DEVELOPING A DEEP LEARNING MODEL TO DETECT SOCIAL MEDIA HATE SPEECH TEXTS

BY

BESTER LOO MAN TING

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)

JUNE 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express gratitude towards my project supervisor, Dr Ramesh Kumar Ayyasamy for his inspiration of the project title, his immense patience and proactivity in guiding and aiding me towards completion and success of this project. At the meanwhile, I would also like to thank Dr Tahayna Bashar for his experience in this problem domain, his recommendations in the tasks involved for this project, your advice had set a marvellous direction for this project's journey.

To my parents, thank you for having faith in me, supporting me all the way and allocating endless time and resources in helping me completing my diligence.

Finally, to Caitlyn, your support is a great motivation towards completing this subject.

ABSTRACT

Hate speech detection on online social media (OSM) platforms remains a significant challenge due to the complexity of linguistic expression and inherent class imbalance in available datasets. This study proposes a hybrid deep learning framework that integrates DistilBERT with CNN and BiLSTM layers to perform multi-class classification, categorizing input text into hate speech, offensive language, or neutral classes. Five experimental configurations were conducted using the Davidson dataset, including baseline training, resampling, class weighting, combined imbalance mitigation, and an ablation study on preprocessing. DistilBERT was selected as the core architecture to balance computational efficiency with representational power. The baseline model achieved strong performance with 88.32% accuracy, a Cohen's Kappa of 0.6805, and a macro-AUC of 0.9260. Class imbalance mitigation techniques demonstrated trade-offs: resampling and class weighting improved the minority class F1-score to above 0.40 but reduced overall accuracy. The ablation study confirmed the critical role of preprocessing, as the exclusion of data cleaning and tokenization degraded minority class F1-score to 0.2886 despite stable accuracy. Overall, results highlight the effectiveness of lightweight BERT-based architectures and produced comparable result in detecting underrepresented class of data for hate speech detection while emphasizing the importance of preprocessing and balanced training strategies for equitable classification performance.

Area of Study: Artificial Intelligence, Text-mining

Keywords: Hate Speech, Social Media, Multi-Label Classification, Deep Learning, Natural Language Processing

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

D_{cleane}	Cleaned D_{tweet}
D_{stem}	D_{tweet} after Stemming
D_{token}	Tokenized D_{tweet}
D_{tweet}	"Tweet" Attribute of DataFrame D
e^{a_i}	Non-Normalized Output from the Previous Dense Layer
F_s	F1-Score
$f(x)_{ReLU}$	Rectifier Linear Unit Activation Function
P_r	Precision
R_c	Recall
AC	Accuracy score
D	DataFrame Object
p	Probability Output from the Softmax Loss Function

LIST OF ABBREVIATIONS

AdaBoost	Adaptive Boosting
BERT	Bidirectional Encoder Representations from Transformer
BiGRU	Bi-directional Gated Recurrent Unit
BiLSTM	Bi-directional Long Short-Term Memory
CNN	Convolutional Neural Networks
DL	Deep learning
DT	Decision Tree
FN	False Negative
FP	False Positive
GRU	Gated Recurrent Unit
HS	Hate speech
KNN	K Nearest Neighbour
LR	Logistic Regression
LSTM	Long Short-Term Memory
mBERT	Multilingual BERT
ML	Machine learning
MLP	Multilayer Perceptron
NB	Naïve Bayes
NLP	Natural Language Processing
OSM	Online social media
RF	Random Forest
RNN	Recurrent Convolutional Network
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Support Vector Machine
TFIDF	Term Frequency-Inverse Document Frequency
TN	True Negative
TP	True Positive
FC	Fully Connected
ReLU	Rectifier Linear Unit
DistilBERT	Distilled Version of BERT

CHAPTER 1 INTRODUCTION

1.1 Problem Statement and Motivation

The advent new features on OSM platforms and along with the consistent growth of users, these platforms has been a source for a huge diversity of data generation. User's online activities on the OSM sites and platforms has generated a great volume of data, in text form, image, music, and videos., structured and unstructured data that are readily collected in a voluminous amount. With these data available and generated, the processing of these data has been challenging. With microblogging websites like Facebook, Instagram, X (previously known as Twitter), and Reddit, users has access to a wide array of features to communicate such as text messaging, image uploading, posts, forums, threads, etc.

With the features on these OSM platforms, people has been allowed to share news, educational media, and sharing the own opinion. With the freedom and conveniences as such, it is no surprise that it can be an open ground for people to start spreading HS. Although free speech can be defended from the standpoint of freedom of speech and people's civil rights, however, the boundary should undeniably be addressed when the speech has incited violence. For instance, the fatal Christchurch shooting[1] incident that originated with the spreading of hate sentiment on Facebook.

Spreading HS has always been a concern for these OSM platforms as the preserving of individual right to free speech and preventing violent HS is posing a dilemma. Some existing approaches has taken effect to moderate the spread of HS, such as the usage of community moderation bot that define the rules of post or comment removal based on predefined rules on Reddit[2], or Instagram's feature that removes or disables accounts of HS abusers[3], comment filters and comment warnings[4] to prevent users from interacting or viewing abusive comments. However, such approaches are still limited to organizations that utilizes it, this is because, X[5] on the other hand, although X has a zero-tolerance policy, it still lacks such feature to prevent the existence of HS and still require manual moderation of HS to remove posts. Hence, with

[1] The New York Times, "Christchurch Shooting Live Updates: 49 Are Dead After 2 Mosques Are Hit," Mar. 2019.
[2] https://support.reddithelp.com/hc/en-us/articles/23511059871252-Content-Moderation-Enforcement-and-Appeals
[3] https://about.instagram.com/blog/announcements/an-update-on-our-work-to-tackle-abuse-on-instagram
[4] https://about.instagram.com/blog/announcements/national-bullying-prevention-month
[5] https://help.x.com/en/rules-and-policies/hateful-conduct-policy

the existing current solution the requirement on extension of research on a generalized approach towards tackling this issue exists.

HS has interpretation that varied across culture, legal system, and integration. To a certain extent, a grand challenge in the existing literature of HS is to establish segregation between HS and hate crime [1]. As such issue poses a moral and legal dilemma, whether to incriminate the spread of HS or to justify HS as a moral right to freedom of expression [2]. Furthermore, HS poses the challenge of being subjectively difficult to be differentiated from language that are purely offensive without the underlying sentiment of hate. On the other hand, even statements that are considered objectively as HS could differ according to its target group, i.e. sexism, racism, religious hate, etc. Such existing complexity has made research difficult in constructing definitive and consistent frameworks.

As privacy is a major privilege for every users in the cyberspace, the anonymity of parties that promotes HS usage further proliferates. As a result, without drastic moderation of online HS, it could cause psychological detriment, as it was evident that HS victim especially adolescent are negatively impacted [3], and further causes undesirable social division, and prolonging the existence of societal marginalization and prejudice. Therefore the lack of an ultimate resolution towards such issues through appropriate action, it ultimately creates an unpleasant cyberspace that encourages HS, which could influence HS victims' mental health and incite violence.

The role of OSM platforms provider is crucial as they play a dual role in the spread of hate speech. While they provide spaces for free expression, the diverse set of existing application features can inadvertently amplify harmful content. Meanwhile, these platforms has to enforce effective guidelines and policies that reflect strategic content moderation and effective communication to mitigate the spread of HS. As it was evident that when extremists or HS offenders are appropriately moderately, it was the exploitation on the platforms by these irresponsible parties can be adequately controlled. For instance, limiting the exposure of users by removing content or shutting down forums of radicalized ideologies or narratives on OSM platforms although compromises the shift of these perpetrators to other platforms, however it negatively affected the growth of hate sentiments or ideologies in the online communities [31].

The need for a safe cyberspace along with the surge of the artificial intelligence field's popularity, the circumstance drove the extensive exploration for computers to understand human language. From traditional machine learning (ML) to the current milestone of implementing DL in HS detection, though the latter being relatively more complex to be

implemented, however advancing techniques in the DL field are continually being researched and improved. The recently improved DL architectures such as neural networks, Bidirectional Encoder Representations from Transformer (BERT) has shown superiority in performance of the HS detection and other NLP-related tasks. Furthermore, the research and interest on HS detection also made feasible with publicly available dataset from diverse sources such as X, Hatebase[6], Reddit comments, etc. Although given the access to these dataset, there are still issues such as annotation biases, difference in interpretation of HS [32], and the lack of contextual information about HS. Therefore, the performance of detection model can be significantly influenced when these problems are not actively addressed.

While detection models are often evaluated through the classification report that examines the algorithm's prediction, these models tends to fail to capture the nuanced and complex nature of HS in human communication [4]. Other than that, the models has to be adaptive to constant changes and evolution of the human language, as OSM users would use language to bypass the system filtering and detection such as using acronyms, abbreviations, and sarcastic remarks [33]. Therefore, the limitations on the robustness and adaptability of these DL frameworks still hinders HS detection tasks to shift from rule-based to fully-automated on the OSM platform.

1.2 Objectives

This project will be delivering a DL model designed to detect HS in unstructured English text from OSM platforms. The model will be trained and validated, having capabilities upon deployment to classifying unstructured English statements that are user-generated based on its HS sentiment and content. The performance of the model would be evaluated against evaluation metrics to demonstrate the effectiveness and accuracy in detection of the model across different test cases.

This project aims to overcome the conventional rule-based system and limitation posed by keyword filtering and regular expression using hybrid transformer and DL architectures that is able to capture semantics and context. Upon establishing a baseline, this project further explores the impact of different data imbalance resolution strategies including data resampling and class weight loss on the model. Experimental results on the different strategies were to be performed comparative analysis on to provide insights about resampling technique on the transformer-based DL framework.

[6] https://hatebase.org/

Furthermore, this study will be assessing the text preprocessing step's importance in HS detection task to understand the effect of data preprocessing in NLP task. An ablation study will be performed to compare the experimental result from the model with preprocessing step and without preprocessing steps. A balanced performance for the model are targeted in order to avoid unfair censorship of contents on OSM that are non-hateful and just purely offensive.

1.3 Project Scope and Direction

This project will be delivering a DL model designed to detect HS in English text from OSM platforms. The model will be trained and validated, having capabilities upon deployment to classifying unstructured English statements that are user-generated based on its HS sentiment and content. The performance of the model would be evaluated against evaluation metrics to demonstrate the effectiveness and accuracy in detection of the model across different test cases.

This project includes the complete pipeline from data acquisition and data pre-processing to model training, model evaluation and model tuning to produce a capable model. It will be utilizing publicly available annotated datasets from OSM contents. The scope of the project does not fully integrate the project's model, however, focus on the abstraction of the prototype that could be adapted to applications such as upcoming microblogging services, brand-management and cyberspace safety tool.

This project does not include the full deployment and integration into real-time applications or production environment that uses OSM APIs. Given the diversity of available media from OSM site, however, multilingual texts and non-text data formats are also excluded, i.e. images, videos, emojis, or memes. The focus of the project remains in the boundary of developing the detection model rather than the user interface or OSM site moderation module.

This project encompasses the complete pipeline from data acquisition and data preprocessing to model training, model evaluation and model tuning to produce a capable model. It will be utilizing publicly available annotated datasets from OSM contents. The scope of the project does not fully integrate the project's model, however, focus on the abstraction of the prototype that could be adapted to applications such as upcoming microblogging services, brand-management and cyberspace safety tool.

1.4 Contributions

This project tackles an ongoing global issue of HS on OSM where the platform continually evolves. The development of the detection model capable of detecting unstructured HS text can be a contribution to the ultimate aspiration of a safer cyberspace for user to interacts among

one another, without having the fear of getting exposed to verbal abuse or harassment. It contributes to tools that helps building inclusive and civil communities online. The impact of the work are dedicated towards the development of tools in content moderation, user protection, legal compliance, and brand reputation management. Through the development of the DL model in this research, the project presents timely contributions towards ethical and responsible digital communications. The important contributions of the project are as follows:

- Developing and proposing DL frameworks that learns the text sentiment of HS from OSM platform, which will be capable of predicting the hate sentiment of an input text.
- DL frameworks developed will be performing multi-label classification by categorizing an input text into class of either HS, offensive language or neither HS nor offensive language.
- Perform different preprocessing strategies for transfer learning DL frameworks to compare model performance under ablation study.
- Performing resampling technique and class weight on the Davidson Dataset to address the data imbalance.
- Comparison of proposed model with existing SOTA transfer learning model.

1.5 Report Organization

This proposal for the Final Year Project is structured according to the following chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Design, Chapter 4 Experiments, Chapter 5 System Outcome and Discussion, Chapter 6 Conclusion. The first chapter provides an overview on the project research domain, the problem statement and the objectives of the project. The second chapter of this project review existing works on HS detection, related research on architectures and findings from these works. The third chapter discusses the methodologies involved with adaptation of researches in the HS detection domain. The fourth chapter will encompasses the experimental setup and configuration, showcasing the workflow of the experiments to be readily reproduced. The fifth chapter presents the results obtained through experimentation ran on Google Colab. The sixth chapter concludes the project and provides recommendation for future study in the similar domain.

CHAPTER 2 LITERATURE REVIEW

2.1 Overview of Hate Speech Detection

With the rapid growth OSM usage, especially during the COVID-19 pandemic, OSM platforms became a complementary interactive platform to fulfil user's social needs [5]. Along with the more available annotated datasets, such occurrence prompted various research on the topic of automated sentiment detection task, such as propaganda detection, sentiment polarity detection, and beyond the scope of texts, the inter-modal tasks that detects HS from media that includes both images and texts [6], [7], [8]. This section presents a comprehensive review of recent literatures that research on the HS detection on OSM platforms, surveying progression of reseacrhes in their approaches, findings and novelty.

2.1 ML-Based Works

For ML frameworks in this problem domain, S. Abro et al. [9] performed comparative analysis among ML models in HS detection. The study implemented automated text classification techniques that detects HS messages, it was found that the Term Frequency-Inverse Document Frequency (TFIDF) feature engineering method to present words in numeric vectors allows model to perform better than word2Vec and Doc2Vec. The ML classifiers explored by the study includes Naïve Bayes (NB), Random Forest (RF), Support Vector Machines (SVM), K Nearest Neighbour (KNN), Decision Tree (DT), Adaptive Boosting (AdaBoost), Multilayer Perceptron (MLP) and Logistic Regression (LR). In their task, the study found that among ML models experimented, the SVM classifier performed the best followed by AdaBoost, they concluded that this is due to SVM's ability in tackling non-linear data through utilizing kernel function, a mathematical trick [10] to transform data from simpler, low-dimensional space to a more complex dimensional space. [11]'s study performed binary classification on HS dataset from Twitter via Twitter API using ensemble methods and ML models, the model's feature selection uses the TFIDF approach and Bag of Words. The difference in their ML models were multinomial NB, and stochastic Gradient Boosting. The two best performing from their implemented classifiers are their proposed DT and stochastic Gradient Boosting accordingly, with accuracy result of 97.04% and 96.96% respectively. Anna Chiu et al. [12] study aimed to create a safer cyberspace for children and teenagers, worked on similar ML and ensemble approaches using KNN, NB, SVM, and soft-voting classifier ensemble model, targeting HS dataset to perform binary classification. Among all models, the ensemble model performed the best with 0.95 accuracy score, and NB with highest precision score.

A comparative study on was done on two different English dataset comprising of tweets, ML models included were LR, SVM, RF, NB, DR, MLP. In this study, the author also compared the ML models against DL models. Amongst all the models against two dataset, the author concluded that Bi-LSTM model performed the best when trained for 50 epochs along with word embeddings.

[25] on the other hand worked with ML classifiers on Davidson[7] HS dataset, to compare the feature extraction method of TFIDF, Bag of words (BOW) and GloVe. From the result it was found that the feature extraction significantly improves the classifier's performance. Among all the feature extraction methods, TFIDF and BOW has the higher result score of 89, 16 points higher than GloVe approach.

2.2 DL-Based Works

As opposed to the traditional ML method, the recent literature started exploring the HS detection framework through modelling DL algorithms. As revie by N. S. Mullah and W. M. Nazmee [14] in 2021 on the advances of HS detection suggested the direction of HS detection research should shift towards implementing DL models. they addressed that the number of research using DL and ensemble models is relatively less than that of traditional ML. Through the comparison of the advantages of DL and ML, such that DL will be requiring a reasonably large dataset to learn while ML would require less, however the learning of ML would stop while DL continues learning from dataset. Hence, given the rate of which user-generated data exponentially grows on OSM, DL would be better fit for this task.

[15] introduced a novel DL model of BiCHAT which integrates Hierarchical Attention-based model along with CNN and BiLSTM model to detect HS. The architecture train on tweet input by first parsing the data through a BERT to encode the tweet as a format of vector representation. The vectors were then passed to a deep CNN layer to extract the position-invariant and spatial feature. Their deep CNN has 6 layers with 256 filters of size 3, and passed through vectors are filtered into feature representation for the next phase to assign a weight variable to every feature depending on the feature importance. The attention layer compute the similarity of the feature representation and calculate an attention score that uses softmax function to produce attention-based representation. The representation was then passed to the BiLSTM, which has LSTM pair, a forward LSTM to process sequential text in one direction and a paired backward LSTM to perform in opposite direction. The BiLSTM produce a vector

[7] https://github.com/t-davidson/hate-speech-and-offensive-language

and to be passed to as attention that is low-level and dense layer that output a sigmoid non-activation function to perform classification of tweet into hate and non-hate. The authors compared the BiCHAT model with three state-of-the-art (SOTA) and six baseline models, which BiCHAT majorly outperform other models in terms of every evaluation metrics. Hyperparameters for the model is also discussed to outline the best-fitting attribute for the model, i.e. BERT as the embedding method, sigmoid function as the activation function, batch size of 32, and *Adam* as optimization algorithm. The study utilizes strengths different aspects of DL model to learn contextual dependencies, and examines hyperparameters that influences the performance of their BiCHAT model which contributes to iteration in research of this project. However, the model lacks feature to learn content and sentiment of HS. Furthermore, given the robustness of the model, the classification performance of the model on multi-label task was not covered.

Khan et al. [16] proposed their novel HCovBi-Caps model that aimed to solve contextual information understanding through the integration of convolutional, bi-directional gated recurrent unit (BiGRU), and a Capsule network. The dataset that the study uses is the Founta et al. abusive Twitter dataset [17], the dataset is firstly embedding using the GloVe[8] method to transform into vector representation, the model then pass such vector to the convolutional layer with 128 filter to extract temporal and hate-related features, and produce a feature sequence that is passed to the BiGRU. The BiGRU output a representation of the input text that is hate-incorporated as a sequence to be sent to the Capsule network layer. The Capsule layer utilizes the dynamic routing to extract additional features which is limited in CNN, and generate a vector to be output as classification through the sigmoid function. From the result of the study's experiment, the performance of the proposed model has the best performance in HS classification, followed by BiLSTM which highlights the importance of having a forward and backward learning process for the model to learn text context. Key observation on the model hyperparameters include the effectiveness of sigmoid function, the CNN filter size produces the best result with size 3 which decreases the model performance as this filter size increases, 128 hidden units on BiGRU, and *Adam* optimization algorithm. This study also poses the limitation of lacking to include the user profile-related features and sentiments of HS text.

J. S. Malik [18] performed comparative study on the GloVe and transformers embedding methods of models, namely shallow model that are MLP, SVM, extreme gradient boosting (XGB), bag of words methods that involves TFIDF, GloVe, FastText [19] and transformer

[8] https://nlp.stanford.edu/projects/glove/

methods. The experiments of HS detection were done on three datasets, Davidson dataset, Founta dataset and TSA[9] dataset. From the result, the combination BERT-based method with network-based classifier perform better in terms of detection effectiveness however more computationally costly, i.e. AI-BERT. Insights from this study also suggests that the model are not the only factor contributing to model performance, this is because shallow classifier with TFIDF has the ability to perform better than GloVe-based DL classifier. The cross-domain training of the model in this study also highlight some limitation in BERT's generalizability such that the best performer only performs well on one dataset and drastically drops in F1 score when changing predicting on other dataset.

P. Kumar et al. [20] employed six DL models including a CNN-LSTM hybrid and Autoencoder model, to detects racist and HS on Twitter dataset. The study's novelty during its approach is that during the implementation of DL classifiers, the implementation included the alternative model optimization method of quantum-based artificial hummingbird algorithm (QAHA) to determine best set of features for the DL models to classify. The resulting comparison has shown that the best performing model is the CNN-LSTM hybrid model, with 95.67% accuracy without and 98.45% with QAHA optimization. The study also found that in the particular racist and HS detection task, the hybrid model and CNN model is better performing at the task than LSTM model.

[22] proposed a hybrid LSTM+GRU to detect Twitter HS text, and the performance was compared to other existing DL frameworks. The purpose of the hybridization is to utilize the long-range dependencies handling of LSTM such that it can retain information from earlier sequence parts, and the computational efficiency offered by GRU. Such mixed architecture was argued by the author to be suitable in addressing the complexities of HS in multiple language. The proposed model also performs the HS classification task on the Davidson dataset. The study findings suggested that their model has better ability in capturing underlying speech pattern of HS, feature extraction, and showed that the sequential processing is better than processing independent instances.

[23] performed a binary classification using LSTM model on Twitter dataset, the DL model has a result of 91.14% accuracy. The study presented LSTM's suitability in the HS detection domain and compared against a baseline model of BERT-BiLSTM and was found that the proposed model in the study performed better. However, the study did not share findings on optimization of the model and lacks details on vectorization method.

