, 64, 128]
  - Learning rate = [0.0001, 0.0005, 0.001, 0.005]

- DistilBERT and RoBERTa models underwent hyperparameter tuning using ‘hyperparameter_search’ function provided in the Trainer class, with the optuna backend, with the objective of maximising the validation F1 score. Six trials were conducted for the model trained with the downsampled dataset, and four trials were conducted for model trained with the oversampled dataset. The configurations of the hyperparameters to be tuned are shown as follows:
  - Learning rate = range from [0.00001, 0.0001]
  - Per device train batch size = [16, 32, 64] for DistilBERT, [8, 16, 32] for RoBERTa

4.2.5 Model Evaluation

The performance of the hypertuned models was then evaluated using the test dataset, which comprised 4,738 instances of hate speech tweets and non-hate speech tweets. The specific evaluation metrics used to evaluate the model's performance were precision, recall, F1 score, and confusion matrix. Accuracy was not considered as an evaluation metric for the models in this project because the dataset used in this project was imbalanced and had been oversampled/ downsampled. Using accuracy as an evaluation metrics would not have effectively evaluated the model. Details about the evaluation metrics were mentioned in Chapter 6.

Chapter 5
System Implementation

5.1 Hardware Setup

PC Model:	HP Gaming Pavillion – 15-dk0243tx
Number of Processors	12 (6 Cores)
RAM:	32 GB
Storage:	512 + 960 GB
Cuda Cores:	1536
GPU Memory	6 GB

Table 5.1. 1 Hardware Specifications.

5.2 Software Setup

In this project, two software programs were required to be installed: Python version 3.11.7 and Jupyter Notebook version 7.0.6. Python is the programming language used for preprocessing the dataset and developing the hate speech detection model. Jupyter Notebook is an open-source web application that allowed users to create Python code for developing the hate speech detection model. It supported a wide range of programming languages. The python libraries required to develop the hate speech detection model were listed in Table 5.2.1.

Python Libraries	
datasets	safetensors
gensim	scikeras
keras	scikit-learn
keras-tuner	seaborn
matplotlib	tensorflow
nltk	tokenisers
numpy	torch
optuna	tqdm
pandas	transformers

Table 5.2. 1 Python libraries required.

5.3 Implementation Issues and Challenges

There were several challenges faced during implementation of the hate speech detection models. The first challenge was the class imbalanced issue in the dataset used to train the developed models. Despite attempts to resolve the issue through oversampling and downsampling techniques, these approaches did not provide suitable solutions. This was because the majority class had instances five times greater than the minority class, increasing the risk that the models would not perform well on the downsampled class. Additionally, there was a risk of the model becoming overfitted to the oversampled minority class due to its instances being oversampled five times.

Furthermore, the keras-tuner library utilised in this project to hypertune the CNN and BiLSTM models, was not compatible with the callbacks set to begin monitoring the model's performance from the fifth epoch and stop training after 3 epochs without any improvement. Although the training process in the hypertuning trials executed the set callbacks, the best performance recorded in the keras file sometimes fell within the first five training epochs. This inconsistency caused the best hyperparameters stored in the keras tuner object to be unreliable, necessitating the review of the results in each trial to select the hyperparameters. Similarly, the hyperparameter search function provided by the trainer class also faced the same issue with callbacks. It only selected the best trial by comparing the performance of the last epoch for each trial, thus requiring a check of the performance before the callback stopped the model training for each trial.

5.4 Concluding Remark

To develop a hate speech detection model using deep learning techniques, it was necessary to install Python version 3.11.7 and Jupyter Notebook version 7.0.6, along with several Python libraries. Additionally, several challenges were encountered during the implementation of hate speech detection models, such as the class imbalanced issue in the training dataset and the incompatibility between the tuner library and the callbacks set. The downsampling and oversampling techniques used to resolve the class imbalance issue in the training dataset could raise the drawbacks of these techniques. Due to the incompatibility of tuner library and the callbacks set, it requires going through each trial manually to select the hyperparameters.

Chapter 6
System Evaluation and Discussion

6.1 System Testing and Performance Metrics

There were several evaluation metrics used to assess the performance of the hate speech detection model, including precision, recall, difference between the precision and recall, and the F1-score. Precision evaluated the model's ability to accurately predict tweets as hate speech. It was calculated by determining the ratio of actual hate speech predicted as hate speech (True Positives) to the sum of actual hate speech predicted as hate speech (True Positives) and actual non-hate speech predicted as hate speech (False Positives). In other words, precision quantified the accuracy of the model's positive predictions, specifically those related to hate speech.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (6)$$

where

- $True\ Positives$ represent instances of actual hate speech correctly predicted as hate speech
- $False\ Positives$ represent instances of actual non-hate speech incorrectly predicted as hate speech