[9] https://www.analyticsvidhya.com/datahack/contest/practice-problem-twitter-sentiment-analysis/##DiscussTab

A. Chhabra and D. K. Vishwakarma [24] proposed fuzzy logic classifier in the HS detection research to compare against ML and DL classifier. The justification of such approach is to balance the trade-off of classifier in model interpretation and classification performance; however, the computational cost is compromised. The model firstly produces a fuzzy set generated and a probabilistic parameterization technique for the fuzzy pattern classifier, then followed by fuzzy tree top-down classifier of which the tree node translates the feature values from node interval of the fuzzy pattern tree to an assessment of HS categorization. The result from the experimentation and comparison among implemented models shows decent performance of introducing fuzzification in the HS detection research, TFIDF also shown to be a better vectorization method in this study.

2.3 Related Work in Other Language

The current contribution to the HS detection research are mostly English and Spanish, however research in other languages also provided useful insights towards the domain of study, allowing researchers to explore SOTA models in other domain, novelty in findings and dataset in languages that are not in English.

E. Hashmi and S. y. Yayilgan [26] implemented a BiLSTM-GRU architecture that fuses with FastText embedding method, a FAST-RNN model to classify HS text in the Norwegian language. The model has undergone regularization to prevent overfitting and hyperparameters were fine-tuned using the Adam optimizer for the model to have dynamic and adaptive learning rate. The novelty of the study include using the technique of Local Interpretable Model-Agnostic Explanations (LIME), which serve to provide information about the alignment of model prediction and task requirement. Their FAST-RNN that utilizes LIME predicts on categories of HS, which are labelled as “Hateful”, “Moderately Hateful”, “Offensive”, “Provocative”, and “Neutral”, which provide robustness and greater level of sentiment analysis and could be included in future research or applied in the English language domain of HS detection. Upon implementation of LIME in their model, the model was able to perform decently well for all classification of HS as previously listed. They not only discover the performance on language-specific problem, but also included multilingual problems, which was enhanced using hyperparameter tuning and generative configuration.

[27] focused their MalHate research on vernacular HS detection in the Malayalam language. They focused modelling transformer-based classifier for the Malayalam HS detection and

resulted in a variety of BERT-based models. They constructed a dataset consisting of Malayalam language HS through data crawling with twitter APIs, and utilizes some other HS tweets to be translated into the Malayalam language using google translation API. The authors compared the result produced by each model for each phase of the data cleaning process, as expected, the result of each model gradually increased as data were cleansed and translated using google translate. The model that they found fitting for this task and performed comparatively well is the m-BERT, a multilingual BERT model that trained on Wikipedia content in multiple language sets. Similarly in the Malayalam language, [28] also obtained great result when utilizing DistilBERT, a smaller BERT model that was able to simultaneously be domain-specific and domain-agnostic through fine-tuning, which learns from the distillation loss of knowledge from larger BERT. Additionally, the approach of using translation or Code-Mix can also be found on V. Anand S et al. [29] research that modelled four DL algorithm on a wide array of BERT-based embedding and focused on detecting HS in the Tamil language. The study concluded that for the Tamil HS detection task, the combination of XLM-RoBERTa embedding, a Facebook developed multilingual trained RoBERTa, with LSTM model. Among these studies that focuses on Indian language, it was observed that the embedding method are BERT-based, which highlights the efficiency of its application in non-English language HS detection task.

2.4 Related Works

2.4.1 Advanced BERT-CNN

[21] proposed a hybrid BERT-CNN to compare against baseline model of several hybrids among FastText, LSTM, BERT and their proposed Advanced CNN which is implemented by using multiple kernels on BERT embeddings, followed by *maxpoling* and *concenate* function, to classify sentences into hate speech categories. The HS detection task is done on Davidson dataset which obtained significant F1-score on the Hate class and high accuracy scoreand TRAC-1 dataset. It was found that the feature selection offered by FastText combined with the proposed CNN model demonstrated superior performance on HS classification. This study reported in detail the scores obtained from the experiment on its BERT-CNN architecture achieving SOTA scores on the Davidson Dataset, hence this project will be evaluating results against this study.

2.5 Research Gaps and Summary

From the reviewed literature, it was shown that the recent study to perform HS detection in OSM platform has gradually transitioned from traditional ML to the current surge in research of DL. However, the majority number of existing study are focuses on the ML techniques and studies that involve DL is relatively lower. Hence, this highlights the need of further examining and understanding DL model's performance in HS detection task.

Moreover, the studies that involved DL regardless of the language of dataset also highlighted the superiority of the utilization of the pre-trained BERT embedding method for DL models to understand the word context as opposed to existing embedding methods such as GloVe or Word2Vec. Hence the DistilBERT embedding method will be adapted and experimented to be evaluated during the model's implementation.

Lastly, most of the study that proposed a novel approach during the HS detection has shown superiority of hybrid model in performing HS model, which can be implemented by initially using an DL algorithm followed by another layer of model, such approach were to be implemented in order to verify the efficacy of such approach and propose a combination of hybrid that shows superiority among implemented model in the HS detection task for English language.

CHAPTER 3 SYSTEM MODEL

3.1 System Model

This section presents the proposed system implemented, which aims to classify the dataset's tweet into three classes, which are "hate speech", "offensive language", and "neither hate speech nor offensive language". As the nature of the project is NLP, whereby the computers are explored and utilized to manipulate and attain understanding natural language text or speech [30].

The proposed methodology, as outlined in the flowchart (Figure 3.1.1), describes the sequential process for acquiring the results of five distinct experiments conducted on the proposed DL framework of this project. To initiate the pipeline, the overall workflow begins with the acquisition of the raw dataset through identification of suitable dataset that fits the criteria of attributing three distinct classes of tweets extracted from OSM platform. The dataset is then subjected to decision of performing text preprocessing steps to prepare the text data for analysis, or the data is extracted directly by skipping the preprocessing step to perform ablation study on the proposed model.

After that, the dataset is branched to either resample or keep its original class distribution, this is the step to address the data imbalance in the dataset (See Section 3.2). Following this, the dataset is partitioned into distinct subsets namely training, validation and testing dataset to facilitate robust model training and evaluation in later stages. The subsequent stages focus on the core modeling process, which initialize the DistilBERT transformer model, and transfer knowledge to DL layers to fine tune the model, catering to the classification task of this project.

The class weights are then computed to be decided to be paired with the model training, penalizing the model on experiments that involves class weight training. The subsequent stage obtain the result from the model's classification to be evaluated. With the pipeline ending with five distinct experimental results from the automated pipeline. This structured approach ensures that the model development lifecycle is systematic, repeatable, and designed to produce reliable results.

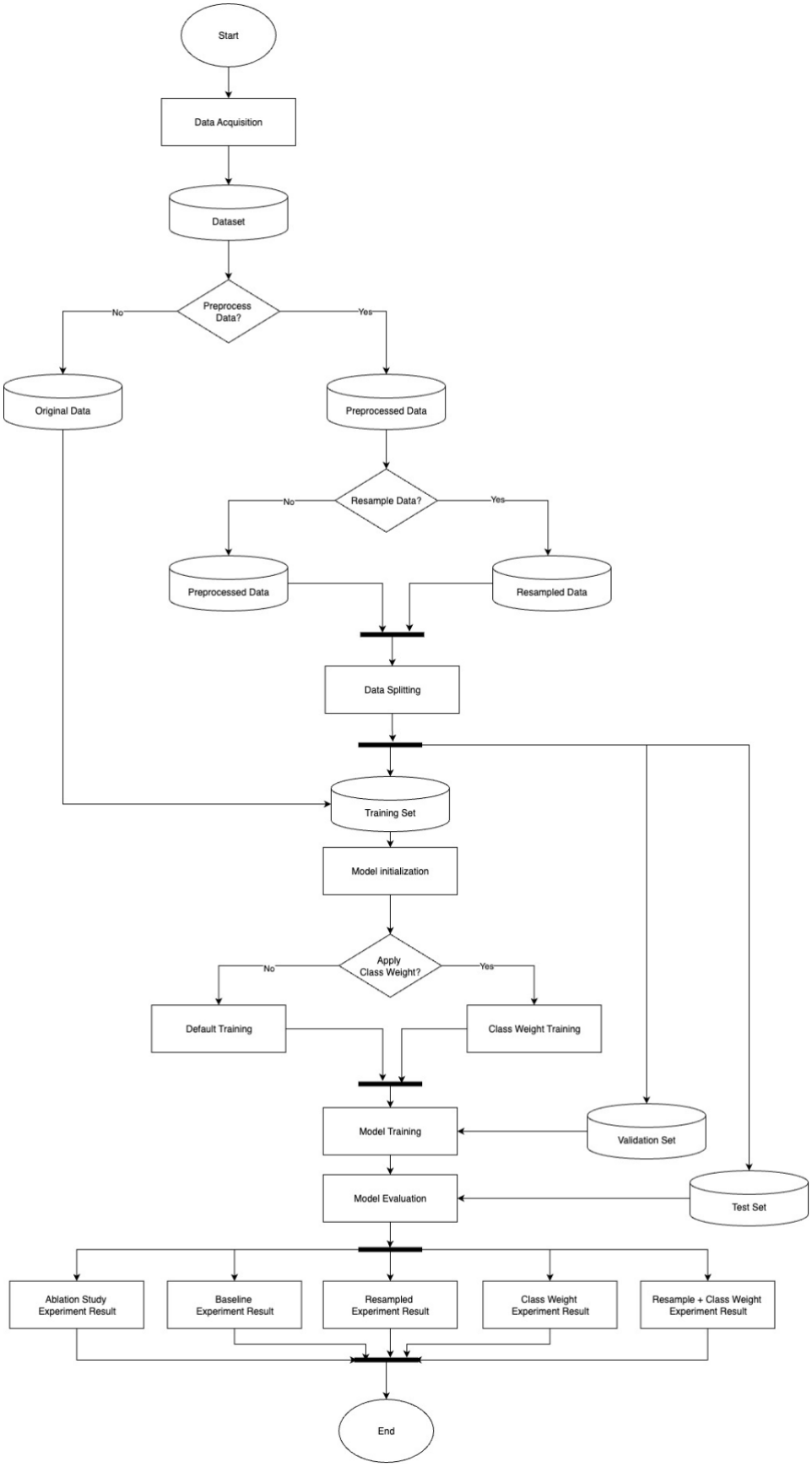


Figure 3.1.0.1 Flowchart of Project Pipeline

3.1.1 Data Acquisition

In this study, the HS dataset collected are tweets that were compiled and labelled by CrowdFlower. The dataset is multilabel, contains a total of 24,782 instances with three different target classes to represent the hate sentiment in tweets, the data annotations are tabulated in 3.1.1.1 as follows:

No.	Attribute Name	Description	Data type
1	count	Number of CrowdFlower (CF) users that coded for each tweet, more CF users coded a tweet when judgements were deemed unreliable by CF.	Integer
2	hate_speech	CF user count judging a tweet as HS.	Integer
3	offensive_language	CF user count judging a tweet as offensive language	Integer
4	neither	CF user count judging a tweet as neither offensive nor non-offensive.	Integer
5	tweet	Statements extracted from Twitter.	Object
6	class	Class label of tweet. (0 represents HS, 1 represents offensive language, and 2 represents neither classes)	Integer

3.1.1.1 Data Annotation of the Davidson Twitter HS Dataset

3.1.2 Exploratory Data Analysis (EDA)

Following the text preprocessing stage, the tweets are represented as *processed_tweets*. To gain insights into the dataset, the most frequently occurring words within these processed tweets are illustrated in Figure 3.1.2.1 and Figure 3.1.2.2. In addition, the distribution of tweet lengths after preprocessing is presented in Figure 3.1.2.3 and Figure 3.1.2.4. These visualizations provide an understanding of the dataset textual characteristics and distribution, serving as foundation for modeling tasks.

Figure 3.1.2.1 Word Cloud of Hate Speech in Processed Tweets

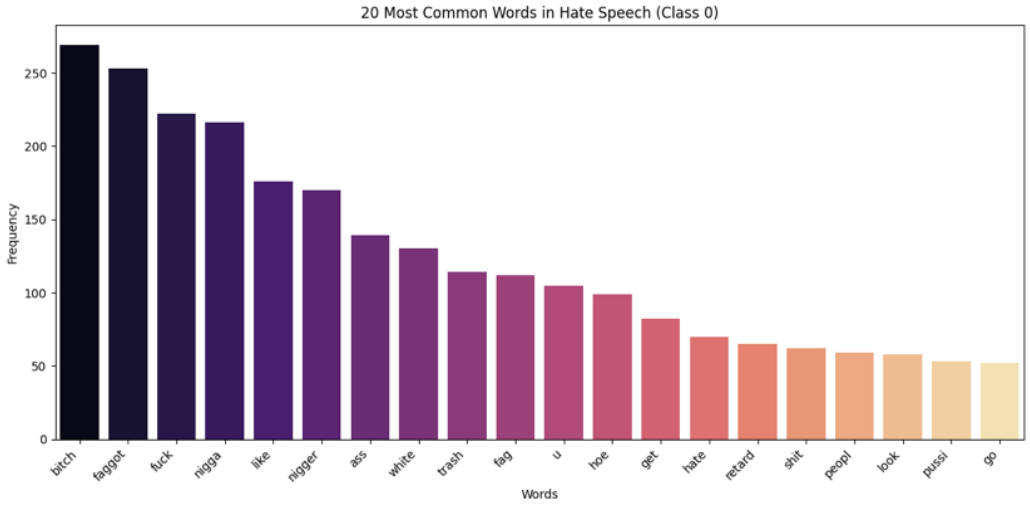


Figure 3.1.2.2 Twenty Most Common words of Hate Speech in Processed Tweets

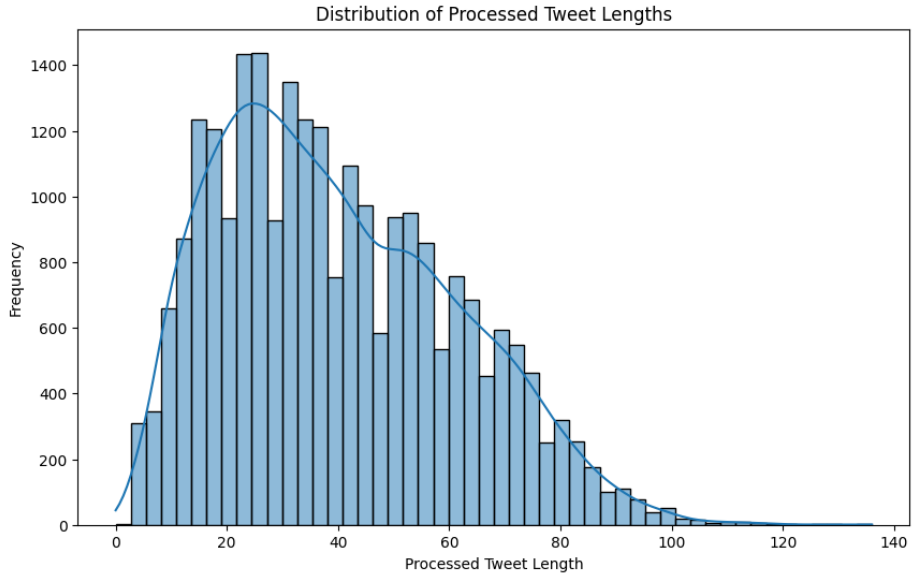


Figure 3.1.2.3 Distribution of Processed Tweet Lengths

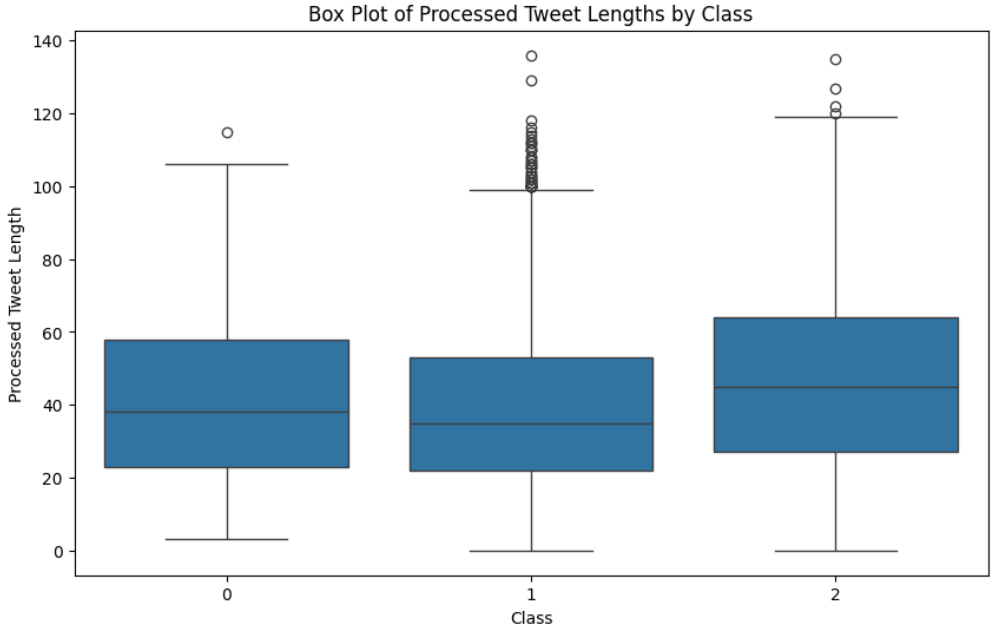


Figure 3.1.2.4 Box Plots of Processed Tweet Lengths (By Class)

3.2 Data Distribution

Figure 3.2.0.1 shows the distribution of the target classes in the dataset, which is found to be having imbalance, having a record frequency for class 0, 1, and 2 of 1,430, 19,190, and 4,163 counts of records respectively. The distribution of the dataset is tabulated in Table 3.1.2.1.

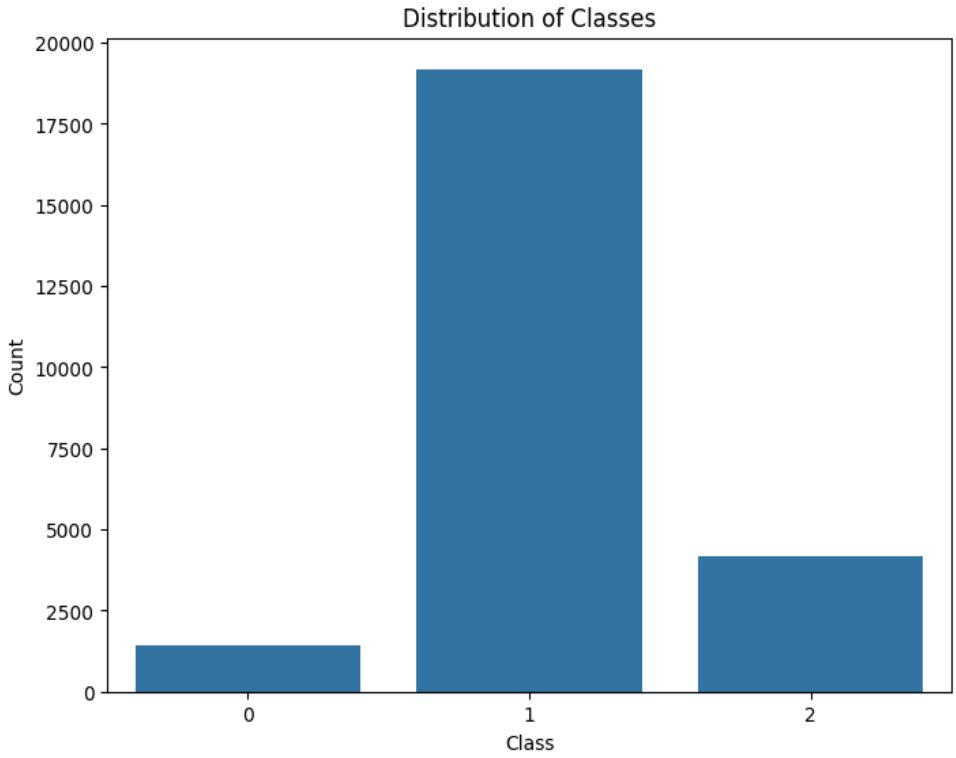


Figure 3.2.0.1 Distribution of the Target Classes in Davidson Dataset

Target Class	Description	Frequency	Proportion (%)
0	HS	1430	5.77
1	Offensive language	19190	**77.43**
2	Neither HS nor offensive language	4163	16.80
Total		24,783	100.00

3.1.2.1 Proportions of Existing Target Classes and Respective Frequency Count

The distribution of such dataset can be a potential issue for the model training to classify the HS, as it was evident that the target class (class 0) of HS has relatively and significantly lower percentage of tweets (5.77%), while the offensive language class (class 1) accounts for the major portion of the data (77.43%). Such imbalance would potentially cause bias when the models are predicting especially towards the majority class of offensive language. Although such imbalance occurrence is a normal and reflects well in the context of attaining real-word data, it can however lead to biased decision such as overfitting of data due to classifier favoring towards the major class samples during training without proper treatment prior to training the model [62].