Recall is a metric employed to assess the model's effectiveness in predicting actual hate speech tweets as hate speech. The calculation involved determining the ratio of instances where actual hate speech was correctly predicted as hate speech (True Positives) to the total of actual hate speech predicted as hate speech (True Positives) and instances where hate speech was predicted as non-hate speech (False Negatives). In simpler terms, recall measured the model's capability to identify and predict instances of genuine hate speech.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (7)$$

where

- $True\ Positives$ represent instances of actual hate speech correctly predicted as hate speech

- $False\ Negatives$ represent instances of actual hate speech incorrectly predicted as non-hate speech

The difference between precision and recall was evaluated to have a better monitor on the model's performance on different techniques to solve the class imbalance issue. The F1 score combined precision and recall, providing a balanced measure of the model's effectiveness. This was particularly valuable when both precision and recall were deemed important, and the dataset used to train the model was imbalanced, as was the case in this project. The F1 score was considered more reliable in this project because it calculated performance using both precision and recall, ensuring a balanced evaluation of the model's effectiveness.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \qquad (8)$$

6.2 Testing Setup and Result

The initial models, configured with hyperparameters as mentioned in Chapter 4.2.3 for CNN and BiLSTM with different word embedding techniques as outlined in Chapter 4.2.2.10, along with the DistilBERT and RoBERTa transformer models, were hypertuned using the hyperparameter configurations detailed in Chapter 4.2.4. Subsequently, they were trained with the hyperparameters obtained from the hyperparameter tuning process. The results illustrated will represent the evaluation outcomes of the hypertuned models. They will be presented separately in different tables. Specifically, the evaluation results for the hypertuned CNN and BiLSTM models with different word embedding techniques will be showcased in the same table, while the evaluation results for the hypertuned DistilBERT and RoBERTa transformer models will be presented in another table, as DistilBERT and RoBERTa utilise their own word embedding vectors from the models themselves.

Deep Learning Model	**Embedding Technique**	**Dataset**	**Performance Evaluation**			
			Precision (%)	**Recall (%)**	**Precision vs Recall (%)**	**F1 score (%)**
CNN	Word2Vec	Twitter -	94.9	82.3	12.6	88.2
		Twitter +	91.9	86.6	5.3	89.2
	GloVe	Twitter -	93.1	86.0	7.1	89.4

		Twitter +	91.7	87.7	3.9	89.7
	GloVe + Word2Vec (stack)	Twitter -	91.7	89.3	2.4	90.4
		Twitter +	91.2	89.0	2.2	90.1
	GloVe + Word2Vec (mean)	Twitter -	96.0	82.5	13.5	88.7
		Twitter +	90.6	88.3	2.2	89.4
	GloVe + Word2Vec (rms)	Twitter -	93.1	87.5	5.6	90.2
		Twitter +	93.5	85.8	7.7	89.5
BiLSTM	Word2Vec	Twitter -	92.4	86.0	6.4	89.1
		Twitter +	92.9	86.8	6.1	89.7
	GloVe	Twitter -	94.5	82.9	11.6	88.3
		Twitter +	90.8	89.5	1.3	90.1
	GloVe + Word2Vec (stack)	Twitter -	91.3	86.9	4.3	89.0
		Twitter +	92.6	84.1	8.5	88.1
	GloVe + Word2Vec (mean)	Twitter -	94.9	80.7	14.2	87.2
		Twitter +	90.7	87.7	3.0	89.2
	GloVe + Word2Vec (rms)	Twitter -	92.7	84.2	8.6	88.2
		Twitter +	90.9	87.8	3.1	89.3

Table 6.2. 1 Testing Precision, Recall, and F1 score results for different word embedding techniques applied to hypertuned CNN and BiLSTM models. Twitter + denote models trained with oversampled Twitter datasets, while Twitter – denote models trained with downsample.

Deep Learning Model	Dataset	Performance Evaluation			
		Precision (%)	Recall (%)	Precision vs Recall (%)	F1 score (%)
DistilBERT	Twitter -	88.5	82.0	6.5	83.8
	Twitter +	85.2	85.1	0.1	85.1
RoBERTa	Twitter -	87.7	82.2	5.5	83.9

	Twitter +	86.0	85.1	0.9	85.5

Table 6.2. 2 Testing Precision, Recall, and F1 score results for the hypertuned DistilBERT and RoBERTa models. Twitter + denote models trained with oversampled Twitter datasets, while Twitter – denote models trained with downsample.

6.3 Project Challenges

The only and main challenge faced in this project would be the limited computational resources of the hardware, including RAM, GPU memory, storage, and the number of processors available for training the deep learning model and hypertuning it. These resource constraints were significant challenges faced during the project. Due to these limitations, training the model and hypertuning its hyperparameters required more time. Consequently, a significant portion of the project's time was dedicated to the hyperparameter tuning process.

Moreover, due to the limited computational resources, we were unable to hypertune many of the model's hyperparameters. Instead, we focused solely on several key hyperparameters that could significantly affect the model's performance. These included the number of filters generated by the 1D convolutional layer, the number of neurons in a fully connected layer, and the batch size used for fine-tuning a transformer model. Consequently, while the hypertuned model's performance may have improved, it may not have been fully optimized.

6.4 Objectives Evaluation

From the results presented in Table 6.2.1, it was evident that there was no significant difference in performance between the CNN and BiLSTM models. This lack of distinction could be attributed to the relatively shallow architecture of both models, each comprising only a single layer of convolutional 1D or a single BiLSTM layer with two hidden layers, including the output dense layer. Increasing the depth of these models by adding more layers could have potentially enhanced their performance and allowed for better differentiation between CNN and BiLSTM. However, this approach came with its challenges, as augmenting the model depth may have significantly increased the training time for each model and prolonged the hyperparameter tuning process, owing to computational resource limitations. This could have potentially led to project delays or failure to meet deadlines. As a result, a decision was made

to stick with simpler CNN and BiLSTM architectures to ensure timely completion of the project.

For CNN models trained on downsampled dataset, they had a precision ranging from 91 to 96%, a recall ranging from 82 to 90%, and an F1 score ranging from 88 to 91%. In contrast, CNN models trained on oversampled dataset demonstrated precision, recall, and F1 scores ranging from 90 to 94%, 85 to 89%, and 88 to 91% respectively. Moving on to the BiLSTM models trained on the downsampled dataset, their precision ranged from 91 to 95%, recall ranged from 80 to 87%, and F1 score ranged from 88 to 89%. Similarly, BiLSTM models trained on the oversampled dataset exhibited precision, recall, and F1 scores ranging from 90 to 93%, 84 to 90%, and 88 to 90%, respectively.

This highlighted the drawbacks of the downsampling and oversampling techniques employed in this project to address the class imbalance issue in the training dataset. Downsampling the training dataset resulted in lower recall performance for all CNN and BiLSTM models compared to models trained on the oversampled dataset, except for models that utilised a combination of two word embedding techniques. Downsampling reduced instances in the majority class, which was the hate class (labelled as '1'), potentially causing the models to lose vital information needed to predict hate class instances accurately. Consequently, models trained with the downsampled dataset may have exhibited lower recall compared to those trained with the oversampled dataset, which maintained the original number of hate class instances.

Conversely, oversampling the training dataset could have led to slight overfitting to the oversampled class, which was the non-hate class (labelled as '0'), potentially resulting in inaccurate predictions of actual non-hate tweets in the testing dataset. This could have resulted in lower precision for models trained on the oversampled dataset compared to the downsampling technique. Despite these drawbacks, training models with the oversampled dataset tended to reduce the difference between precision and recall. The precision-recall difference for models trained on the oversampled dataset ranged from only 1 to 9%, which was relatively lower compared to models trained on the downsampled dataset, where the precision-recall difference ranged from 2 to 15%.

In most cases, the difference between the precision and recall of these models decreased when the models were trained on the oversampled dataset compared to the downsampled dataset, except for the CNN model that utilised the word embedding technique of GloVe + Word2Vec (rms) and the BiLSTM model that utilised the word embedding technique of GloVe

+ Word2Vec (stack). These two models exhibited abnormal behavior when trained on the oversampled dataset, showing a higher difference between precision and recall compared to models trained on the downsampled dataset.

This anomaly could be attributed to the issue of insufficient training for the models, which led to the early termination of training as per the set callback. Consequently, the models might have had limited learning opportunities to optimise, particularly in the majority class. Despite oversampling the minority class five times, the models might have achieved better performance in this class, even though they could not undergo sufficient training. Moreover, the risk of overfitting to the oversampled class was lower. As a result, the precision of the models trained on the oversampled dataset would increase, while the recall would be lower compared to models trained on the downsampled dataset.

Combining two different word embedding techniques was supposed to have the benefit of reducing the required training time and improving the model's performance, as it utilised semantic information from different word embedding techniques. This was expected to generate word embedding vectors that captured more accurate semantic information for tokens. However, in this project, the word embedding techniques used combined different word embedding techniques, such as taking the mean or root mean square of both word embedding techniques, without increasing the word embedding vector size. This did not necessarily improve the model's performance; in fact, it even lowered the model's performance and prolonged the training time compared to models trained using a single word embedding technique with the same size as the word embedding vector that combined different word embedding techniques. [10] made a statement that concatenating different word embedding techniques would be able to improve the performance of the developed model. More accurately, the model's performance would improve when the word embedding vector size was larger, instead of concatenating different word embedding techniques, which provided better model performance.