3.3 Data Preprocessing

3.3.1 Text Preprocessing

Text pre-processing is an essential phase in NLP tasks, in the dataset, the original values of the tweets are noisy and contains information that are non-informative. Therefore, customized algorithm was implemented to transform text in the raw form to structure that could be effectively analyzed. For example, the uncleaned data has punctuations, symbols, word like "RT" that symbolizes a retweet. In order to achieve preparation of a cleaned baseline samples, the tweet of the dataset is applied the programming routines as follows:

- Elimination of double whitespaces: All the double whitespaces are removed manually to standardize the code.
- Elimination of symbols and punctuation: Punctuations such as commas, exclamation marks are removed.
- Removing Uniform Resource Locator (URL): URLs that are associated with each tweet are directory towards external link, for example, "https://www.example.com" could be found throughout the *tweet* records in this HS detection task. Since the URLs do not have informative purposes hence, they are also removed.
- Normalization of text: The tweets are normalized by changing all texts in the tweets to lowercase form.
- Lemmatization: Reducing the words originally in the dataset into their respective root form.

With the preprocessing algorithm, the Davidson dataset is ready for additional analysis, cleansing data existing in the tweet that are not useful while retaining the unique features of the texts. The cleaned data is then ready to proceed to the next phase in the workflow. The current stage is important in ensuring that the model training fully utilize only texts that are crucial to understand the underlying hate sentiments in the HS dataset and perform accurate classification. The pseudocode of the data pre-processing for the Davidson Dataset will be following the convention of Algorithm 1.

Algorithm 1: Preprocess (D_{tweet})

BEGIN

$D_{tweet} \leftarrow lowercase(D_{tweet})$

$D_{tweet} \leftarrow remove\ 'RT', mentions\ and\ URLs\ from\ D_{tweet}$

$D_{tweet} \leftarrow remove\ punctuation\ and\ numbers\ from\ D_{tweet}$

$D_{tweet} \leftarrow remove\ double\ whitespace\ from\ D_{tweet}$

$D_{token} \leftarrow tokenize(D_{tweet})$

$D_{stem} \leftarrow Lemmatize(D_{token})$

$D_{cleaned} \leftarrow join\ D_{token} \& D_{stem}$

Return $D_{cleaned}$

END

3.3.2 Ablation Study

The impact of text preprocessing on the outcome of classification on HS is perform in this project through an ablation study. Such approach is achieved by removing the data preprocessing step from the NLP pipeline. By isolating the preprocessing contribution to the final model's accuracy, precision and F1-score will be analysed to spot for differences with baseline models. The importance of this step is to attain understanding in NLP researches about components of an architecture that are contributing towards optimized model performance while avoid unnecessary computational resource usage [63].

3.4 Data Resampling

The dataset used in this project is imbalanced, with one or more classes having significantly fewer samples compared to others. Class imbalance can bias a model toward predicting the majority class, thereby reducing the effectiveness of classification for minority classes. To mitigate this issue, resampling strategies are commonly employed. Some conventional approaches in addressing imbalanced class distribution of dataset are done as follows:

- Oversampling: Increasing the minority classes sample count, either by simple duplication or by generating synthetic samples such as with the Synthetic Minority Oversampling Technique (SMOTE) [53].
- Undersampling: Reducing the number of samples in the majority class to balance the class distribution, such as through Random Under-Sampling (RUS) [54].

Although SMOTE is widely applied in imbalanced classification tasks, it was not adopted in this project because the dataset comprises textual data. SMOTE operates by interpolating feature vectors in a continuous space, which is appropriate for numeric datasets but may

produce semantically invalid synthetic samples in natural language tasks. Therefore, Random Oversampling was preferred to ensure balanced class distribution without introducing incoherent text representations. This method duplicates existing samples from underrepresented classes until all classes have the same number of samples. The primary advantage of this approach is that it allows the model to generalize better for minority classes without discarding potentially useful information from the majority class.

After applying random oversampling, the dataset was balanced such that each target class (hate speech, offensive language, and neutral) had an equal number of samples. The balanced class distribution is shown in Figure 3.4.0.1 and numerical figures in 3.4.0.1

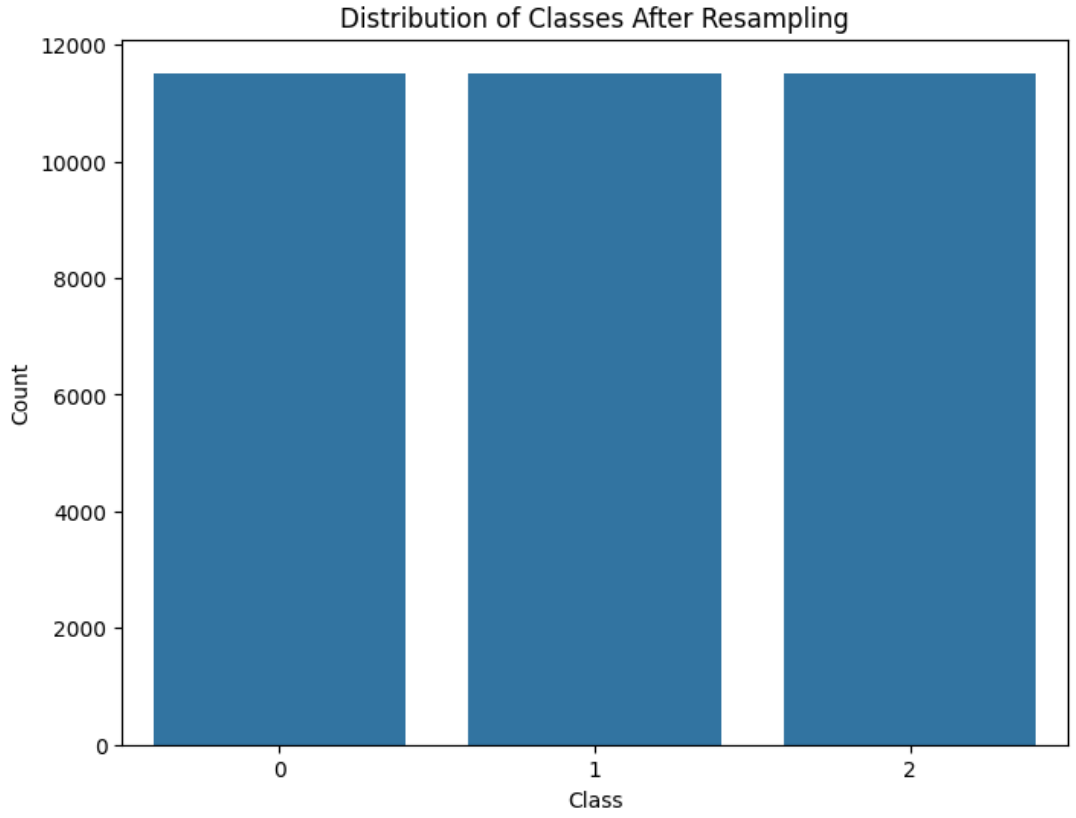


Figure 3.4.0.1 Distribution of the Target Classes in Training Dataset after Oversampling

Target Class	Description	Frequency	Proportion (%)
0	HS	11,513	33.33
1	Offensive language	11,513	33.33
2	Neither HS nor offensive language	11,513	33.33
Total		34,539	100.00

Table 3.4.0.1 Balanced Training Set Distribution after Performing oversampling

3.5 Data Splitting

In supervised ML, a fundamental step in the methodology involves partitioning the dataset into distinct subsets to ensure reliable model evaluation. In this project, the tweet dataset was divided into three subsets: training, validation, and testing in the ratio of 60:20:20 accordingly [35]. This split helps the learning of model parameters from the training set, while the validation set monitors performance of generalization and help in hyperparameter optimization. Such approach reduces the memorization or overfitting during model training.

The test set, on the other hand, will be an hold-out set, exclusively used for evaluation once model training has been completed. Since the test data are not employed during training or validation, the performance obtained will be the unbiased estimate of the model's effectiveness on new data. Relying solely on a train-test split, while repeatedly tuning hyperparameters based on the test performance, can lead to overfitting to peculiarities within the test partition and an overly optimistic assessment of generalization ability.

A diagnostic perspective suggests evaluating how representative a given partition is of the overall data distribution and how sensitive model performance is to perturbations in the split. Random splitting, although commonly used, has been shown to sometimes produce misleadingly optimistic results if the partitions are not representative or if information leakage occurs [55].

3.6 Class Weight Loss

Class imbalance presents a common challenge in hate speech detection, as the minority class (e.g., hate speech) is significantly underrepresented compared to the majority classes. One widely adopted approach to address this issue is the use of class weight loss, where higher penalties are assigned to misclassified samples of minority classes during training. This ensures that the learning process does not become biased toward the majority class, thereby improving the model's ability to detect underrepresented categories [56].

In this project pipeline, class weights were calculated from the training set applied during the training stage, the numeric figures of class weights are tabulated in Table 3.6.0.1, as shown the

computed class weight is inverse to its class proportion. This approach complemented alternative imbalance handling strategies such as resampling, allowing a comparative assessment of their effectiveness.

Class	Class Weight	Proportion (%)
0 (Hate Speech)	5.7766	5.77
1 (Offensive Language)	0.4304	**77.43**
2 (Neither)	1.9841	16.80

Table 3.6.0.1 Class Weights for Each Classes in the Dataset

3.7 Proposed Model

To achieve the objective of the paper, developing a DL framework utilizing transformer-based framework while designing the architecture for multiclass classification, this section discusses the model selection for the task at hand, specifically tailoring the DL framework to the Davidson dataset's characteristics. While BERT is part of the model, this project also aims to hybridize the transformer with DL layers, fine tuning the pretrained model with combination of CNN, LSTM and Dense layers to further make the architecture more adaptive. This section will thoroughly introduce the selected model's layers, and hyperparameters, to justify the design choices and discusses how the architecture fit to the task. An overview of the designed model architecture is illustrated in Figure 3.7.0.1.

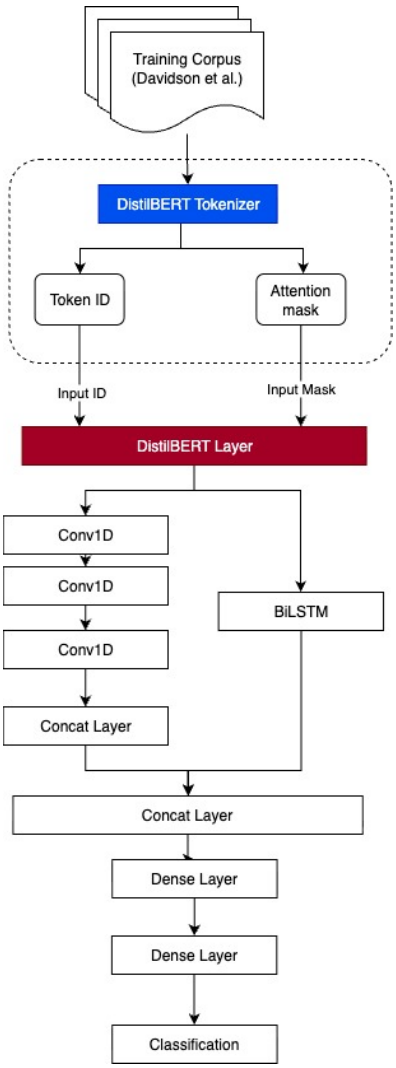


Figure 3.7.0.1 Proposed DistilBERT-CNN-BiLSTM Architecture

3.7.1 BERT

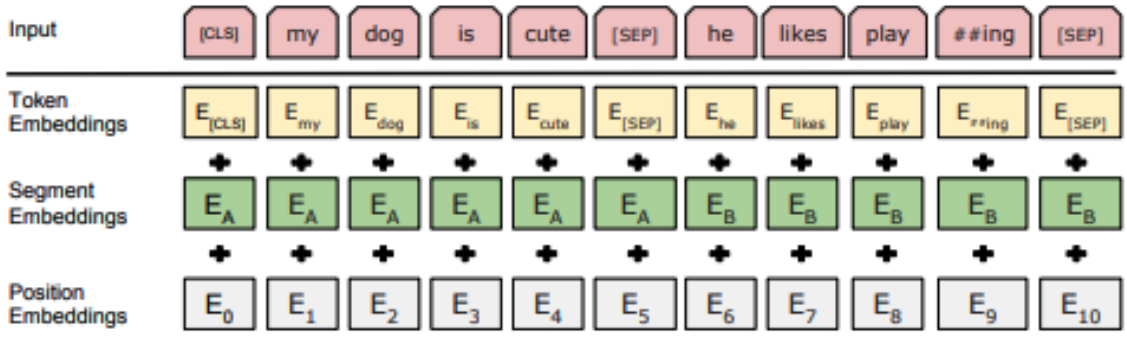


Figure 3.7.1.1 BERT input representation [44]

With the rising popularity of transformer architecture in current DL framework's progression, the project adapts BERT-based transformer in the fine-tuned model for the classification task.

To introduce the architecture adapted for this project, BERT is a transformer-based language model that utilizes self-attention mechanism to learn bidirectional context instead of conventional unidirectional, i.e. from left to right or vice versa on language modelling, allowing nuanced textual semantic relationships to be captured [44]. The BERT architecture consists of stacked transformer encoder layers, each comprising multi-head self-attention mechanisms that appended by feed-forward neural networks connected with residual path and layer normalization.

Input text is first processed using *WordPiece* tokenization, with each token embedding combined with positional and segment embeddings. The tokenization step for BERT architecture are distinctive as the first token will be a [CLS] token which helps performing aggregation on the classification task, while a separate [SEP] token serves to separate sentences. For HS detection, this representation is passed through a dense layer with *softmax* activation to generate class probabilities corresponding to hate speech, offensive language, or neutral categories.

In this project, the BERT model adapted will be experimenting on a variant of BERT that has reduced parameters, the model will be discussed in the followed section.

3.7.2 DistilBERT

Recent advancements have explored efficient variants of BERT, notably DistilBERT, which reduces the number of transformer layers by half as shown in Figure 3.7.2.1, while retaining approximately 97% of the performance of the original model [52]. DistilBERT achieves this by employing knowledge distillation during pre-training, where a student network is trained to reproduce the behavior of the larger BERT teacher. A triple loss function—comprising masked language modeling loss, distillation loss on soft target distributions, and cosine embedding loss on hidden states—ensures that the distilled model inherits both predictive power and representational structure from the teacher. Additionally, redundant components such as token-type embeddings and the pooler are removed, leading to a model that is 40% smaller and up to 60% faster at inference.

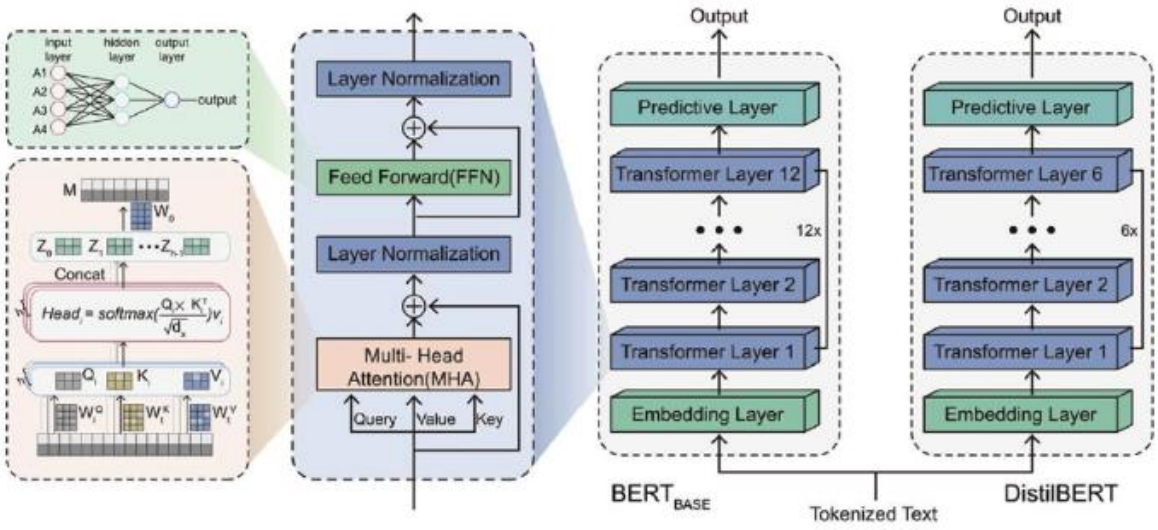


Figure 3.7.2.1 Diagram Comparing BERT base and DistilBERT Architecture [57]

In the context of hate speech detection, where models may need to process high volumes of short texts from online social media in real time, efficiency is critical. Deploying transformer architectures such as BERT or its distilled variants enables effective contextual understanding while balancing accuracy with computational feasibility. While the full BERT model provides deeper contextual representations that can capture subtle and implicit forms of hate speech, smaller models like DistilBERT offer practical trade-offs in environments with limited resources or latency constraints. To ensure the experimental computational efficiency and reduce the required amount of parameter to train the hybrid model, while simultaneously preserve BERT transformer's feature extraction, the DistilBERT variant is applied instead in this project's proposed DL framework. With the DistilBERT as the initial transformer handling the input data, other DL layers are added prior to the classification to make the model more robust and perform better in classifying HS. The followed up sections discusses CNNs, Bidirectional LSTM and Dense layers, which are components of the proposed architecture.

3.7.3 CNN Branch

CNNs are DL model that are applied widely in the fields of computer visions, speech processing, and face recognition. The CNN architecture has several key components that helps in its efficiency in image classification and feature extraction task. Firstly, the convolutional layer are kernels that process the input, performing dot products prior to producing feature maps. Secondly, the kernels has random weights dynamically adjusted during training to make feature extraction. The pooling layer follows, reducing dimension of the feature maps to reduce

spatial dimensions and preserves essential information. The common pooling methods for CNN include max pooling, average pooling, and global variants, the method applied for this project will be adapting the max pooling method. The memory efficiency of CNNs is attributed to its sparse connectivity and weight as opposed to fully connected (FC) networks. The fourth component is the activation function which allows non-linearity, allowing CNNs to understand complex relationships through the decision of neurons activation according to input values. After convolution and pooling, the flattened feature maps is processed by FC layer to produce output. Finally, a loss function calculates actual and predicted outputs error, adjusting the learning process by reducing this error during backpropagation. Overall, CNNs offer robust learning with fewer parameters, improved generalization, and efficient training compared to traditional neural networks [39].

3.7.4 Bidirectional LSTM

LSTM is a sub-category of RNN which processes sequential data using three layers which are the input layer, hidden layer and the output layer. Firstly, The input layer reads a words sequence and performs transformation of words into a vector representation that is continuous. Such transformation results in vector sequence to be an input for hidden layer. The representation of word sequence as vector allows for greater understanding of dependencies and semantic relationships. Secondly, The hidden layer processes data by taking the vector representation of the current word as input and the hidden state from the previous iteration. The hidden state is as a memory of the processed data at the current point, and the state is updated to include new information. Such mechanism allows temporal dependencies to be captured and patterns along the sequence to be learnt. Thirdly, the hidden state is used by the output layer to compute the probability distribution of the consecutive word in the sequence. This probability distribution is follows the preceding word context, which enables the model to predict the likelihood of different words appearing next [40]. .LSTM similarly has three gates but applies the sigmoid function for the previous hidden state and input. The first gate of LSTM determine the portion of data, which is added to state of the cell by sigmoid function. Secondly, the forget gate which decide information based on the hidden state and current input to be discarded. Lastly, the output gate utilizes the cell state parts according to the current input and perform update on the outputting cell state[41].

In this project, a Bidirectional LSTM (BiLSTM) is employed, which processes the sequence in both forward and backward directions, thereby capturing past and future context

simultaneously. This dual perspective enhances semantic understanding in textual data, such as tweets, where the meaning of a token often depends on both its preceding and succeeding words [58]. The output from the BiLSTM branch serves as a complementary representation to the CNN branch, enabling the hybrid model to leverage both local feature extraction and sequential context modeling.

3.7.5 Concatenation

To combine the strengths of the CNN and BiLSTM branches, their outputs are concatenated into a single feature vector. Concatenation is a widely adopted technique in multimodal or hybrid neural networks as it integrates heterogeneous feature representations into a unified space [59]. In this framework, the CNN branch contributes local n-gram and phrase-level features, while the BiLSTM captures global contextual dependencies across sequences. The concatenated vector thus provides a richer representation of the input text, which is subsequently passed to the fully connected layers for classification.

3.7.6 Dense Layer and Classifier

Following concatenation, the feature vector is processed by dense layers (FC layers) to enable higher-level abstraction and decision-making. Non-linear activation functions, specifically ReLU, are employed to introduce non-linearity and improve model expressiveness. Dropout regularization is applied to mitigate overfitting by randomly deactivating neurons during training [60].

The final classification is performed using a dense output layer with a softmax activation function, which converts logits into class probability distributions. The model is trained using a weighted categorical cross-entropy loss (Section 3.8), enabling the classifier to account for class imbalance. This design ensures that the system can effectively differentiate between hate speech, offensive but not hateful speech, and neutral content, consistent with the labelling scheme of the Davidson dataset.

To summarize, the proposed model, as recorded in Algorithm 2, starts by loading the pretrained DistilBERT model, which is the foundation of word embeddings. The input tweets are tokenized [36] and encoded into token IDs and attention masks. These representations are passed into DistilBERT to generate contextual embedding that are dense and capture semantic relationships between words. The output embeddings are then fed into two branches: a CNN

branch for extracting local n-gram level features depending on the filter size, and a BiLSTM branch for the model to understand sequential dependencies in both directions. The outputs from both branches are concatenated to form a combined feature representation, then processed through dense layers and lastly a classifier layer with a SoftMax activation to predict the class labels. The function returns the tokenizer to be utilized during experimentations.