The lower performance of the model by combining two different word embedding techniques could have been due to the CNN and BiLSTM models utilised in this project being shallow, with only a single 1D convolutional layer and BiLSTM layer. This caused the CNN and BiLSTM models to be unable to extract a higher level of features from the word embedding vectors. Alternatively, it could have been caused by the domain focus for Word2Vec and GloVe being different, as Word2Vec was trained supervisely for predicting tokens in a

sentence, while GloVe was trained unsupervisely with the co-occurrence of words to be present in the same sentence.

According to the results presented in Table 6.2.2, RoBERTa demonstrated better performance in F1 score compared to DistilBERT. However, there was not a significant difference in performance between these two models. Additionally, the performance of these transformer models was lower compared to the CNN and BiLSTM models' performance shown in Table 6.2.1. There was roughly a difference of 4 to 5% for models trained on the downsampled dataset, and around 5% for models trained on the oversampled dataset. This discrepancy could be attributed to the fact that the transformer models used in this project were pretrained models and due to the limited number of hate and non-hate instances in the training dataset. As a result, the transformer models did not have enough training data to fine-tune the model parameters, leading to lower performance compared to the CNN and BiLSTM models trained from scratch.

RoBERTa exhibited better performance in F1 score compared to DistilBERT, but the difference was not significant. This could be due to the fact that the RoBERTa model utilised in this project was fine-tuned for several layers, while other layers were frozen, whereas the DistilBERT model fine-tuned parameters in all layers. The decision to fine-tune only several layers in the RoBERTa transformer models was due to the extensive computational resources required to fine-tune all layers, which were not available with the hardware used in this project. However, even with only fine-tuning several layers, RoBERTa still outperformed DistilBERT in terms of Precision, Recall, and F1 score. This demonstrates that utilising a pretrained model on hate speech text and fine-tuning with another dataset can reduce computational resources and improve model performance.

Furthermore, the application of downsampling and oversampling techniques on the training dataset to address class imbalance issues also affected the transformer models. However, the effect of these techniques was not as significant, as the difference between precision and recall in all the models was lower compared to the CNN and BiLSTM model performance shown in Figure 6.2.1. The difference between precision and recall for the DistilBERT model on the downsampled dataset and oversampled dataset was 6.5 and 0.1 respectively, while the difference between precision and recall for the RoBERTa model on the downsampled dataset and oversampled dataset was 5.5 and 0.9 respectively. The difference was smaller than some of the CNN and BiLSTM models. In terms of the difference between precision and recall, DistilBERT and BiLSTM models could perform better compared to CNN

and BiLSTM models. However, in terms of Precision, Recall, and F1 score, CNN and BiLSTM models outperformed DistilBERT and RoBERTa models.

Although CNN and BiLSTM models had better performance in Precision, Recall, and F1 score, DistilBERT and RoBERTa models could perform better on a testing dataset that contained hate speech text different from the testing dataset currently used to evaluate the model performance. This was because the CNN and BiLSTM models were developed from scratch with a specific dataset, while the DistilBERT and RoBERTa transformer models were pretrained with a large corpus of text to learn the features of tokens and improve model generalisation. Therefore, DistilBERT and RoBERTa transformers developed in this project could outperform the CNN and BiLSTM models on a different dataset that was completely different from the dataset used to train the model.

A notable point to consider was that using BiLSTM required more training time compared to the CNN architecture. This was because BiLSTM processed the input sequentially in both forward and backward directions, effectively processing each tweet twice, resulting in longer training times compared to the CNN architecture. Additionally, DistilBERT required longer training time compared to BiLSTM, and RoBERTa required even longer training time compared to DistilBERT, despite RoBERTa only fine-tuning several layers. The longer training times indicated that more computational resources were required to train the models. Therefore, the computational resources consumption among these models followed the order: RoBERTa > DistilBERT > BiLSTM > CNN.

6.5 Concluding Remark

Conducting a deep learning project is resources-intensive, requiring large amounts of computational resources such as RAM, GPU memory, storage, and others. It is recommended to utilise other open-source web applications that provide computational resources, such as Google Colab or Kaggle to conduct the project. Otherwise, extensive time spent on training or hypertuning the model could have led to project delays or incomplete execution. The computational resources consumption among the models developed in this project followed the order: RoBERTa > DistilBERT > BiLSTM > CNN.

BiLSTM and CNN models outperformed RoBERTa and DistilBERT in terms of Precision, Recall, and F1 score, while RoBERTa and DistilBERT performed better on the difference between precision and recall. CNN and BiLSTM required less computational resources compared to RoBERTa and DistilBERT, as the training time was shorter. If the

computational resources of a project were limited and the dataset was small, CNN and BiLSTM models would be more preferable. If the training dataset was large and computational resources were not an issue for a project, trying out RoBERTa and DistilBERT could have outperformed the CNN and BiLSTM models built from scratch.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

Different word embedding techniques were applied to the CNN and BiLSTM models, including the use of single word embedding techniques and the combination of different word embedding techniques by stacking, averaging, or taking the root mean square of them. Additionally, two pretrained transformer models were utilised to develop the hate speech detection models, including DistilBERT and RoBERTa.