Algorithm 2: Modelling DistilBERT-CNN-BiLSTM

BEGIN

$DEFINE\ CustomDistilBertLayer$:

$LOAD\ pre-trained\ DistilBERT\ model\ (distilbert-base-uncased)$

$Forward\ pass$:

$INPUT \leftarrow (input\_ids, attention\_mask)$

$OUTPUT \leftarrow last\ hidden\ state\ from\ DistilBERT$

$LOAD\ DistilBERT\ tokenizer\ (distilbert-base-uncased)$

$DEFINE\ inputs$:

$input\_ids \leftarrow tensor\ of\ shape\ (batch\_size, 52)$

$attention\_mask \leftarrow tensor\ of\ shape\ (batch\_size, 52)$

$PASS\ inputs\ through\ CustomDistilBertLayer$

$sequence\_output \leftarrow contextual\ embeddings\ (batch\_size, seq\_len, hidden\_size = 768)$

$CNN\ branch$:

$conv3 \leftarrow Conv1D\ (kernel = 3, ReLU) \rightarrow GlobalMaxPool1D$

$conv5 \leftarrow Conv1D\ (kernel = 5, ReLU) \rightarrow GlobalMaxPool1D$

$conv7 \leftarrow Conv1D\ (kernel = 7, ReLU) \rightarrow GlobalMaxPool1D$

$cnn\_out \leftarrow Concatenate(conv3, conv5, conv7)$

$BiLSTM\ branch$:

$lstm\_out$
$\leftarrow Bidirectional\ LSTM(64\ units)\ applied\ to\ sequence\_output$

$CONCATENATE\ cnn\_out\ and\ lstm\_out$

$ADD\ classifier$:
$Dense(256, ReLU) \rightarrow Dropout(0.3)$
$Dense(128, ReLU) \rightarrow Dropout(0.3)$
$Dense(3, Softmax)$

$COMPILE\ model\ with$:
$optimizer \leftarrow Adam\ (learning\_rate = 2e - 5)$
$loss \leftarrow Sparse\ Categorical\ Crossentropy$
$metric \leftarrow Accuracy$

$RETURN\ model, tokenizer$
END

3.8 Model Training

3.8.1 Model Hyperparameter

The hyperparameters for the experiments are tabulated in Table 3.8.1.1.

Category	Hyperparameter	Value / Setting
Tokenizer	Model	`distilbert-base-uncased`
	Max sequence length	52
	Padding / Truncation	`max_length`
Optimizer	Type	Adam
	Learning rate	2e-5
CNN Branch	Filters	64
	Kernel sizes	3, 5, 7

	Activation	ReLU
	Pooling	GlobalMaxPooling1D
BiLSTM Branch	Units	64
	Direction	Bidirectional
	Return sequences	False
Dense Layers	Layer 1 units	256
	Layer 2 units	128
	Activation	ReLU
	Dropout	0.3 (each layer)
Output Layer	Units	3 (classes)
	Activation	Softmax
Training	Loss	Sparse categorical cross-entropy
	Epochs	30 (with early stopping, patience=5)
	Batch size	16
Regularization	Early stopping	Monitor: `val_loss`, patience=5, restore best weights=True
Imbalance Handling	Resampling	RandomOverSampler (optional)
	Class weights	Balanced (optional)

Table 3.8.1.1. Experiment Hyperparameter Settings

The training process implemented maximal sequence length of 52 tokens in the tokenizer to be consistent with practices in transformer-based architectures. Training was carried out for 30 epochs with an early stopping criterion of five epochs to mitigate overfitting, while the batch size was fixed at 16, balancing gradient stability and computational feasibility [61].

The activation function of Rectified Linear Unit (ReLU) is applied in the CNN layers, converting whole values of input to a positive number, which introduces low load on computational time, generally computed through Eq. 1 where x is the neuron input. Consequently, the output layer unit represent the number of target number, since the target class in this project is three hence the value defined. The output layer activation function is the Softmax Loss function utilized for multi-label classification problem. The output of the function compute probability $p \in \{0,1\}$ from vector of real-valued score, calculated through Eq. 2, where N is output layer neuron number and e^{a_i} is the previous layer's non-normalized

output. Dropout[10] is feature selection method to drop neurons randomly during every training epoch, and the value is the unit fraction to be dropped. The optimization algorithm used is the Adam optimizer, a stochastic gradient-based optimization which computes the learning rate using estimates [43]. Sparse categorical cross entropy is a complementary loss function for models with class-dependent weights, expressed in Eq. 3, which computes the inverse to the class frequency proportion and gives greater importance to underrepresented class [34].

$$f(x)_{ReLU} = \max(0, x) \tag{1}$$

$$p_i = \frac{e^{a_i}}{\sum_{j=1}^{N} e_k^a} \tag{2}$$

$$L = \sum_{j=1}^{N} \sum_{j=1}^{N} w_c y_{ic} \log(\hat{y}_{ic}) \tag{3}$$

3.8.2 Model Training

In the model training phase the model parameters will be iteratively optimized to minimize loss function. In this project, the training process passes DistilBERT embeddings to hybrid DL branches consisting of CNN and BiLSTM layers, followed by dense layers and a SoftMax classifier. The model uses the Adam optimizer with class-weighted sparse categorical cross-entropy loss. Hyperparameters such as batch size, learning rate, and dropout rate are carefully tuned to enhance generalization performance, early stopping is used to prevent overfitting. All five of the experiments configured in the project adheres to the workflow in Algorithm 3.

Algorithm 3: Running Experiment

BEGIN

Load preprocessed dataset D_train, D_val

Set hyperparameters:

$batch\_size = 16$

$max\_length = 52$

$epochs = 30$

$patience = 5$

$learning\_rate = 2e - 5$

[10] https://keras.io/api/layers/regularization_layers/dropout/

$Initialize\ DistilBERT\ encoder\ \rightarrow\ BERT\_Embedding$

$Initialize\ CNN\ branch\ \rightarrow\ Conv1D(filters = 64, kernel\_size = [3,5,7])\ +\ MaxPooling$

$Initialize\ BiLSTM\ branch\ \rightarrow\ BiLSTM(units = 64)$

$Concatenate(CNN\_output, BiLSTM\_output)$

$Pass\ through\ Dense(256)\ +\ ReLU\ +\ Dropout(0.3)$

$Pass\ through\ Dense(128)\ +\ ReLU\ +\ Dropout(0.3)$

$Output\ layer\ \rightarrow\ Dense(3)\ +\ Softmax$

$Define\ loss\ function\ \rightarrow\ SparseCategoricalCrossEntropy(class\_weights)$

$Define\ optimizer\ \rightarrow\ Adam(learning\_rate)$

$FOR\ epoch\ =\ 1\ to\ epochs\ DO$

  $FOR\ each\ batch\ (X\_batch, y\_batch)\ in\ D\_train\ DO$

    $Compute\ predictions\ y\_pred$

    $Compute\ loss\ =\ Loss(y\_batch, y\_pred)$

    $Backpropagate\ gradients$

    $Update\ model\ weights\ with\ optimizer$

  $END\ FOR$

  $Compute\ validation\_loss\ on\ D\_val$

  $IF\ validation\_loss\ not\ improved\ for\ patience\ epochs\ THEN$

    $Stop\ training\ (Early\ Stopping)$

  $END\ IF$

$END\ FOR$

$Save\ best\ model\ weights$

$Return\ trained\_model$

ENDEND

3.9 Model Evaluation

Performing evaluations on the experimented models is a critical stage to validate the effectiveness and reliability of the proposed DL models for hate speech detection. This stage involves the systematic assessment of model outputs against annotated ground truth labels using established performance metrics and diagnostic tools. The goal of this phase is not only to measure predictive accuracy but also to understand the model's strengths and limitations across different classes, particularly given the inherent imbalance in hate speech datasets. By incorporating multiple evaluation measures such as the confusion matrix, precision, recall, F1-score, accuracy, AUC, and Cohen's Kappa, the evaluation framework provides a comprehensive perspective on both the discriminative power and fairness of the models. This ensures that the developed system is not only optimized for performance but is also robust, generalizable, and suitable for real-world application in online social media moderation.

The confusion matrix as shown in Figure 3.9.0.1 is utilized to gain visualizations of the prediction results of the model in It categorizes outcomes into four groups: true positives (TP), where hateful content is correctly classified; true negatives (TN), where non-hateful content is correctly recognized; false positives (FP), where neutral or offensive content is incorrectly flagged as hateful; and false negatives (FN), where hateful content is missed by the classifier.

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figure 3.9.0.1 Confusion Matrix

The evaluation of the classifiers' performance on the test dataset includes the calculation of various conventional performance metrics to measure the precision and efficacy of the DL classifier model. During testing of each model, tasks were to be performed are producing predictions, evaluating the accuracy, precision, recall, and F1 score, and constructing the confusion matrix. In the context of HS classification, precision is to measure how many HS

statements have been correctly identified among classified tweets, calculated according to Eq. 3. The recall is to measure the percentage of HS statements have been correctly identified among actual HS, given in Eq. 4. The F1-score is calculated using Eq. 5, which is the harmonic mean between precision and recall. Lastly, the accuracy score calculates the percentage of correctly categorized tweets to their respective target classes, computed using Eq. 6 [15]. Lastly, as the data imbalance may influence model bias, the Cohen's Kappa score provides alternative scoring to compute the actual hit of the model prediction rather than hit by chances [38], through Eq. 7 where x_{ii} represents the cases count the confusion matrix main diagonal, N is the examples count, $x_{.i}$is the column total count and $x_{i.}$ is the row total count.

$$P_r = \frac{TP}{TP + FP} \tag{3}$$

$$R_c = \frac{TP}{TP + FN} \tag{4}$$

$$F_S = 2 \times \frac{P_r \times R_c}{P_R + R_C} \tag{5}$$

$$AC = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

$$Kappa = \frac{N \sum_{i=1}^{k} x_{ii} - \sum_{i=1}^{k} x_{i.} . x_{.i}}{N^2 - \sum_{i=1}^{k} x_{i.} . x_{i}} \tag{7}$$

3.10 Chapter Summary

The proposed system model for hate speech detection follows a structured workflow beginning with dataset acquisition, preprocessing, resampling, model training, and evaluation. The dataset used is the Davidson Twitter Hate Speech dataset, consisting of 24,782 tweets labelled into three categories: hate speech, offensive language, and neither, though with notable class imbalance. Exploratory data analysis was performed to examine text distributions and patterns, followed by preprocessing steps such as tokenization, lemmatization, and the removal of noise like punctuation, URLs, and stop-words. To address data imbalance, oversampling techniques and class weights computations were applied to increase minority class representation and penalize the model on misclassifications based on class distribution respectively. The core classification models implemented were DistilBERT, a transformer-based architecture leveraging attention mechanisms and contextual embeddings, it provides a more lightweight yet efficient alternative to BERT architecture. Model performance was evaluated using a range of metrics including accuracy, precision, recall, F1-score, and Cohen's Kappa to account for

imbalance, along with confusion matrices for classification visualization and the AUC score for discriminative power. This structured methodology ensures systematic, reliable, and generalizable outcomes in classifying tweets across the three categories.

CHAPTER 4 EXPERIMENT/SIMULATION

4.1 Hardware Setup

The experiments for this project were conducted using cloud-based computational resources rather than physical hardware. Specifically, Google Colab[11] was employed, which provides access to Graphics Processing Units (GPUs) suitable for DL workloads specifically DistilBERT in this project with high number of parameters. The GPU allocated during experimentation was typically an NVIDIA Tesla T4 with 16 GB of VRAM. This setup is widely used for research and prototyping due to its accessibility and ability to handle Transformer-based models such as DistilBERT efficiently.

In addition to the GPU, the Colab environment includes CPU cores and RAM sufficient for data preprocessing and model training. While Colab resources are shared and may vary depending on availability, the provided hardware consistently supported all experimental runs without the need for dedicated high-performance computing infrastructure.

The specifications of the hardware used and the average training duration are summarized in Table 4.1.1

Resource Type	**Specification**	**Description**
GPU	NVIDIA Tesla T4	16 GB VRAM, CUDA-enabled
CPU	Colab allocated (variable)	Supports preprocessing
Training time	~10–30 minutes per experiment	Depends on dataset configuration

Table 4.1.0.1 Hardware Specifications and Training Duration on Google Colab

Training times were relatively efficient under this configuration. On average, each experiment required approximately 10 to 30 minutes to complete, depending on the dataset variant and balancing strategy used where resampling takes longer duration due to a higher count of data. This rapid turnaround facilitated iterative experimentation, including hyperparameter adjustments and ablation studies.

[11] https://colab.research.google.com/

Overall, the use of cloud-based GPU resources provided a practical and cost-effective platform for conducting the experiments in this project, eliminating the need for physical servers or specialized local hardware.

4.2 Software Setup

The software environment for this project was established using a Python-based (See Appendix A) DL stack within the Google Colab platform. The platform provides a managed Jupyter Notebook interface with pre-installed libraries and GPU support, making it a convenient choice for DL research and experimentation. In addition to the default Colab environment, several packages were installed to support data preprocessing, natural language processing (NLP), model training, and evaluation.

4.2.1 Python Environment and Dependencies

The experiments were conducted using Python 3.11 in Google Colab. All dependencies were installed directly through the `pip` package manager. The following command was used to install the required packages:

```
!pip install tensorflow transformers pandas scikit-learn nltk spacy imbalanced-learn
```

The installed libraries served the following purposes:

- TensorFlow: Core DL framework used for model training and evaluation.
- Transformers (Hugging Face)**:** Provided pre-trained models such as DistilBERT for text embeddings and fine-tuning.
- Pandas**:** Used for dataset loading, cleaning, and manipulation.
- Scikit-learn: Supported train-test splits, evaluation metrics, and preprocessing utilities.
- Natural Language Toolkit (NLTK): Assisted in text preprocessing (e.g., tokenization, stopword handling).
- spaCy: Provided additional NLP preprocessing, such as lemmatization.
- imbalanced-learn**:** Enabled oversampling and resampling techniques to address class imbalance.

For spaCy, an English language model was additionally downloaded:

```
!python -m spacy download en_core_web_sm
```

This model was used for text tokenization, lemmatization, and linguistic preprocessing tasks.

Component	Purpose
Python	Programming environment
TensorFlow	Deep learning framework
Hugging Face Transformers	Pre-trained Transformer models (e.g., DistilBERT)
Pandas	Data handling and manipulation
Scikit-learn	Evaluation metrics, preprocessing, and utilities
NLTK	NLP preprocessing
spaCy	Tokenization, lemmatization, linguistic processing
imbalanced-learn	Oversampling and class imbalance handling
Jupyter/Colab	Notebook interface and GPU integration

Table 4.2.1.1 Software environment and libraries

4.3 Setting and Configuration

This section outlines the experimental configurations and hyperparameter settings adopted for the hate speech detection task. Multiple setups were designed to evaluate the impact of data preprocessing, resampling, and loss balancing strategies. In addition, the choice of hyperparameters was guided by both prior literature on Transformer fine-tuning and empirical considerations such as dataset characteristics and computational resources.

4.3.1 Experimental Configurations

Five experimental setups were implemented to examine different preprocessing and balancing strategies. The details are summarized in Table 4.3.1.1.

Experiment Name	**Data Function**	**Resample?**	**Use Class Weights?**	**Description**
Base (Processed Data)	get_train_test_data	False	False	Standard with preprocessing

Resampled	get_train_test_data	True	False	Oversampling for balance
Class Weights	get_train_test_data	False	True	Weighted loss
Resampled + Class Weight	get_train_test_data	False	True	Oversampled balanced data paired with weighted loss
Ablation (Raw Data)	get_ablation_study_data	False	False	No preprocessing

Table 4.3.1.1 Experimental configurations

These configurations allow systematic comparison of preprocessing and balancing strategies. In particular, the Resampled and Class Weights setups address class imbalance through two different mechanisms, while the Ablation experiment provides insight into the contribution of preprocessing steps.

Hyperparameter	Value	Justification
Max sequence length	52	Chosen to cover the maximum tokenized length in the dataset, avoiding truncation while minimizing padding overhead.
Epochs	30	Sufficient training time; convergence expected earlier, with EarlyStopping to prevent overfitting.
Batch size	16	Balances GPU memory usage and gradient stability. Smaller batches also improve generalization.
Optimizer	Adam	Widely adopted optimizer for NLP fine-tuning due to adaptive learning rate.
Learning rate	2e-5	Standard small value for Transformer fine-tuning to ensure stable convergence without catastrophic forgetting.
EarlyStopping patience	5	Prevents unnecessary computation by halting when validation loss fails to improve for 5 epochs.

Table 4.3.1.2 Hyperparameters for the proposed DL framework

The learning rate was set at 2×10^{-5}, a common default in Transformer fine-tuning literature, as it balances convergence speed with stability. The batch size of 16 was chosen due to GPU

memory constraints and because smaller batch sizes can improve generalization. Training was capped at 30 epochs, although in practice convergence typically occurred earlier. To avoid overfitting, EarlyStopping with a patience of 5 epochs was employed.

The sequence length of 52 tokens was selected based on the distribution of tweet lengths in the dataset. This ensured full coverage of the majority of samples while minimizing computational cost from unnecessary padding. The Adam optimizer was adopted due to its proven effectiveness in NLP fine-tuning tasks, particularly in conjunction with adaptive learning rates.

Together, these settings form a balanced configuration that emphasizes stability, efficient convergence, and generalization across different experimental setups.

4.4 System Operation

The operational methodology for this research was devised as a multi-stage workflow, encompassing dataset preparation, model training, and performance evaluation. This entire process was systematically executed within the Google Colab environment, which facilitated both the iterative development of the computational models and the subsequent visualization of the results. The following sections provide a chronological and sequential account of each phase of the system's operation, automated programmatically training each model to classify each experimental setting's configuration accordingly.

4.4.1 Loading dataset

The first stage in the system operation involves loading the Davidson Twitter dataset into the Google Colab environment. As illustrated in Fig. X, the dataset is imported in CSV format using the Pandas library, which provides efficient handling of structured data. Each record in the dataset comprises the tweet text along with its corresponding label indicating the class: hate speech, offensive language, or neutral. During this step, an initial inspection is carried out to verify the integrity of the data, ensuring that there are no missing entries or malformed records. The successful execution of this process guarantees that the dataset is correctly structured for subsequent preprocessing operations, such as tokenization, text normalization, and splitting into training, validation, and testing subsets.