BiLSTM and CNN models outperformed RoBERTa and DistilBERT in terms of Precision, Recall, and F1 score, with values ranging from 90 to 96%, 80 to 90%, and 87 to 91%, respectively. However, RoBERTa and DistilBERT performed better on the difference between precision and recall, ranging from 0.1 to 6.5%. The difference between the CNN and BiLSTM models was not significant, while CNN could perform better in some cases of combining two word embedding techniques, possibly due to the shallow model architecture. The lower performance of DistilBERT and RoBERTa compared to CNN and BiLSTM could be attributed to the limited hate and non-hate instances in the training dataset.

In this project, it was also observed that utilising oversampling techniques to resolve the class imbalance issue on the training dataset could reduce the model's performance on precision, as the model could be overfit to the minority class which had been oversampled in the training dataset. Additionally, the use of downsampling techniques to resolve class imbalance issues could have caused the model not to learn well on the majority class, which had been downsampled in the training dataset. CNN required less computational power compared to BiLSTM, while BiLSTM required less computational power compared to DistilBERT. RoBERTa remained the model that required the most computational power among the models developed in this project.

7.2 Recommendation

In future work, a multilingual hate speech dataset containing texts or comments related to hate speech, not only from X (Twitter) but also from different social media platforms, will be created. Since datasets collected from social media could raise the risk of class imbalance

issues, we will not only gather texts or comments from various social media platforms but also conduct crowdsourcing to obtain hate speech samples from the public. This will increase the instances of hate speech and help resolve the issue of class imbalance. All texts or comments obtained from the public or social media will be reviewed by different experts in the field of hate speech and carefully annotated to reduce the risk of misclassification misannotation.

Next, an investigation will be conducted to evaluate the performance of models trained with datasets with and without removing stopwords. Although stopwords are considered to contain less meaning in NLP tasks, they could be important in allowing a deep learning model to understand the context of data. Removing these stopwords could raise concerns that the model may not be able to learn the context features of the data, affecting the model performance. Therefore, there is a need to study the performance of models with stopwords, particularly for transformer models.

Lastly, in addition to focusing on the development of hate speech detection models using different deep learning techniques, there will be an effort to develop a software application that can be used to detect the hate speech text, making the models practically usable. The software application will assist users in identifying hate speech text and can serve as one of the methods for collecting hate speech instances when users classify the text. Moreover, it can be integrated into social media platform applications to help users identify hate speech text on these platforms. This task could be challenging as the data flow in a social media platform application is intensive and requires a large amount of computational power to process the text. However, this step aims to apply the developed models to address hate speech cases on social media platforms and practically enhance the online environment.

REFERENCES

[1] T. Lavelle, “The Musk Bump: Quantifying the rise in hate speech under Elon Musk,” *Center for Countering Digital Hate | CCDH*, Dec. 06, 2022. https://counterhate.com/blog/the-musk-bump-quantifying-the-rise-in-hate-speech-under-elon-musk/

[2] S. A. Mohd Fadhli, J. Liew Suet Yan, A. S. Ab Halim, A. Ab Razak, and A. Ab Rahman, “Finding the Link between Cyberbullying and Suicidal Behaviour among Adolescents in Peninsular Malaysia,” *Healthcare*, vol. 10, no. 5, p. 856, May 2022, doi: https://doi.org/10.3390/healthcare10050856.

[3] “Deep Learning vs. machine learning: A beginner’s guide,” Coursera, https://www.coursera.org/articles/ai-vs-deep-learning-vs-machine-learning-beginners-guide?utm_source=gg&utm_medium=sem&utm_campaign=B2C_APAC__branded_FTCOF_courseraplus_arte_PMax_set2&utm_content=Degree&campaignid=20520149492&adgroupid=&device=c&keyword=&matchtype=&network=x&devicemodel=&adpostion=&creativeid=&hide_mobile_promo&gclid=CjwKCAiAgeeqBhBAEiwAoDDhn1Dy1U2aZHwbgsKDOyuDwCDUI_nNDQ4JIdyw0GOhoWwHWuu7WQa_AhoCJLQQAvD_BwE (accessed Nov. 19, 2023).

[4] X. Zhang et al., "Cyberbullying Detection with a Pronunciation Based Convolutional Neural Network," 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), Anaheim, CA, USA, 2016, pp. 740-745, doi: 10.1109/ICMLA.2016.0132.

[5] M. A, Al-Ajlan and M. Ykhlef, “Deep Learning Algorithm for Cyberbullying Detection,” International Journal of Advanced Computer Science and Applications, vol. 9, no. 9, 2018, doi: https://doi.org/10.14569/ojacsa.2018.090927.

[6] M. Susanty, Sahrul, A. F. Rahman, M. D. Normansyah and A. Irawan, "Offensive Language Detection using Artificial Neural Network," 2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT), Yogyakarta, Indonesia, 2019, pp. 350-353, doi: 10.1109/ICAIIT.2019.8834452.

[7] S. Butt, “Sexism Identification using BERT and Data Augmentation - EXIST2021,” 2021. https://www.semanticscholar.org/paper/Sexism-Identification-using-BERT-and-Data-EXIST2021-Butt-Ashraf/507e739e2d2931ff23b36c8a42d68fcfb836a56d#citing-papers

[8] S. Agrawal and A. Awekar, “Deep Learning for Detecting Cyberbullying Across Multiple Social Media Platforms,” *Lecture Notes in Computer Science*, pp. 141–153, 2018, doi: https://doi.org/10.1007/978-3-319-76941-7_11.

[9] J. Wang, K. Fu and C. -T. Lu, "SOSNet: A Graph Convolutional Network Approach to Fine-Grained Cyberbullying Detection," 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, pp. 1699-1708, doi: 10.1109/BigData50022.2020.9378065.

[10] M. Raj, S. Singh, K. Solanki, and R. Selvanambi, “An Application to Detect Cyberbullying Using Machine Learning and Deep Learning Techniques,” *SN Computer Science*, vol. 3, no. 5, Jul. 2022, doi: https://doi.org/10.1007/s42979-022-01308-5.

[11] J. Fillies, M. P. Hoffmann and A. Paschke, "Multilingual Hate Speech Detection: Comparison of Transfer Learning Methods to Classify German, Italian, and Spanish Posts," 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, pp. 5503-5511, doi: 10.1109/BigData59044.2023.10386244.

[12] V. -C. Dinh, T. -D. Vo, M. -P. T. Nguyen and T. -H. Do, "A Scalable Hate Speech Detection System for Vietnamese Social Media using Real-time Big Data Processing and Distributed Deep Learning," 2023 International Conference on Advanced Technologies for Communications (ATC), Da Nang, Vietnam, 2023, pp. 95-100, doi: 10.1109/ATC58710.2023.10318848.

[13] Q. Sifak and E. B. Setiawan, "Hate Speech Detection using CNN and BiGRU with Attention Mechanism on Twitter," 2023 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), Malang, Indonesia, 2023, pp. 170-175, doi: 10.1109/COMNETSAT59769.2023.10420628.

[14] H. Adel et al., “Improving Crisis Events Detection Using DistilBERT with Hunger Games Search Algorithm,” *Mathematics*, vol. 10, no. 3, p. 447, Jan. 2022, doi: https://doi.org/10.3390/math10030447.

[15] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” *arXiv.org*, Feb. 29, 2020. https://arxiv.org/abs/1910.01108v4

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv.org*, May 24, 2019. https://arxiv.org/abs/1810.04805#

[17] B. Vidgen, T. Thrush, Z. Waseem, and Douwe Kiela, “Learning from the Worst: Dynamically Generated Datasets to Improve Online Hate Detection,” *arXiv (Cornell University)*, Dec. 2020, doi: https://doi.org/10.48550/arxiv.2012.15761.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” *arXiv.org*, Sep. 06, 2013. http://arxiv.org/abs/1301.3781

[19] J. Pennington, R. Socher, and C. Manning, “Glove: Global Vectors for Word Representation,” *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014, doi: https://doi.org/10.3115/v1/d14-1162.

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 1**
Student Name & ID: Thong Wei Xin 20ACB02627	
Supervisor: Ts Dr. Vikneswary a/p Jayapal	
Project Title: Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques	

1. WORK DONE

Transfered some of the content in FYP1 report to FYP2 report.

2. WORK TO BE DONE

Hypertuning the CNN models and completing some parts of the FYP2 report.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

-

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 3**
Student Name & ID: Thong Wei Xin 20ACB02627	
Supervisor: Ts Dr. Vikneswary a/p Jayapal	
Project Title: Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques	

1. WORK DONE

The hyperparameter tuning for CNNs models have been completed. Parts of the FYP2 report is completed.

2. WORK TO BE DONE

Experiment with the root mean square of GloVe and Word2Vec word embedding vectors in Convolutional Neural Networks (CNNs) for subsequent analysis. Revise the literature review to update the most recent advancements in hate speech detection models. Optimize the Bidirectional Long Short-Term Memory (BiLSTM) model by tuning the number of output units in the BiLSTM layer, neurons in the hidden layer, and the learning rate for BiLSTM models.