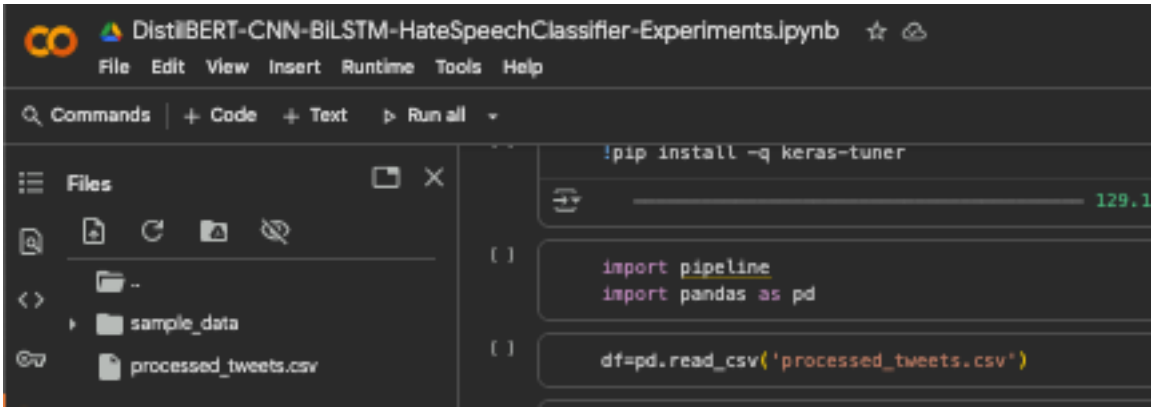
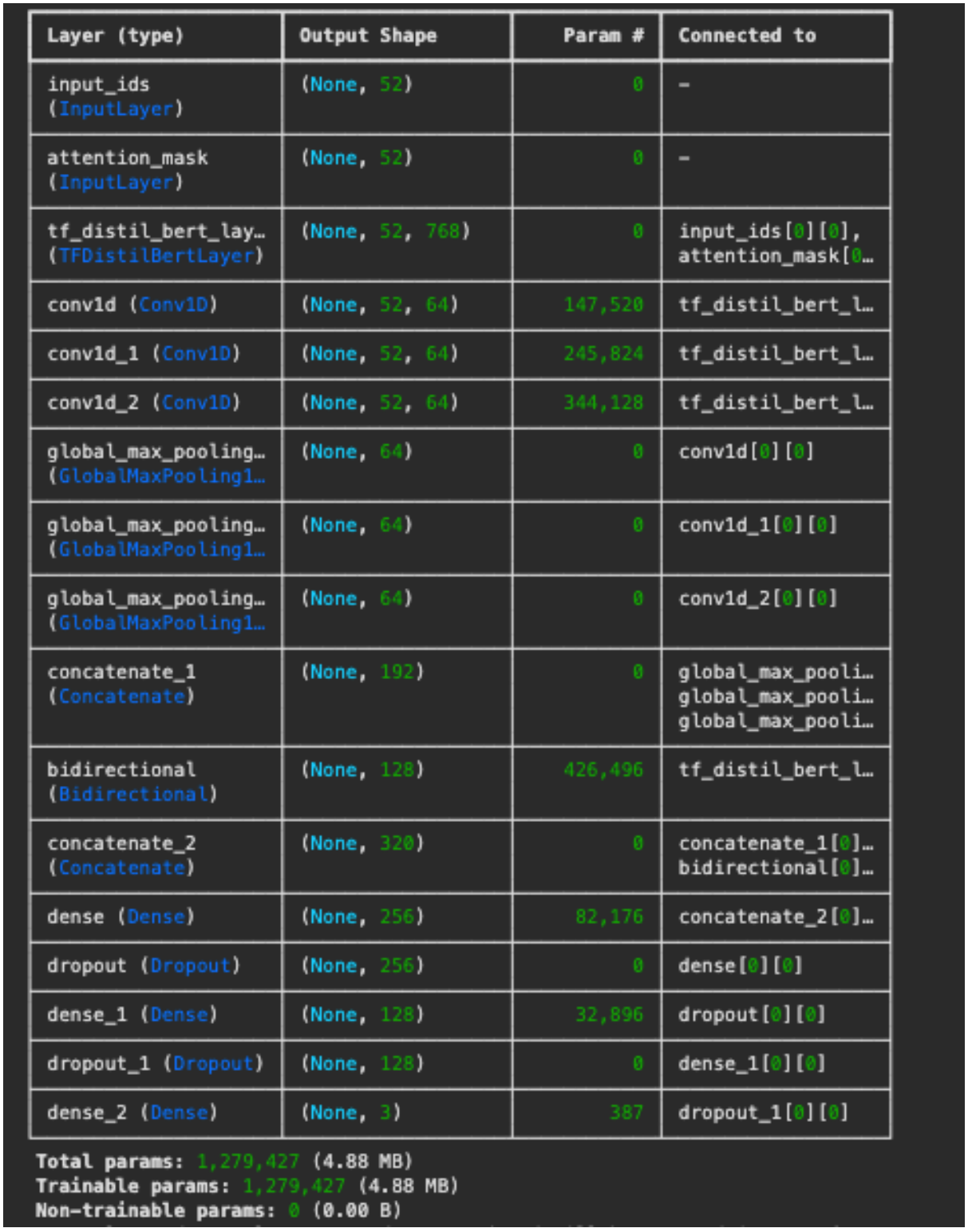


Figure 4.4.1.1 Snapshot of Data Loading in Google Colab

4.4.2 Model Summary

The architecture of the proposed model was systematically defined and compiled using the TensorFlow Keras framework. To validate the structural configuration, model summary function was executed, generating a comprehensive overview of each layer, including type, output shape, and the number of trainable parameters. The complete snapshot of this summary is presented in Figure X. The total parameter count was recorded at 1,279,427, highlighting the reduced complexity of the DistilBERT backbone compared to the standard BERT architecture, while still retaining sufficient representational capacity for downstream classification. This compact design ensures efficiency in both training and inference phases, with lower computational and memory costs.



Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	(None, 52)	0	-
attention_mask (InputLayer)	(None, 52)	0	-
tf_distil_bert_lay… (TFDistilBertLayer)	(None, 52, 768)	0	input_ids[0][0], attention_mask[0…
conv1d (Conv1D)	(None, 52, 64)	147,520	tf_distil_bert_l…
conv1d_1 (Conv1D)	(None, 52, 64)	245,824	tf_distil_bert_l…
conv1d_2 (Conv1D)	(None, 52, 64)	344,128	tf_distil_bert_l…
global_max_pooling… (GlobalMaxPooling1…	(None, 64)	0	conv1d[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 64)	0	conv1d_1[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 64)	0	conv1d_2[0][0]
concatenate_1 (Concatenate)	(None, 192)	0	global_max_pooli… global_max_pooli… global_max_pooli…
bidirectional (Bidirectional)	(None, 128)	426,496	tf_distil_bert_l…
concatenate_2 (Concatenate)	(None, 320)	0	concatenate_1[0]… bidirectional[0]…
dense (Dense)	(None, 256)	82,176	concatenate_2[0]…
dropout (Dropout)	(None, 256)	0	dense[0][0]
dense_1 (Dense)	(None, 128)	32,896	dropout[0][0]
dropout_1 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 3)	387	dropout_1[0][0]

Total params: 1,279,427 (4.88 MB)
Trainable params: 1,279,427 (4.88 MB)
Non-trainable params: 0 (0.00 B)

Figure 4.4.2.1 Snapshot of Data Loading in Google Colab

4.4.3 Running Experiments

The execution of experiments was automated through the custom pipeline, where each configuration was invoked using scripted function to run each experiment. For this study, the

first experimental setup involved the hybrid model, as illustrated in Figure 4.4.3.1. The pipeline automatically handled dataset preparation, model instantiation, training, and evaluation.

During the training phase, the model was optimized over 30 epochs, with key metrics such as accuracy and loss monitored for both training and validation sets. A snapshot of the training progression is provided in Figure 4.4.3.2, where initial epochs showed steady improvements in generalization performance. As optimization continued, the model achieved a training accuracy of 0.9420 with validation accuracy stabilizing at 0.8864 by Epoch 15. Correspondingly, the training loss decreased from 0.6233 to 0.1589, while validation loss improved from 0.4154 to 0.3290. These results indicate convergence of the model training, with validation performance reflecting positive generalization and controlled overfitting.

```
# Exp 1: Baseline
all_results["Experiment 1"] = pipeline.run_experiment(df,
    exp_name="DistilBERT-CNN-BiLSTM",
    get_data_fn=pipeline.get_train_test_data
)
```

Figure 4.4.3.1 Snapshot of Executing Experiment in Google Colab

```
TensorFlow and JAX classes are deprecated and will be removed in Transformers v5. We recommend migrating to PyTorch classes or pinn
Epoch 1/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 61s 54ms/step - accuracy: 0.7692 - loss: 0.6233 - val_accuracy: 0.8483 - val_loss: 0.4154
Epoch 2/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 78s 53ms/step - accuracy: 0.8520 - loss: 0.4196 - val_accuracy: 0.8634 - val_loss: 0.3598
Epoch 3/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 92s 64ms/step - accuracy: 0.8694 - loss: 0.3625 - val_accuracy: 0.8701 - val_loss: 0.3508
Epoch 4/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 83s 64ms/step - accuracy: 0.8803 - loss: 0.3391 - val_accuracy: 0.8794 - val_loss: 0.3278
Epoch 5/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 81s 64ms/step - accuracy: 0.8893 - loss: 0.3111 - val_accuracy: 0.8822 - val_loss: 0.3207
Epoch 6/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 73s 54ms/step - accuracy: 0.8912 - loss: 0.3020 - val_accuracy: 0.8830 - val_loss: 0.3174
Epoch 7/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 81s 54ms/step - accuracy: 0.9000 - loss: 0.2829 - val_accuracy: 0.8858 - val_loss: 0.3150
Epoch 8/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 82s 53ms/step - accuracy: 0.9068 - loss: 0.2624 - val_accuracy: 0.8850 - val_loss: 0.3172
Epoch 9/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 59s 64ms/step - accuracy: 0.9132 - loss: 0.2486 - val_accuracy: 0.8858 - val_loss: 0.3192
Epoch 10/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 83s 65ms/step - accuracy: 0.9168 - loss: 0.2307 - val_accuracy: 0.8858 - val_loss: 0.3108
Epoch 11/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 81s 64ms/step - accuracy: 0.9253 - loss: 0.2202 - val_accuracy: 0.8860 - val_loss: 0.3114
Epoch 12/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 84s 67ms/step - accuracy: 0.9351 - loss: 0.1979 - val_accuracy: 0.8860 - val_loss: 0.3355
Epoch 13/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 79s 64ms/step - accuracy: 0.9348 - loss: 0.1842 - val_accuracy: 0.8874 - val_loss: 0.3198
Epoch 14/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 59s 64ms/step - accuracy: 0.9409 - loss: 0.1761 - val_accuracy: 0.8725 - val_loss: 0.3788
Epoch 15/30
930/930 ━━━━━━━━━━━━━━━━━━━━ 72s 53ms/step - accuracy: 0.9420 - loss: 0.1589 - val_accuracy: 0.8864 - val_loss: 0.3290
```

Figure 4.4.3.2 Snapshot of Model Training History in Google Colab

4.4.4 Evaluation Report and Classification Results

The evaluation of the DistilBERT–CNN–BiLSTM architecture was performed on the unseen test set. The input tensors had dimensions were printed presenting both the input IDs and the attention masks for the number of samples in the test set, reflecting the tokenized sequence length of 52 tokens per instance. Quantitative performance for each experiment following the similar summary convention in Figure 4.4.4.1, presenting test loss and test accuracy. Furthermore a classification report is generated using reporting the model's precision, recall, F1-score and support for each classes. The Kappa score is then computed based on the test result, and confusion matrix is lastly generated to show the model's classification result, as shown in Figure 4.4.4.1.

```
Input IDs shape: (4957, 52)
Attention Mask shape: (4957, 52)

DistilBERT-CNN-BiLSTM Evaluation:
Test Loss: 0.3132
Test Accuracy: 0.8832

Classification Report:
              precision    recall  f1-score   support

           0     0.5000    0.2797    0.3587       286
           1     0.9318    0.9325    0.9322      3839
           2     0.7518    0.8630    0.8036       832

    accuracy                         0.8832      4957
   macro avg     0.7279    0.6917    0.6982      4957
weighted avg     0.8767    0.8832    0.8775      4957

Cohen's Kappa Score: 0.6805

Confusion Matrix:
[[  80  158   48]
 [  70 3580  189]
 [  10  104  718]]
```

Figure 4.4.4.1 Snapshot of Model Testing Result in Google Colab

4.5 Implementation Issues and Challenges

During the development and experimentation phases, several technical and methodological challenges were encountered. These challenges primarily related to computational constraints due to the robustness of the model implemented, nature of the dataset with its imbalanced characteristics with majority classes overly represented, and preprocessing overhead. Each challenge and its corresponding mitigation strategy is outlined below.

4.5.1 GPU Memory Limitations

One of the main issues faced during model training was the GPU memory limitation in the Google Colab environment. The allocated NVIDIA Tesla T4 GPU (16 GB VRAM) is sufficient for most medium-sized models but can become restrictive when handling Transformer architectures with long sequences and large batch sizes. In certain runs, out-of-memory errors were triggered during training.

To mitigate this, the training pipeline incorporated memory management techniques such as explicitly clearing the environment cache, freeing unused memory between experimental runs. Additionally, conservative hyperparameter choices, i.e. batch size of 16 and sequence length of text for the model to predict on is capped at 52 were adopted to ensure execution stability without exhausting GPU resources from the virtual environment.

4.5.2 Class Imbalance

The dataset used in this project exhibited a high degree of class imbalance, with hateful (tweet class 0) or neither offensive nor hate speech (tweet class 2) underrepresented relative to the majority class (tweet class 1). This imbalance adversely affects model training, as the model may become biased toward predicting the dominant class. To address the problem mentioned, this project employed the following balancing strategies and their combination:

- Random oversampling of minority classes to artificially balance the training data.
- Class-weighted loss functions to penalize misclassification of minority categories more heavily.

These methods, when applied individually and in combination, were systematically evaluated to determine their effectiveness in improving model robustness against imbalance.

4.5.3 Preprocessing Overhead

Text preprocessing introduced notable computational overhead. In particular, the use of spaCy for lemmatization and NLTK for tokenization and stopword removal slowed down preprocessing when applied to large datasets. While these libraries provide high-quality NLP tools, their performance can become a bottleneck in resource-limited environments such as Google Colab.

This challenge was mitigated by preprocessing the dataset once and saving intermediate results, rather than repeating the operations for every run. However, preprocessing remained time-intensive relative to model training, highlighting a trade-off between linguistic quality and computational efficiency.

4.5.4 Dataset Bias

A further limitation was the bias inherent in the dataset. The corpus was restricted to English tweets collected from a single source. Consequently, the trained models may not generalize well to other languages, domains, or communication platforms. This introduces potential bias in performance evaluation and limits the broader applicability of the system.

While addressing dataset bias was beyond the immediate scope of this study, the limitation is acknowledged. Future work could incorporate multilingual or cross-platform datasets to enhance generalizability.

4.5.5 Summary of Issues and Challenges

In summary, the main implementation challenges included GPU memory constraints**,** class imbalance**,** preprocessing overhead**,** and dataset bias**.** Each issue required careful handling to ensure stable and meaningful experimentation. Although mitigations were applied where feasible, certain limitations such as dataset bias remain open problems and form opportunities for future extensions of this work.

4.6 Concluding Remark

The experimental setup was discussed, along with hardware and software configuration, and the various strategies implemented to address dataset imbalance, preprocessing requirements, and computational constraints. Despite the challenges encountered during development such as GPU memory limitations, preprocessing overhead, and dataset bias, the implementation phase has been completed and experimental results were obtained and ready for analysis. All experimental configurations, including baseline, resampling, class weighting, and ablation studies, were automated in the Python script and ran in the Google Colab environment.

The models were trained, validated and tested under controlled environment, to make sure the project's reproducibility and scalability across different configurations. At this stage, although the system has been fully experimented and prepared for deployment-level testing, however the applicability of these implementations is only assessable through model's performance evaluations.

The next chapter presents the evaluation and analysis of results obtained from the experiments. This transition marks the shift from system implementation to performance assessment, where the impact of preprocessing strategies, resampling, and hyperparameter choices on hate speech detection will be thoroughly analysed.

CHAPTER 5 SYSTEM EVALUATION AND DISCUSSION

5.1 Experimentation and Results

This section presents the projects findings and analysis on the results obtained from different experimental settings. Evaluation were done on the proposed model, DistilBERT-CNN-BiLSTM to assess its performance through numerical and graphical outcome generated after model training.

5.1.1 Experiment I: Base Model

The first experiment establishes a baseline by evaluating the proposed DistilBERT–CNN–BiLSTM hybrid model without the incorporation of any imbalance handling techniques or additional optimization strategies. This configuration serves as a reference point against which subsequent modifications and enhancements can be assessed. The experiment focuses on examining the model's natural learning capacity when trained directly on the imbalanced dataset, thereby highlighting inherent strengths and weaknesses in capturing minority class patterns. The outcomes of this baseline evaluation provide a foundation for analyzing the impact of resampling, class weighting, and ablation studies presented in later experiments

Figure 5.1.1.1 shows the training and validation, accuracy and loss result across its multiple epochs for the baseline model, in this setting, the preprocessing stage is preserved. From the diagram, the model achieved a high accuracy score throughout the iteration of the epochs, with its training loss gradually reducing through the training routine. The validation routine on the other hand shows that the model parameter is fitting moderately well on the data throughout the training, as the validation and training accuracy are roughly the similar with validation accuracy subpar to training accuracy, showing the model generalize well with not too much overfitting.

From the confusion matrix in Figure 5.1.1.2, the baseline model correctly classified on the test set 80 out of 286 for class 0 test samples indicating the model was affected by the data imbalance as the baseline does not address the issue, followed by 3580 out of 3839 samples for class 1 and 718 from 832 class 2 samples. The AUC score achieved by the baseline model, achieving 0.87, 0.94, and 0.97 for class 0, 1 and 2 accordingly as shown in Figure 5.1.1.3.

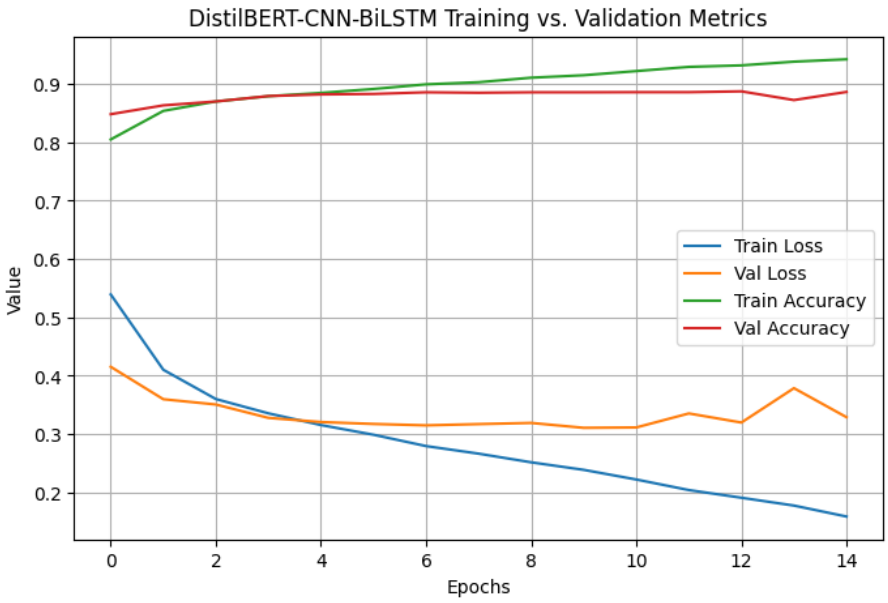


Figure 5.1.1.1 DistilBERT-CNN-BiLSTM Baseline Accuracy and Loss Plot for Training and Validation

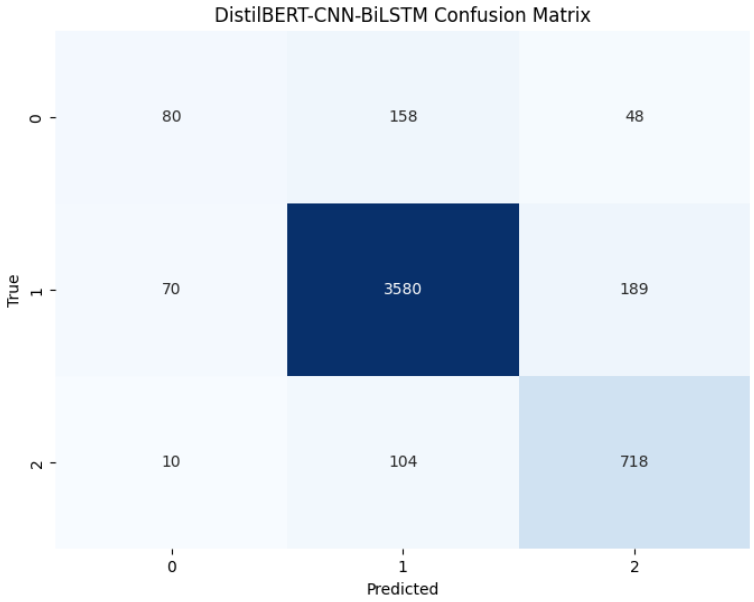


Figure 5.1.1.2 DistilBERT-CNN-BiLSTM Baseline Confusion Matrix

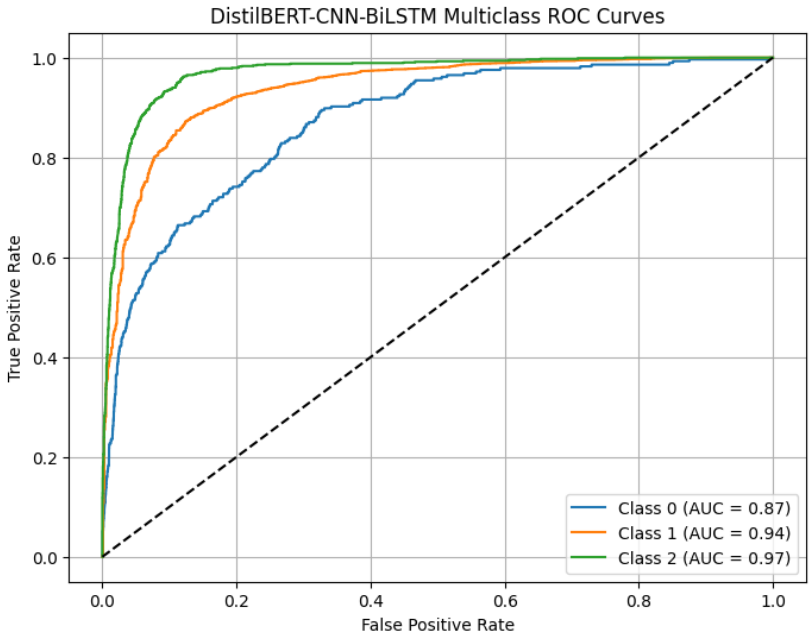


Figure 5.1.1.3 DistilBERT-CNN-BiLSTM Multiclass ROC Curve

5.1.2 Experiment II: Resampled Model

The second experiment investigates the impact of addressing class imbalance through resampling techniques. Specifically, the minority class instances were oversampled to achieve a more balanced distribution across categories. This strategy aims to improve the model's recall and F1-score for underrepresented classes, which are often overlooked in imbalanced datasets. The performance of the resampled model is compared with the baseline to assess improvements in capturing minority class signals.

Figure 5.1.2.1 is the training and validation, accuracy and loss result across resampled model multiple epochs history. From the diagram, the model achieved a high accuracy score throughout the training iterations, achieving near perfect accuracy on training as it approach the end of training, with its training loss showing steep descent through the training routine. However, the validation routine on the other hand shows that the model parameter is struggling to adapt well on the data throughout the training, as the validation loss and accuracy scores are fluctuating throughout, until the end of training where the model has achieved high training score, the validation score remains a gap, this indicate the model has overfit its generalization, due to the resampled data that are repetitive rather than newly synthesized points of data

From the confusion matrix in Figure 5.1.2.2, the baseline model correctly classified on the test set 113 out of 286 for class 0 test samples showing improvement from baseline due to the exposure of the model to balanced data, followed by 3515 out of 3839 samples for class 1

and 710 from 832 class 2 samples. The AUC score achieved by the baseline model, achieving 0.85, 0.93, and 0.97 for class 0, 1 and 2 accordingly as shown in Figure 5.1.2.3.

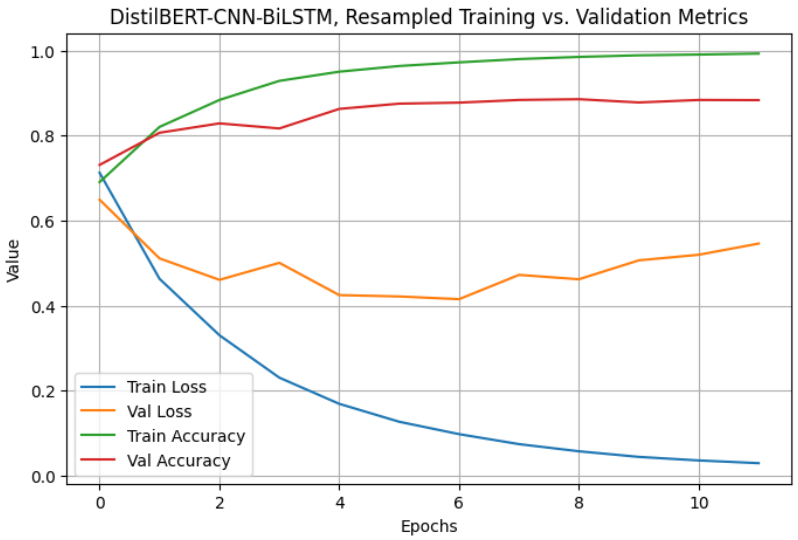


Figure 5.1.2.1 DistilBERT-CNN-BiLSTM Resampled Accuracy and Loss Plot for Training and Validation

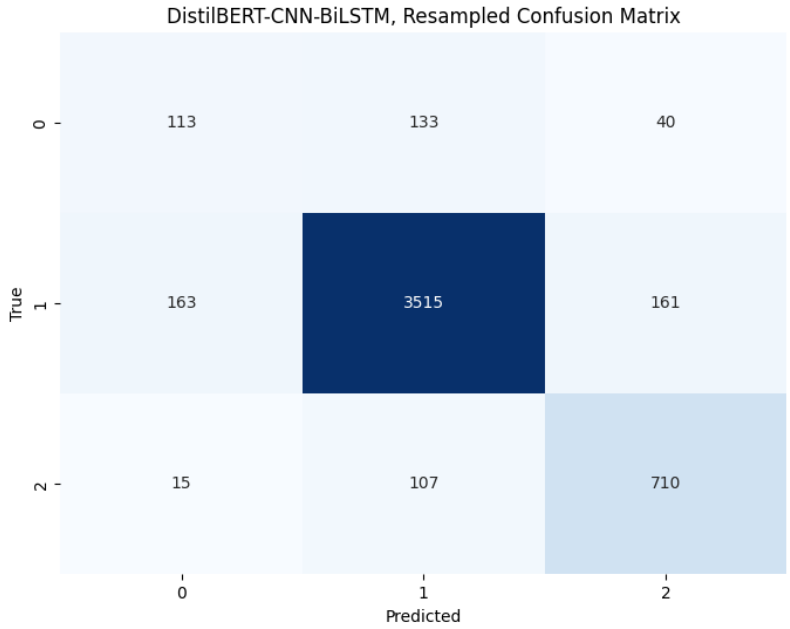


Figure 5.1.2.2 DistilBERT-CNN-BiLSTM Resampled Confusion Matrix

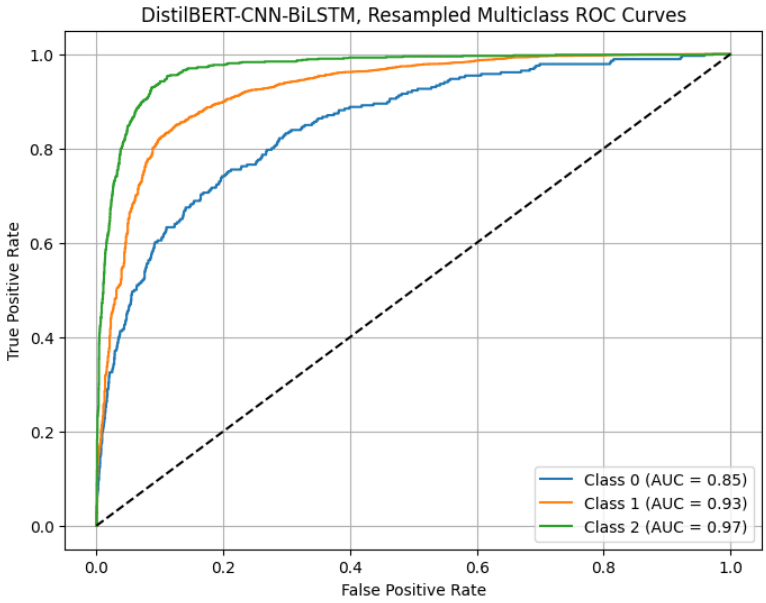


Figure 5.1.2.3 DistilBERT-CNN-BiLSTM Resampled Multiclass ROC Curve

5.1.3 Experiment III: Class Weights Model

In the third experiment, class weights were introduced into the training process to penalize misclassification of minority classes more heavily than that of majority classes. By integrating class-dependent weighting directly into the loss function, the model is guided to pay greater attention to underrepresented instances during optimization. This approach provides an alternative to resampling while maintaining the original dataset distribution. The results are analysed relative to the baseline to determine whether class weighting enhances fairness across categories.

Figure 5.1.3.1 shows the training and validation, accuracy and loss result across its multiple epochs for the model with class weight training. From the diagram, the model achieved a high accuracy score throughout the iteration of the epochs, with its training loss gradually reducing through the training routine. The validation routine on the other hand shows that the model parameter is fitting moderately well on the data throughout the training, as the validation and training accuracy are roughly the similar with validation accuracy subpar to training accuracy, showing the model generalize well with not too much overfitting.

From the confusion matrix in Figure 5.1.3.2, the baseline model correctly classified on the test set 135 out of 286 for class 0 test samples which is the highest among all experiments in this project, with class 1 having correctly being classified 3456 out of 3839 samples for class 1 and 696 from 832 class 2 samples. The AUC score achieved by the baseline model, achieving 0.86, 0.93, and 0.97 for class 0, 1 and 2 accordingly as shown in Figure 5.1.3.3.

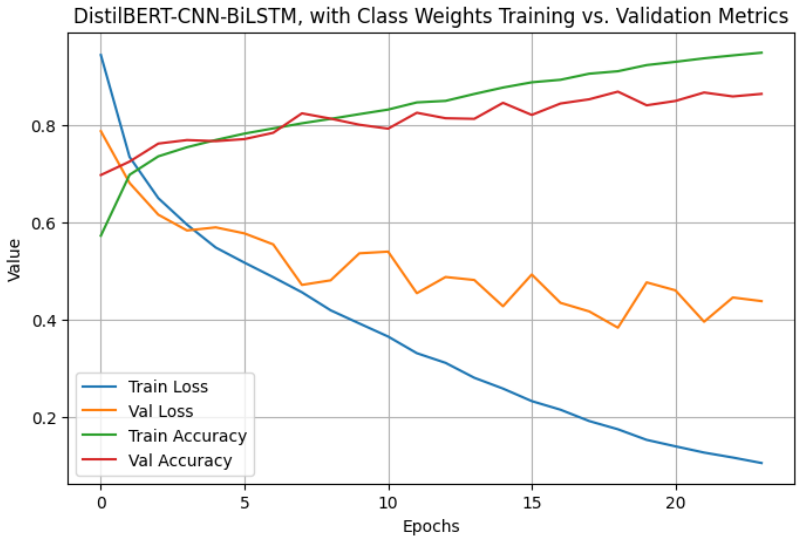


Figure 5.1.3.1 DistilBERT-CNN-BiLSTM with Class Weight Accuracy and Loss Plot for Training and Validation

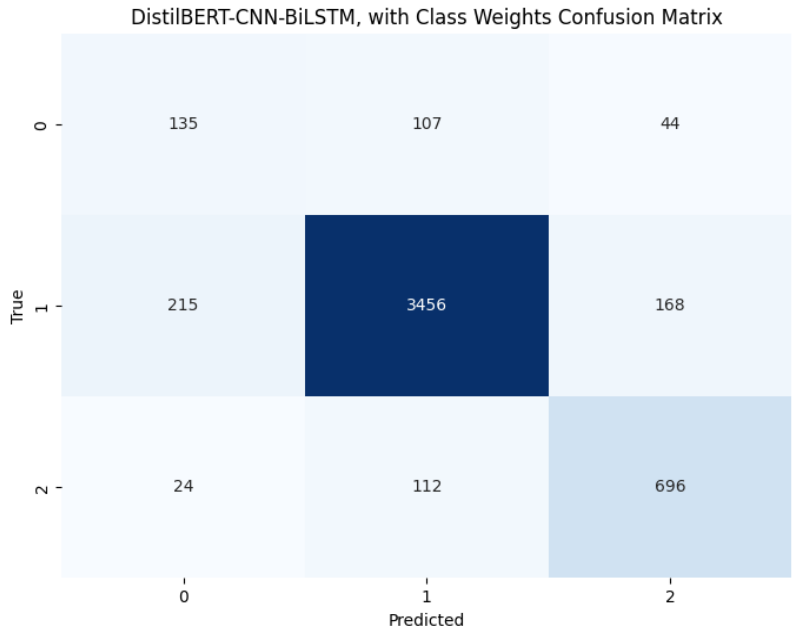


Figure 5.1.3.2 DistilBERT-CNN-BiLSTM with Class Weight Confusion Matrix

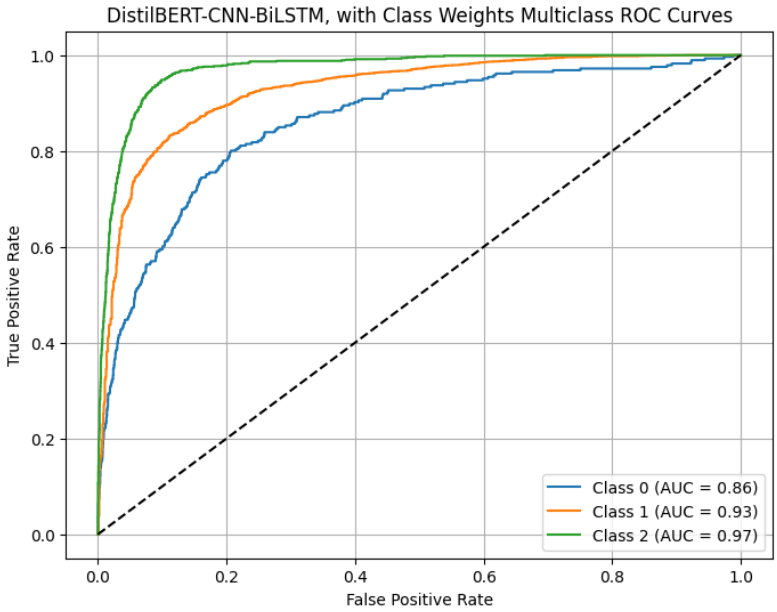


Figure 5.1.3.3 DistilBERT-CNN-BiLSTM with Class Weight Multiclass ROC Curve

5.1.4 Experiment IV: Resampled + Class Weight Model

The fourth experiment combines both resampling and class weighting to examine whether a hybrid approach can achieve more balanced performance across classes. While resampling adjusts the dataset distribution, class weighting simultaneously modifies the learning objective to account for imbalance. The integration of these techniques is designed to exploit their complementary benefits, aiming to maximize recall and F1-score for minority classes without substantially compromising overall accuracy.

Figure 5.1.4.1 shows the training and validation, accuracy and loss result across its multiple epochs training with combination of class weights and resampling. From the diagram, the model has a smooth training accuracy and train loss improvement throughout the iteration training, with its training loss reducing throughout the training routine. The validation routine shows that the model parameter is overfitting when terminating training, as the validation accuracy stabilizes after the fifth epochs with a gap from the training accuracy, indicating the model no longer learns patterns after the fifth epoch. The validation loss fluctuation also showed the model's parameter slowly does not validate well and the training stops.

From the confusion matrix in Figure 5.1.4.2, the baseline model correctly classified on the test set 111 out of 286 for class 0 test samples which is the highest among all experiments in this project, with class 1 having correctly being classified 3547 out of 3839 samples for class 1 and 688 from 832 class 2 samples. The AUC score achieved by the baseline model, achieving 0.85, 0.93, and 0.97 for class 0, 1 and 2 accordingly as shown in Figure 5.1.4.3.

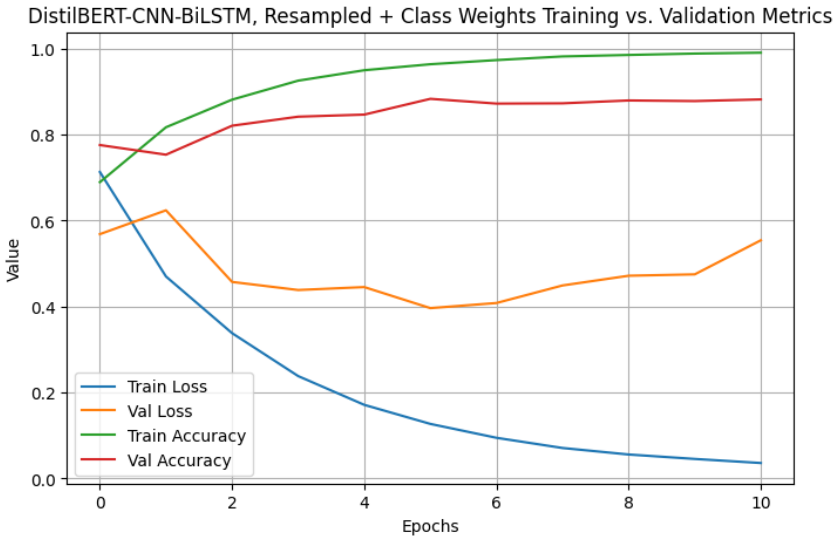


Figure 5.1.4.1 DistilBERT-CNN-BiLSTM, Resampled + Class Weight Accuracy and Loss Plot for Training and Validation

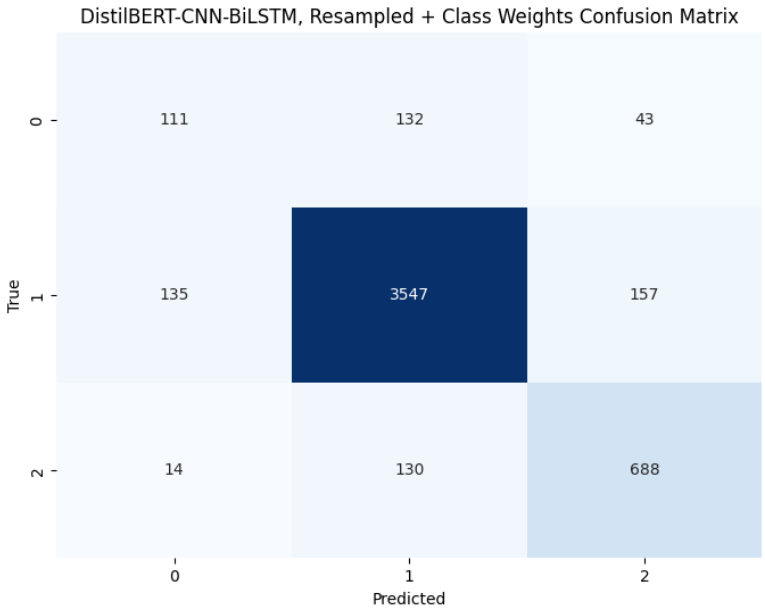


Figure 5.1.4.2 DistilBERT-CNN-BiLSTM, Resampled + Class Weight Confusion Matrix

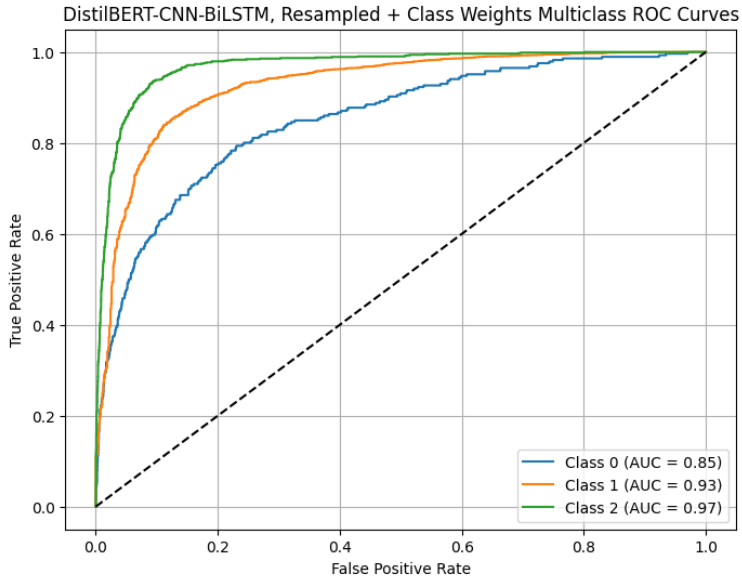


Figure 5.1.4.3 DistilBERT-CNN-BiLSTM, Resampled +Class Weight Multiclass ROC Curve

5.1.5 Experiment V: Ablation Model

The fifth experiment performs an ablation study by removing the preprocessing step to evaluate its importance by training the model on unprocessed data. Text cleaning, lemmatization, and normalization procedures were removed to assess their impact to classification performance. By comparing the ablation results with other experiments, the role of preprocessing were isolated while evaluating the DistilBERT–CNN–BiLSTM model's ability to detect HS.

Figure 5.1.5.1 shows the training and validation, accuracy and loss result across its multiple epochs for the model under ablation study. From the diagram, the model achieved a high accuracy score throughout the iteration of the epochs, with its training loss gradually reducing through the training routine. The validation routine although has little increment in the start it loses performance towards the end. This can be indication that the model was unable to learn from the noises existing in the textual data, as processed data provides better representation of data allowing the model to generalize better.

From the confusion matrix in Figure 5.1.5.2, the baseline model correctly classified on the test set 58 out of 286 for class 0 test samples which is the lowest score among all experiments in this project, with class 1 having correctly being classified 3651 out of 3839 samples for class 1 and 670 from 832 class 2 samples. The AUC score achieved by the baseline model, achieving 0.86, 0.94, and 0.97 for class 0, 1 and 2 accordingly as shown in Figure 5.1.5.3.