3. PROBLEMS ENCOUNTERED

The computational limitations of the PC extend the hyperparameter tuning process, allowing only a few parameters such as the number of filters, neurons in the hidden layer, and the learning rate of the deep learning model to be tuned.

4. SELF EVALUATION OF THE PROGRESS

The hyperparameter tuning process is progressing smoothly, albeit taking 1-2 weeks to complete for each deep learning architecture. We anticipate completing the hyperparameter tuning process for BiLSTM by the week after next.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 7**
Student Name & ID: Thong Wei Xin 20ACB02627	
Supervisor: Ts Dr. Vikneswary a/p Jayapal	
Project Title: Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques	

1. WORK DONE

The hyperparameter tuning for CNNs and BiLSMT models has been completed. Several recently published research papers regarding the techniques used in hate speech detection or cyberbullying have been studied. CNN and BiLSTM models utilised root mean square word embedding techniques have been hypertuned.

2. WORK TO BE DONE

Hypertuning the learning rate for several transformer models such as RoBERTa for hate speech, and DistilBERT. Train and test the models using the hypertuned hyperparameters to obtain optimized results for each model. From the results obtained from different models, analyze the findings to identify potential issues and provide justifications.

3. PROBLEMS ENCOUNTERED

The evaluation F1 score for models that utilised combined word embedding techniques did not improve significantly compared to the use of a single word embedding technique.

4. SELF EVALUATION OF THE PROGRESS

The progress of the project is smooth. Hypertuning and training of models with the hypertuned parameters are expected to be completed in 3 more weeks, and this will not delay my report submission.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 10**
Student Name & ID: Thong Wei Xin 20ACB02627	
Supervisor: Ts Dr. Vikneswary a/p Jayapal	
Project Title: Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques	

1. WORK DONE

DistilBERT transformer models hypertuning and evaluate the DistilBERT models.

2. WORK TO BE DONE

Evaluate the hypertuned CNN and BiLSTM models and start finalising the FYP2 report.

3. PROBLEMS ENCOUNTERED

Due to the limitation of computational resources, the hypertuning for transformer models requires more time and only a few trials were conducted for each model.

4. SELF EVALUATION OF THE PROGRESS

Report almost completed, remain the hypertuning and evaluation of transformer models.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 12**
Student Name & ID: Thong Wei Xin 20ACB02627	
Supervisor: Ts Dr. Vikneswary a/p Jayapal	
Project Title: Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques	

1. WORK DONE

RoBERTa is hypertuned and evaluated. The hypertuned CNN and BiLSTM models are evaluated and analysis.

2. WORK TO BE DONE

Evaluate the DistilBERT and RoBERTa transformer models.

3. PROBLEMS ENCOUNTERED

The callbacks are not compatible with the keras-tuner library, callback is set to monitor performance of models starting from fifth epoch and stopped training if performance do not have improvement for three consecutive epochs. But the result obtained by the tuner contains performance from the first five epochs.

4. SELF EVALUATION OF THE PROGRESS

Although hypertuning transformer models require more computational power compared to CNN and BiLSTM models and requires longer time, it can complete hypertuning 1 week before the submission deadline.

Supervisor's signature

Student's signature

POSTER

Faculty of Information and Communication Technology

UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Deep Learning for HATE Speech Detection on X (Twitter) with Different Word Embedding Techniques

Problem Statement:

- Racism term increased by 202% to 3,876 daily tweets
- Homophobic term increased by 58% to 3,964 daily tweets
- Misogynist term increased by 33% to 17,937 daily tweets
- Transphobic term increased by 62% to 5,117 daily tweets

Objectives:

- Investigate deep learning techniques
- Develop models with the investigated techniques
- Evaluate the developed models

Motivation:

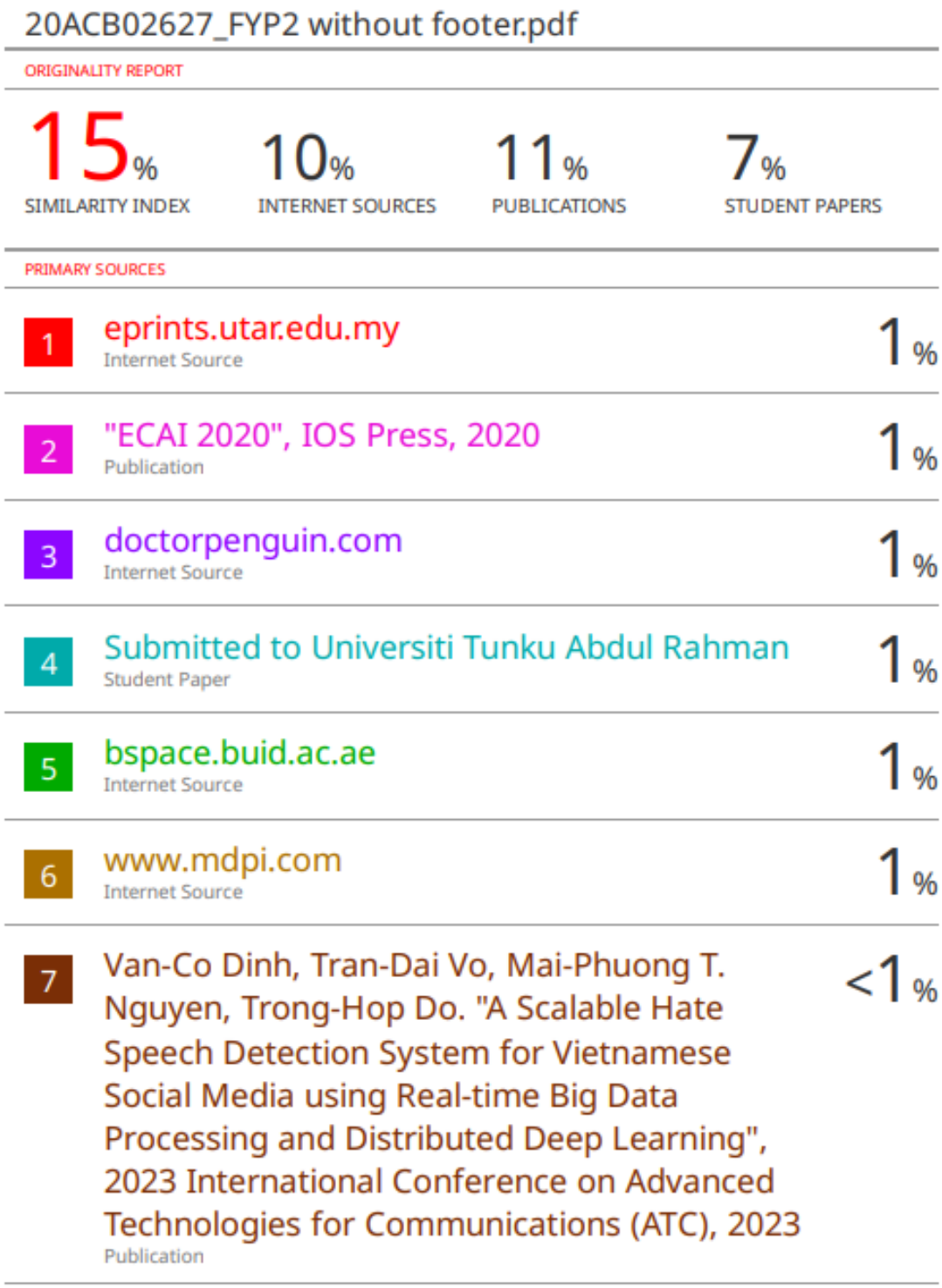
- Enhance online communication environment
- Reduce suicide rate due to cyberbullying

Methodology:

- CNN, BiLSTM, DistilBERT, RoBERTa Deep Learning Techniques
- Utilisng different word embedding technique such as GloVe, Word2Vec, GloVe + Word2Vec(stack), GloVe + Word2Vec(mean), GloVe + Word2Vec(root mean square)

Final Year Project By
Thong Wei Xin

PLAGIARISM CHECK RESULT



20ACB02627_FYP2 without footer.pdf

ORIGINALITY REPORT

15%	10%	11%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprints.utar.edu.my Internet Source	1%
2	"ECAI 2020", IOS Press, 2020 Publication	1%
3	doctorpenguin.com Internet Source	1%
4	Submitted to Universiti Tunku Abdul Rahman Student Paper	1%
5	bspace.buid.ac.ae Internet Source	1%
6	www.mdpi.com Internet Source	1%
7	Van-Co Dinh, Tran-Dai Vo, Mai-Phuong T. Nguyen, Trong-Hop Do. "A Scalable Hate Speech Detection System for Vietnamese Social Media using Real-time Big Data Processing and Distributed Deep Learning", 2023 International Conference on Advanced Technologies for Communications (ATC), 2023 Publication	<1%

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

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Thong Wei Xin
ID Number(s)	20ACB02627
Programme / Course	Computer Science
Title of Final Year Project	Deep Learning for Hate Speech Detection on X (Twitter) with different Word Embedding Techniques

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 15 % **Similarity by source** Internet Sources: **10** % Publications: **11** % Student Papers: **7** %	
Number of individual sources listed of more than 3% similarity: **0**	
Parameters of originality required and limits approved by UTAR are as Follows: **(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** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

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

Name: Dr. Vikneswary Jayapal

Date: 25/04/2024

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB02627
Student Name	Thong Wei Xin
Supervisor Name	Ts Dr. Vikneswary a/p Jayapal

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.

_____

(Signature of Student)
Date: April 24, 2024