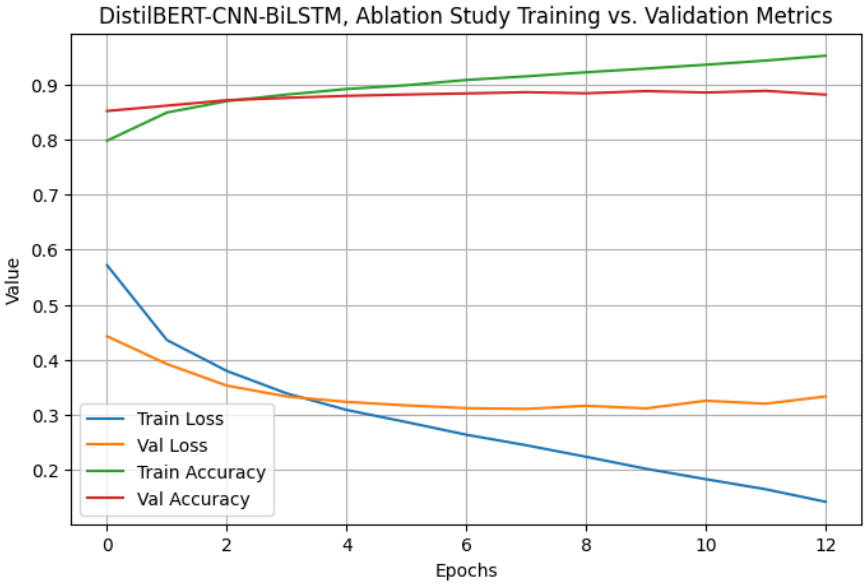


Figure 5.1.5.1 DistilBERT-CNN-BiLSTM, Ablation Study Accuracy and Loss Plot for Training and Validation

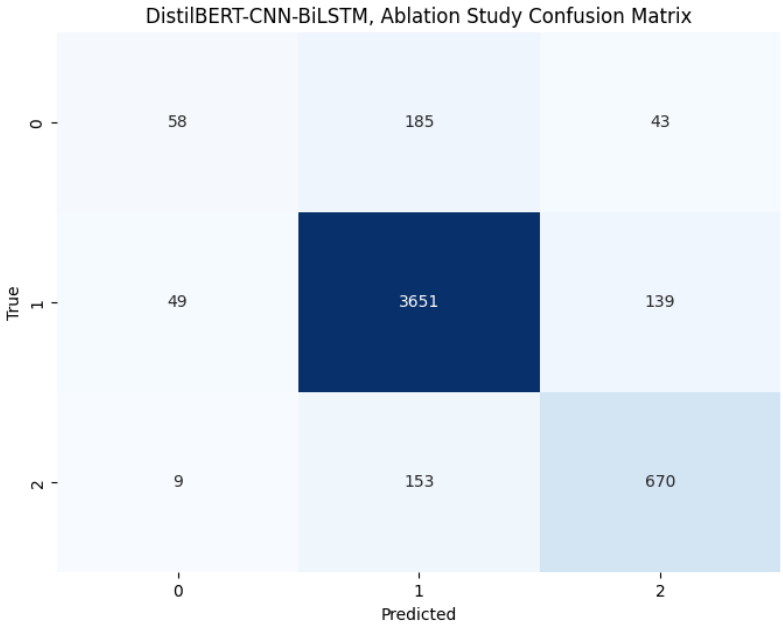


Figure 5.1.5.2 DistilBERT-CNN-BiLSTM, Ablation Study Confusion Matrix

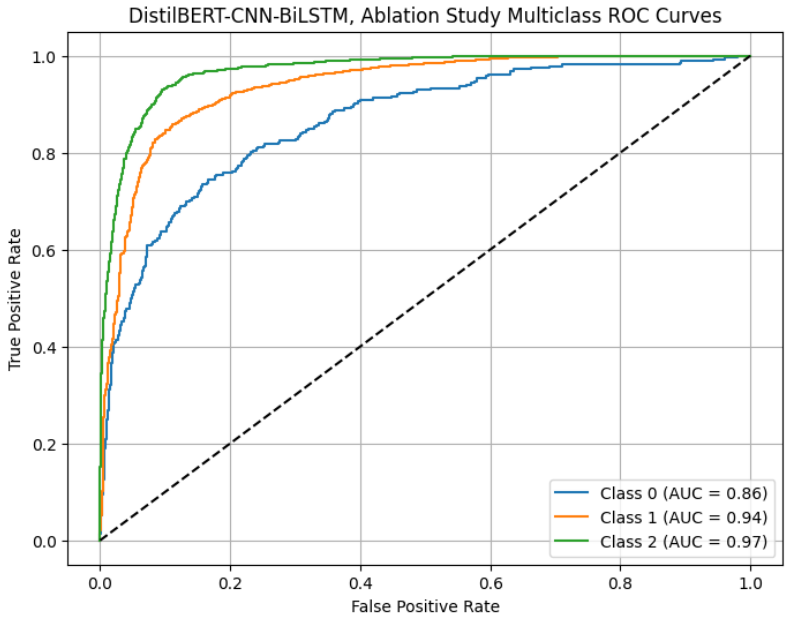


Figure 5.1.5.3 DistilBERT-CNN-BiLSTM, Ablation Study Multiclass ROC Curve

5.1.6 Results Summary

Table 5.1.6.1 summarizes some of the evaluation metrics for all the experiments, namely the F1-score of each classes, accuracy score, Kappa score, and AUC score. The baseline DistilBERT-CNN-BiLSTM model scored the highest in Kappa score and AUC, which are 0.6805 and 0.9260 respectively, showing the model is able to classify correctly most of the data regardless of classes, this however show that the model is bias towards the majority class. With the similar finding found in ablation study which the model has high accuracy score of 0.8834 and bias towards the class 1 with the highest score of F1-score of 0.9328. However the ablation model scored the lowest in classify the lowest frequency class.

The model scored on the underrepresented classes the best when it is paired with class weight, as the model achieved the highest class 0 F1-score. This however has its trade-off of losing overall accuracy, as the class weight model has the lowest accuracy score (0.8648). The most balanced model is the resampled + class weight model, although the model does not has a highest score, however the score obtained are the most harmonized, with high F1-score on the class 0 without losing too much accuracy score. Visualization of the experimental outcomes according to evaluation metrics were illustrated in Figure 5.1.6.1 and Figure 5.1.6.2.

Model	F1 (Class 0)	F1 (Class 1)	F1 (Class 2)	Accuracy	Cohen's Kappa	AUC (Macro)

Base	0.3587	0.9322	0.8036	0.8832	**0.6805**	**0.9260**
Resampled	0.3917	0.9257	**0.8147**	0.8751	0.6706	0.9148
Class Weights	**0.4091**	0.9199	0.8000	0.8648	0.6541	0.9197
Resampled + Class Weights	0.4066	0.9276	0.8000	0.8767	0.6685	0.9159
Ablation Study	0.2886	**0.9328**	0.7957	**0.8834**	0.6636	0.9236

Table 5.1.6.1 Summary of Evaluation Metrics for Five Experiments Implemented

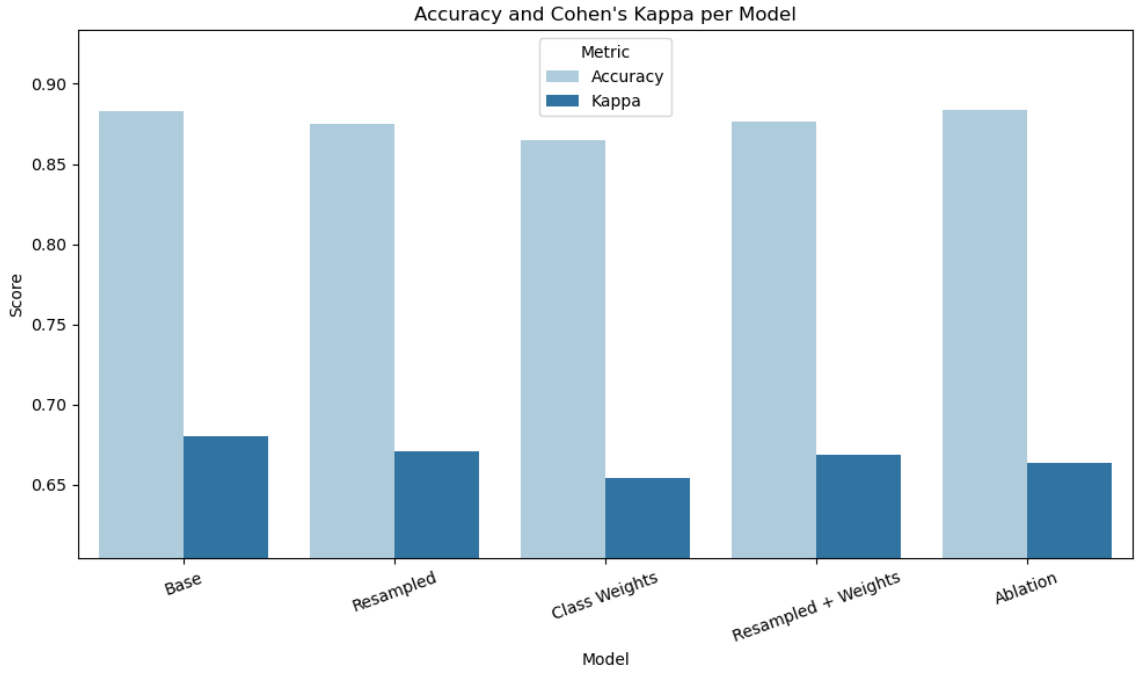


Figure 5.1.6.1 Accuracy and Kappa Score for Five Experiments Implemented

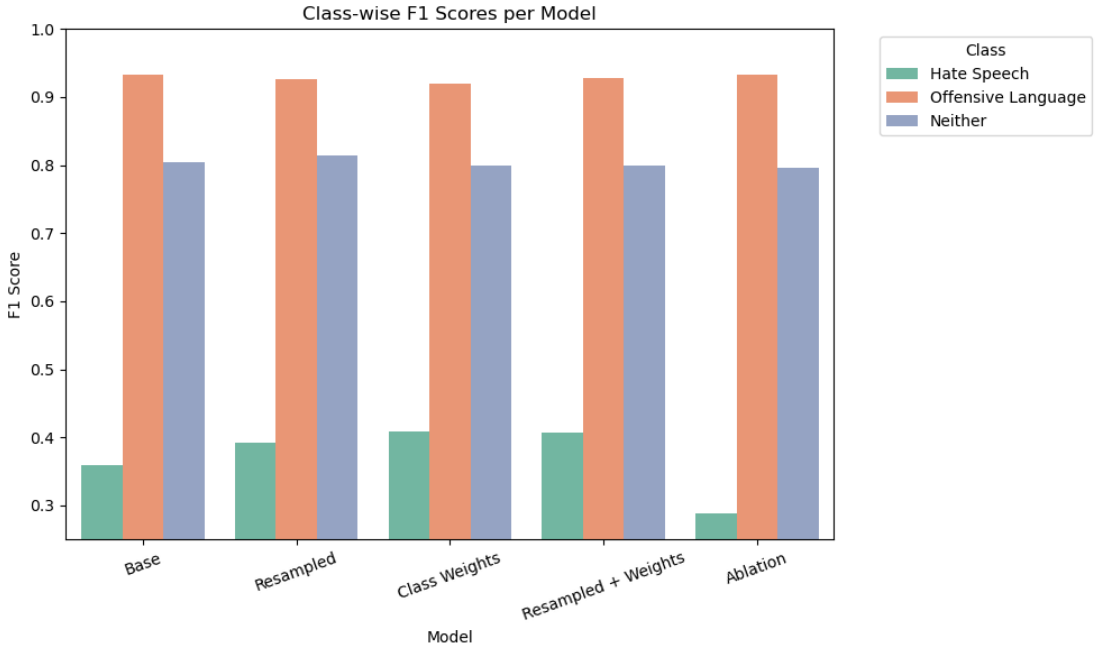


Figure 5.1.6.2 Class-wise F1-score for Five Experiments Implemented

5.2 Comparative Analysis against SOTA

Algorithm	Hate Speech (0)				Offensive (1)			Neither (2)			Macro Avg		
	Acc	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
DistilBERT-CNN-BiLSTM	88%	50%	28%	36%	93%	93%	93%	75%	86%	80%	73%	69%	70%
+ Resampled	88%	39%	40%	39%	94%	92%	93%	78%	85%	81%	70%	72%	71%
+ Class Weights	86%	36%	47%	41%	94%	90%	92%	77%	84%	80%	69%	74%	71%
+ Resampled + Class Weights	88%	43%	39%	41%	93%	92%	93%	77%	83%	80%	71%	71%	71%
Ablation Study	88%	50%	20%	29%	92%	95%	93%	79%	81%	80%	73%	65%	67%
BERT + Advanced CNN [21]	90%	31%	41%	36%	95%	93%	94%	89%	91%	90%	72%	75%	73%

Figure 5.2.0.1 Tabulated Results Performing Comparison between Obtained Result and Existing Study

From the table in Figure 5.2.0.1, it was found that in all classes the BERT + Advanced CNN Architecture from [21] performed relatively better on majority of the isolated label classification score, however, the proposed model in this project perform better in one aspects showing the competitive performance of the DistilBERT-CNN-BiLSTM architecture's robustness in handling the task despite its lower number of parameters [57]. On the class 0 classification which is the minority class, the proposed model performed 5% better than the existing model when configured under the combination of data resampling and class weight training. Two of the experiment configuration in this project share the same result, which are

the Class Weight training model and the Resample + Class Weight training model but the better model is the latter as it score 88% rather than the former's 86%.

5.3 Project Challenges

The implementation of hate speech detection using deep learning methods encountered several challenges, as summarized in Table 5.3.1.1. The first difficulty was class imbalance, where the minority hate speech class was significantly underrepresented. Models favours majority classes which is class 1, reducing performance when detecting underrepresented classes (class 0 and class 2). Such overfitting toward dominant classes are often observed [45] in HS detection, and dataset design and sampling affect reliability of accuracy when class 0 samples are rare [46].

A second challenge was computational cost. Although DistilBERT reduces parameters and memory requirements compared to BERT, fine-tuning still requires GPU support for efficiency. Transformer-based architectures impose heavy resource demands, particularly with larger batch sizes or longer sequences. Recent evaluations of Transformer families confirm that higher model complexity sharply increases computational cost [47].

The third limitation was dataset bias and scale. The Davidson dataset used here is monolingual (English) and platform-specific, which restricts generalizability to other cultural or linguistic contexts. [48] demonstrated that English-only hate speech datasets overrepresent certain geographies, and. [49] further showed that cultural interpretation differences across English-speaking countries affect annotation consistency.

Another challenge was ablation complexity. When experiments were conducted with raw, unprocessed text, performance declined significantly. The likely reason is that noise such as spelling errors, irregular punctuation, and slang increases the tokenization burden and reduces meaningful representation learning. Similar issues were observed in recent dataset evaluations.[50], which highlighted the negative impact of noise on minority class classification.

Finally, linking challenges to solutions, the study confirmed that resampling and class weighting improved recall for the minority class, albeit with minor reductions in overall accuracy. These results align with [46] [51], demonstrating that data resampling and balancing enhance hate speech classification outcomes.

Challenge	Impact on Model
Class imbalance	Poor minority class detection; bias toward majority
Computational cost	GPU dependency despite using DistilBERT
Dataset bias & scale	Limited generalizability to other languages/cultures
Ablation (raw data)	Performance drop due to noise and lack of preprocessing
Imbalance mitigation	Resampling/weights improved recall, but trade-offs in accuracy

Table 5.3.1.1 Summary of Project Challenges

5.4 Objectives Evaluation

The evaluation of the research objectives is presented in this section, mapping the experimental findings to the predefined aims of the study. Each objective is systematically assessed with respect to its achievement and contribution to the overall project outcomes.

The primary objective of this work was to design and implement a hybrid architecture that integrates DistilBERT with convolutional neural networks (CNNs) and bidirectional long short-term memory (BiLSTM) units. This objective was fully achieved, as the hybrid model successfully combined contextual embeddings with sequential and convolutional feature extraction. The integration resulted in strong performance across all experimental settings, thereby validating the effectiveness of the architectural design.

A secondary objective was to mitigate the challenges posed by class imbalance in the dataset. This was achieved through the application of both resampling and class weighting techniques. The resampled model improved the minority class F1-score to 0.3917, while the class weighting approach further enhanced this metric to 0.4091. When combined, the hybrid resampling and class weighting model provided more balanced outcomes across metrics, confirming partial success in addressing imbalance.

The study set explicit performance goals, particularly achieving robust macro-level metrics such as F1-score and area under the curve (AUC). These targets were met, with the baseline hybrid model achieving a macro AUC of 0.9260 and an overall accuracy of 88.32%. Moreover, in multiple experiments the macro AUC exceeded 0.91, surpassing baseline expectations and demonstrating the capability of the hybrid model to effectively detect hate speech.

To evaluate the significance of preprocessing, an ablation study was conducted using raw, unprocessed data. While the accuracy (88.34%) remained comparable to the baseline, the minority class F1-score dropped significantly to 0.2886. This confirmed the importance of preprocessing steps such as cleaning and tokenization in ensuring fair and accurate classification. Thus, this objective was successfully addressed.

Another objective was to ensure the reproducibility of the experimental pipeline. This was achieved through programmatic implementation of the entire workflow, enabling automated dataset preparation, model training, and evaluation under multiple experimental configurations. The reproducibility of the code ensures that future work can replicate and extend this research without ambiguity.

Overall, the project successfully met its core objectives. The hybrid DistilBERT–CNN–BiLSTM model exceeded performance baselines, demonstrated robustness across imbalance-handling strategies, and highlighted the necessity of preprocessing through ablation. With macro AUC consistently above 0.91 and minority class F1-scores surpassing 0.40 in resampling-weighted experiments, the outcomes provide strong empirical evidence of the model's effectiveness.

5.5 Concluding Remark

The experimental study confirms that the proposed hybrid DistilBERT–CNN–BiLSTM architecture is effective for hate speech detection in social media contexts. By combining the semantic representation power of DistilBERT with the local feature extraction of convolutional layers and the sequential modelling capability of BiLSTM, the system demonstrated superior performance across multiple experimental configurations. In particular, the hybrid design yielded an overall accuracy above 88% and macro AUC scores exceeding 0.91, which surpass the baseline performance reported in related literature. These findings validate the central premise of this research: that hybrid architectures can enhance both accuracy and robustness in text classification tasks involving nuanced and imbalanced data.

Despite these strengths, several limitations were identified. The model is computationally demanding, which constrains its scalability for deployment in real-time or resource-constrained environments. Additionally, while class imbalance mitigation techniques such as resampling and class weighting improved minority class recognition, the F1-scores for these categories

remain relatively lower compared to the majority class. This highlights the persistent challenge of ensuring equitable classification performance across all classes in imbalanced datasets.

In conclusion, the system developed in this study provides a strong foundation for automated hate speech detection, balancing high performance with reproducible implementation. Its effectiveness demonstrates the potential of hybrid deep learning approaches in tackling real-world natural language processing challenges. However, addressing the identified limitations—particularly scalability and fairness across minority classes—remains an important direction for future research. These aspects are further elaborated in Chapter 7, where recommendations for model optimization, dataset enrichment, and practical deployment strategies are discussed.

CHAPTER 6 CONCLUSION AND RECOMMENDATION

6.1 Summary of Findings

This project examined the application of DL models for hate speech detection using the Davidson dataset, employing DistilBERT as the underlying architecture. DistilBERT was selected due to its significantly reduced parameter count in contrast to the base BERT model, lowering computational cost throughout experimentation and memory requirements while preserving its competitive edge in performance. Multiple experimental settings were evaluated, including a baseline model, resampling, class weighting, and ablation on raw data. Across these configurations, the models consistently achieved strong classification performance, with macro AUC values exceeding 0.91. The baseline model attained an overall accuracy of 88.32%, a Cohen's Kappa of 0.6805, and a macro AUC of 0.9260, thereby confirming the effectiveness of both the preprocessing pipeline and the model design.

The investigation of class imbalance mitigation techniques produced mixed outcomes. Resampling improved the F1-score of the minority class (Class 0) to 0.3917, while sustaining competitive accuracy (87.51%) and macro AUC (0.9148). Class weighting achieved the highest minority class F1-score (0.4091), albeit with a reduction in overall accuracy (86.48%). The combined approach of resampling and class weighting yielded balanced performance across metrics, maintaining F1-scores above 0.40 for Class 0 and achieving a macro AUC of 0.9159. The ablation study, which excluded preprocessing, further underscored the critical role of data cleaning and tokenization. Although overall accuracy remained comparable (88.34%), the minority class F1-score declined substantially to 0.2886, highlighting the necessity of preprocessing for equitable performance across classes.

In summary, the findings demonstrate the inherent trade-offs between enhancing minority class detection and preserving overall accuracy. More broadly, the results confirm that fine-tuned Transformer-based models, even in lightweight variants such as DistilBERT, are effective for hate speech detection while offering the advantage of reduced computational overhead and lower resource demands.

6.2 Contributions

This study contributes to the domain of automated hate speech detection in several significant ways. First, a DL model utilizing DistilBERT architecture was developed and fine-tuned, and its effectiveness was validated using the Davidson dataset. Second, the work provides a comparative evaluation of class imbalance handling strategies, specifically resampling, class weighting, and their combination, thereby offering insights into their relative impact on minority class detection. Third, a systematic ablation study was conducted to assess the contribution of preprocessing, demonstrating its critical role in enhancing performance, particularly for underrepresented classes. Fourth, comprehensive benchmarking was performed, reporting detailed metrics including F1-scores, accuracy, Cohen's Kappa, and macro AUC, thus establishing reproducible reference points for future research. Finally, the study delivers an implementation framework utilizing Google Colab, TensorFlow, and the Hugging Face Transformers library, enabling replicability and extension to broader datasets and domains.

6.3 Recommendation

While this project successfully implemented and evaluated HS detection models, several avenues remain open for future investigation. First, the application of advanced resampling strategies beyond traditional oversampling methods should be considered. In particular, the use of data synthesis techniques supported by state-of-the-art large language models (LLMs) may generate realistic and diverse training samples, thereby improving representation of minority classes. Second, future work may extend the framework toward multimodal hate speech detection by integrating textual, visual, and auditory signals, since harmful content is often expressed through images, memes, or speech in addition to text. Third, research into real-time applications is warranted, with emphasis on deploying models in streaming environments such as social media platforms, where inference speed and scalability are critical. Fourth, addressing dataset bias requires the incorporation of multilingual corpora, enabling models to operate effectively across different languages, domains, and cultural contexts. Finally, further work should focus on robustness and fairness, particularly with respect to adversarial resistance and equitable performance across demographic subgroups, in order to mitigate risks of bias amplification and to ensure reliable generalization.

REFERENCES

[1] M. A. Paz, J. Montero-Diaz, and A. Moreno-Delgado, “Hate Speech: A Systematized Review”, *Sage Journals*, Nov. 2020.

[2] J. W. Howard, “Free Speech and Hate Speech”, *Annual Review of Political Science*, vol. 22, May 2019.

[3] S. Wachs, M. Gamez-Guadix, and M. F. Wright, “Online Hate Speech Victimization and Depressive Symptoms Among Adolescents: The Protective Role of Resilience”, *Cyberpsychology, Behavior, and Social Networking,* vol. 25, no. 7, 2022, doi: 10.1089/cyber.2022.0009

[4] C. Abderrouaf and M. Oussalah, "On Online Hate Speech Detection. Effects of Negated Data Construction", *2019 IEEE International Conference on Big Data (Big Data)*, pp. 5595-5602, 2019

[5] Z. Yue et al., “Social media use, perceived social support, and well-being: Evidence from two waves of surveys peri- and post-COVID-19 lockdown”, *Journal of Social and Personal Relationships*, vol. 0, pp 1-19, 2023, doi: 10.1177/02654075231188185

[6] L. Al-Henaki et al., “MultiProSE: A Multi-label Arabic Dataset for Propaganda, Sentiment, and Emotion Detection”, arXiv, 2025, doi: https://doi.org/10.48550/arXiv.2502.08319

[7] S. Gupta, A. Singh, and V. Kumar, “Emoji, Text, and Sentiment Polarity Detection Using Natural Language Processing”, Advances in Machine Learning and Intelligent Information Systems, vol. 14, 2023, doi: https://doi.org/10.3390/info14040222

[8] E. F. Ayetiran, O. Ozgobek, “An inter-modal attention-based deep learning framework using unified modality for multimodal fake news, hate speech and offensive language detection”, *Information Systems*, vol. 123, 2023, doi: https://doi.org/10.1016/j.is.2024.102378

[9] S. Abro et al., “Automatic Hate Speech Detection using Machine Learning: A Comparative Study”, *(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 11, no. 8, pp. 484-490, 2020.

[10] W. S. Noble, “What is a support vector machine?”, Nature Biotechnology, vol. 24, no. 12, 2006.

[11] A. M. U. D. Khanday et al., “Detecting twitter hate speech in COVID-19 era using machine learning and ensemble learning techniques”, International Journal of

Information Management Data Insights, vol. 2, Nov. 2022, doi: https://doi.org/10.1016/j.jjimei.2022.100120

[12] A. Chiu et al., "Detecting Hate Speech on Social Media with Respect to Adolescent Vulnerability", 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC), Apr. 2023, doi: 10.1109/CCWC57344.2023.10099373

[13] A. Kumar, H. Mittal, A. Kumar, "Developing and Evaluating a Binary Hate Speech Dataset: A Comparitive Study of Machine Learning Models", 2023 5th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Jun. 2024, doi: 10.1109/ICAC3N60023.2023.10541516

[14] N. S. Mullah, and W. Nazmee, "Advances in Machine Learning Algorithms for Hate Speech Detection in Social Media: A Review", IEEE Access, vol. 9, doi: 10.1109/ACCESS.2021.3089515

[15] S. Khan et al., "BiCHAT: BiLSTM with deep CNN and hierarchical attention for hate speech detection", Journal of King Saud University - Computer and Information Sciences, vol. 34, pp. 4335-4344, Jul. 2022, doi: https://doi.org/10.1016/j.jksuci.2022.05.006

[16] S. Khan, A. Kamal, and M. Fazil, "HCovBi-Caps: Hate Speech Detection Using Convolutional and Bi-Directional Gated Recurrent Unit With Capsule Network", IEEE Access, vol. 10, Jan. 2022, Doi: 10.1109/ACCESS.2022.3143799

[17] A. Founta et al., "Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior", Proceedings of the Twelfth International AAAI Conference on Web and Social Media (ICWSM 2018), Feb. 2018, doi: https://doi.org/10.48550/arXiv.1802.00393

[18] Malik, J.S., Qiao, H., Pang, G. *et al.* Deep learning for hate speech detection: a comparative study. *Int J Data Sci Anal* (2024). https://doi.org/10.1007/s41060-024-00650-6

[19] A. Joulin et al., "FastText.zip: Compressing Text Classification Models", ICLR 2017, Nov. 2016, doi: arXiv:1612.03651v1

[20] P. K. Jayapal, K. R. D. Ramachandraiah, and K. K. Lella, "Racism and hate speech detection on Twitter: A QAHA-based hybrid deep learning approach using LSTM-CNN," Int J. Knowl. Innov Stud., vol. 1, no. 2, pp. 89–102, 2023. https://doi.org/10.56578/ijkis010202.

[21] C. D. Putra and H. C. Wang, “Advanced BERT-CNN for Hate Speech Detection”, Procedia Computer Science

[22] P. Juyal, “Deep Learning Approaches for Effective Hate Speech Classification”, 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT), Greater Noida, India, 2024.

[23] P. Sharma, and R. K. Tiwari, “Deep Learning Approach for Hate and Non Hate Speech Detection in Online Social Media”, 2023 3rd International Conference on Technological Advancements in Computational Sciences (ICTACS), Tashkent, Uzbekistan, 2024.

[24] A. Chhabra, and D. K. Vishwakarma, “Fuzzy and Machine learning Classifiers for Hate Content Detection: A Comparative Analysis”, 2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST), Delhi, India, 2023.

[25] B. Garg, A. Bhardwaj, and T. Jain, “Detection of Hate Speech and Offensive Language Using Machine Learning and Deep Learning for Multi-Class Tweets”, 2024 IEEE International Conference on Smart Power Control and Renewable Energy (ICSPCRE), Rourkela, India, 2024.

[26] E. Hashmi, and S. Y. Yayilgan, “Multi-class hate speech detection in the Norwegian language using FAST-RNN and multilingual fine-tunedtransformers”, Complex & Intelligent Systems, vol. 10, pp. 4535-4556, Mar. 2024.

[27] A. Krishnan et al., “MalHate : Hate Speech Detection in Malayalam Regional Language”, 2022 IEEE 7th International Conference on Recent Advances and Innovations in Engineering (ICRAIE), Mangalore, India.

[28] P. D. Varma et al., “Hate Speech detection in English and Malayalam Code-Mixed Text using BERT embedding”, 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), Kochi, India.

[29] V. Anand et al., “Enhancing Hate Speech Detection in Tamil Code-Mix Content: A Deep Learning Approach with Multilingual Embedding”, 2024 5th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India.

[30] Chowdhury, G. (2003) Natural language processing. Annual Review of Information Science and Technology, 37. pp. 51-89. ISSN 0066-4200

[31] B. Ganesh and J. Bright, “Countering Extremists on Social Media: Challenges for Strategic Communication and Content Moderation”, Policy Studies Organization, 2020, doi: 10.1002/poi3.236

[32] A. Tontodimamma et al., “Thirty years of research into hate speech: topics of interest and their evolution”, Scientometrics, Oct. 2020, doi: https://doi.org/10.1007/s11192-020-03737-6

[33] S. MacAvaney et al., “Hate speech detection: Challenges and solutions”, PLoS ONE 14, Jul. 2019, doi: https://doi.org/10.1371/journal.pone.0221152

[34] S. Susan and A. Kumar, “The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent State of the Art”, Wiley, Nov. 2019. doi: 10.1002/eng2.12298

[35] Z. Reitermanova, “Data Splitting”, WDS'10 Proceedings of Contributed Papers, Part 1, pp. 31–36, 2010.

[36] V. Mohan, “Text Mining: Open Source Tokenization Tools: An Analysis”, Advanced Computational Intelligence: An International Journal (ACII), Vol.3, No.1, Jan. 2016.

[37]

[38] A. Cano, A. Zafra, and S. Ventura, “Weighted Data Gravitation Classification for Standard and Imbalanced Data”, IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics: a publication of the IEEE Systems, Man, and Cybernetics Society, Jan. 2023, doi: 10.1109/TSMCB.2012.2227470

[39] L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions”, Journal of Big Data, 2021, doi: https://doi.org/10.1186/s40537-021-00444-8

[40] J. Xiao and Z. Zhou, “Research Progress of RNN Language Model”, 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China.

[41] W. Yin et al., “Comparative Study of CNN and RNN for Natural Language Processing”, arXiv Computation and Language, Feb 2017, doi: https://doi.org/10.48550/arXiv.1702.01923

[42]

[43] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, arXiv Machine Learning, Dec. 2014. doi: https://doi.org/10.48550/arXiv.1412.6980

[44] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proc. 2019 Conf. North American Chapter Assoc. for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, Minneapolis, MN, USA, Jun. 2019, pp. 4171–4186.

[45] J. Yoo, J. Lee, and H. Ko, “Adaptive ensemble techniques leveraging BERT-based models for imbalanced hate speech detection,” *Scientific Reports*, vol. 15, no. 1, pp. 1–12, 2025.

[46] S. Fetahi, Y. Liu, and L. Schmidt, “Balancing the imbalance: Resampling strategies for hate speech detection on social media,” *Social Network Analysis and Mining*, vol. 15, no. 3, pp. 1–14, 2025.

[47] R. Arora, S. Verma, and P. Kumar, “Advancing hate speech detection with transformers: An efficiency vs. accuracy trade-off,” *arXiv preprint*, arXiv:2508.04913, Aug. 2025.

[48] F. Tonneau, A. Møller, and L. Noiry, “From languages to geographies: Towards evaluating cultural bias in hate speech datasets,” *arXiv preprint*, arXiv:2404.17874, Apr. 2024.

[49] S. Lee, R. Röttger, and I. Augenstein, “Exploring cross-cultural differences in English hate speech annotations,” *arXiv preprint*, arXiv:2308.16705, Aug. 2023.

[50] T. Nguyen and M. Sinha, “Noise, imbalance, and bias: Dataset challenges in hate speech detection revisited,” *arXiv preprint*, arXiv:2505.14272, May 2025.

[51] B. Bakırarar and A. H. Elhan, “Class weighting technique to deal with imbalanced class problem in machine learning: Methodological research,” *Turkiye Klinikleri Journal of Biostatistics*, Nov. 2023.

[52] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," arXiv preprint arXiv:1910.01108, 2019. [Online]. Available: https://arxiv.org/abs/1910.01108

[53] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357.

[54] W. Chen, K. Yang, Z. Yu, Y. Shi, C. L. P. Chen et al., “A survey on imbalanced learning: latest research, applications and future directions,” *Artificial Intelligence Review*, Vol. 57, Article Number 137, May 2024.

[55] E. Jain, J. Neeraja, B. Banerjee, and P. Ghosh, “A Diagnostic Approach to Assess the Quality of Data Splitting in Machine Learning,” *arXiv preprint arXiv:2206.11721*, Jun. 2022

[56] B. Bakirarar and A. H. Elhan, “Class Weighting Technique to Deal with Imbalanced Class Problem in Machine Learning: Methodological Research,” *Turkiye Klinikleri*

Journal of Biostatistics, Nov. 2023.], [A. Cano, A. Zafra, and S. Ventura, “Weighted Data Gravitation Classification for Standard and Imbalanced Data,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Jan. 2023, doi: 10.1109/TSMCB.2012.2227470

[57] B. Akdik and G. Sariman, “Hate Speech Detection in Social Media with Deep Learning and Language Models,” *Bitlis Eren Üniversitesi Fen Bilimleri Dergisi*, vol. 14, no. 2, pp. 1077–1095, 2025, doi: 10.17798/bitlisfen.1637822.

[58] Schuster, M. and Paliwal, K. K., “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[59] L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *Journal of Big Data*, 2021

[60] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R., “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014

[61] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016

[62] C. Gao, "Prediction performance analysis for ML models based on impacts of data imbalance and bias," *Proceedings of the 2024 ACM Southeast Conference*, 2024, pp. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/3603287.3651191

[63] Y. Zhao, "ABGEN: Evaluating large language models in ablation study design and evaluation for scientific research," *Proc. ACL*, 2025, pp. 1–10. [Online]. Available: https://aclanthology.org/2025.acl-long.611.pdf

APPENDIX A: EXPERIMENT SCRIPT

#pipeline.py

```
import pandas as pd
import numpy as np
import re, html, gc, torch
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
import spacy
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras import layers
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Dense, Dropout, LSTM, Conv1D,
GlobalMaxPool1D, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping

try:
    from tensorflow.keras.optimizers import AdamW
except ImportError:
    from tensorflow.keras.optimizers.experimental import AdamW
from tensorflow.keras.optimizers import Adam

from transformers import TFDistilBertModel, DistilBertTokenizer
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score,
roc_auc_score, roc_curve, auc
from sklearn.preprocessing import label_binarize
import keras_tuner as kt
from imblearn.over_sampling import RandomOverSampler

nltk.download('stopwords')
nlp = spacy.load("en_core_web_sm")
stop_words = set(stopwords.words("english"))
stop_words.update({"rt", "#ff", "ff"})

def preprocess(tweets):
    if not isinstance(tweets, pd.Series):
        tweets = pd.Series(tweets)
    tweets = tweets.str.replace(r'\bRT\b', '', regex=True)
    tweets = tweets.str.replace(r'@[\w\-]+', '', regex=True)
    tweets = tweets.str.replace(r'http\S+|www\.\S+', '', regex=True)
    tweets = tweets.str.replace(r'&\w+;', '', regex=True)
    tweets = tweets.str.replace(r'([!?])\1+', r'\1', regex=True)
    tweets = tweets.str.replace(r'^[\'"!:]+', '', regex=True)
```

```
    tweets = tweets.str.replace(r'\s+', ' ', regex=True).str.strip()
    def lemmatize_text(text):
        doc = nlp(text.lower())
        return ' '.join([token.lemma_ for token in doc if token.text not in stop_words and
token.is_alpha])
    tweets = tweets.apply(lemmatize_text)
    return tweets

def plot_training_history(history, model_name="Model"):
    plt.figure(figsize=(8, 5))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    if 'accuracy' in history.history:
        plt.plot(history.history['accuracy'], label='Train Accuracy')
        plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.legend()
    plt.title(f"{model_name} Training vs. Validation Metrics")
    plt.xlabel("Epochs")
    plt.ylabel("Value")
    plt.grid(True)
    plt.show()

def get_df():
    url = "https://raw.githubusercontent.com/NakulLakhotia/Hate-Speech-Detection-in-Social-
Media-using-Python/refs/heads/master/HateSpeechData.csv"
    df = pd.read_csv(url)
    return df

def get_processed_df(df):
    tweet = df['tweet']
    processed_tweets = preprocess(tweet)
    df['processed_tweets'] = processed_tweets
    print(df[['tweet', 'processed_tweets']].head(10))
    return df

def get_train_test_data(df):
    X = df['processed_tweets'].astype(str).tolist()
    y = df['class'].values
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42,
stratify=y)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42, stratify=y_temp)
    return X_train, X_val, X_test, y_train, y_val, y_test

def get_resampled(X_train, y_train):
    ros = RandomOverSampler(random_state=42)
    X_train_resampled, y_train_resampled = ros.fit_resample(pd.DataFrame(X_train), y_train)
    X_train_resampled = X_train_resampled[0].values
```

```
    visualize_resampled_df = pd.DataFrame({"text": X_train_resampled, "class":
y_train_resampled})
    plt.figure(figsize=(8, 6))
    sns.countplot(x="class", data=visualize_resampled_df)
    plt.title("Distribution of Classes After Resampling")
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.show()
    print(visualize_resampled_df["class"].value_counts())
    return X_train_resampled, y_train_resampled

def get_class_weights(y_train):
    class_weights = compute_class_weight(class_weight='balanced',
classes=np.unique(y_train), y=y_train)
    class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}
    print("Class Weights:", class_weights_dict)
    return class_weights_dict

def get_ablation_study_data(df):
    X_ablation = df['tweet'].values
    y_ablation = df['class'].values
    X_train_ablation, X_temp_ablation, y_train_ablation, y_temp_ablation = train_test_split(
        X_ablation, y_ablation, test_size=0.4, random_state=42, stratify=y_ablation)
    X_val_ablation, X_test_ablation, y_val_ablation, y_test_ablation = train_test_split(
        X_temp_ablation, y_temp_ablation, test_size=0.5, random_state=42,
stratify=y_temp_ablation)
    return X_train_ablation, X_val_ablation, X_test_ablation, y_train_ablation,
y_val_ablation, y_test_ablation

def get_model_tokenizer():
    class TFDistilBertLayer(layers.Layer):
        def __init__(self, model_name="distilbert-base-uncased", **kwargs):
            super(TFDistilBertLayer, self).__init__(**kwargs)
            self.bert = TFDistilBertModel.from_pretrained(model_name, from_pt=True)
        def call(self, inputs):
            input_ids, attention_mask = inputs
            outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
            return outputs.last_hidden_state
        def get_config(self):
            config = super().get_config()
            config.update({"model_name": "distilbert-base-uncased"})
            return config
    tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
    input_ids = tf.keras.Input(shape=(52,), dtype=tf.int32, name="input_ids")
    attention_mask = tf.keras.Input(shape=(52,), dtype=tf.int32, name="attention_mask")
    sequence_output = TFDistilBertLayer()(inputs=[input_ids, attention_mask])
    conv3 = GlobalMaxPool1D()(Conv1D(64, kernel_size=3, activation="relu",
padding="same")(sequence_output))
```

```
    conv5 = GlobalMaxPool1D()(Conv1D(64, kernel_size=5, activation="relu",
padding="same")(sequence_output))
    conv7 = GlobalMaxPool1D()(Conv1D(64, kernel_size=7, activation="relu",
padding="same")(sequence_output))
    cnn_out = tf.keras.layers.Concatenate()([conv3, conv5, conv7])
    lstm_out = Bidirectional(LSTM(64, return_sequences=False))(sequence_output)
    merged = tf.keras.layers.Concatenate()([cnn_out, lstm_out])
    dense_out = Dense(256, activation="relu")(merged)
    dense_out = Dropout(0.3)(dense_out)
    dense_out = Dense(128, activation="relu")(dense_out)
    dense_out = Dropout(0.3)(dense_out)
    output = Dense(3, activation="softmax")(dense_out)
    distilBERT_CNN_BiLSTM = Model(inputs=[input_ids, attention_mask], outputs=output)

distilBERT_CNN_BiLSTM.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-
5),
                              loss="sparse_categorical_crossentropy",
                              metrics=["accuracy"])
    distilBERT_CNN_BiLSTM.summary()
    return distilBERT_CNN_BiLSTM, tokenizer

def tokenize_data(texts, tokenizer, max_length=52):
    if not isinstance(texts, list):
        texts = texts.tolist()
    encoded = tokenizer(texts, padding="max_length", truncation=True,
max_length=max_length, return_tensors="tf")
    return encoded["input_ids"], encoded["attention_mask"]

def train_model(model, X_train, y_train, X_val, y_val, tokenizer, epochs, batch_size,
class_weights=None):
    train_input_ids, train_attention_mask = tokenize_data(X_train, tokenizer)
    val_input_ids, val_attention_mask = tokenize_data(X_val, tokenizer)
    early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
    history = model.fit([train_input_ids, train_attention_mask], y_train,
                        validation_data=([val_input_ids, val_attention_mask], y_val),
                        epochs=epochs, batch_size=batch_size, class_weight=class_weights,
                        callbacks=[early_stopping])
    return history

def evaluate_model(model, X_test, y_test, tokenizer, max_length=52, model_name="Model",
plot_cm=True):
    X_test_cleaned = [html.unescape(text) for text in X_test]
    test_input_ids, test_attention_mask = tokenize_data(X_test_cleaned, tokenizer,
max_length)
    print(f"Input IDs shape: {test_input_ids.shape}")
    print(f"Attention Mask shape: {test_attention_mask.shape}")
    y_test = np.array(y_test, dtype=np.int32)
    loss, accuracy = model.evaluate([test_input_ids, test_attention_mask], y_test, verbose=0)
```

```
    print(f"\n{model_name} Evaluation:")
    print(f"Test Loss: {loss:.4f}")
    print(f"Test Accuracy: {accuracy:.4f}")
    predictions = model.predict([test_input_ids, test_attention_mask], verbose=0)
    predicted_classes = np.argmax(predictions, axis=1)
    class_report = classification_report(y_test, predicted_classes, digits=4)
    print("\nClassification Report:")
    print(class_report)
    kappa = cohen_kappa_score(y_test, predicted_classes)
    print(f"Cohen's Kappa Score: {kappa:.4f}")
    cm = confusion_matrix(y_test, predicted_classes)
    print("\nConfusion Matrix:")
    print(cm)
    if plot_cm:
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
                    xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
        plt.title(f"{model_name} Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")
        plt.show()
    return {"loss": loss, "accuracy": accuracy, "classification_report": class_report,
"kappa_score": kappa, "confusion_matrix": cm}

def plot_multiclass_roc(model, X_val, y_val, tokenizer, classes=[0, 1, 2], max_length=52,
model_name="Model"):
    X_val_ids, X_val_mask = tokenize_data(X_val, tokenizer, max_length=max_length)
    y_pred_probs = model.predict([X_val_ids, X_val_mask], verbose=0)
    y_val_bin = label_binarize(y_val, classes=classes)
    auc_score = roc_auc_score(y_val_bin, y_pred_probs, average="macro",
multi_class="ovr")
    print(f"{model_name} Testing AUC (macro): {auc_score:.4f}")
    plt.figure(figsize=(8,6))
    for i, cls in enumerate(classes):
        fpr, tpr, _ = roc_curve(y_val_bin[:, i], y_pred_probs[:, i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f"Class {cls} (AUC = {roc_auc:.2f})")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel
def clear_cache():
    K.clear_session()
    gc.collect()
    try:
        torch.cuda.empty_cache()
    except:
        pass

def run_experiment(df, exp_name, get_data_fn, resample=False, use_class_weights=False,
epochs=30, batch_size=16):
```

```
    print(f"\n{'='*20} {exp_name} {'='*20}")
    X_train, X_val, X_test, y_train, y_val, y_test = get_data_fn(df)
    if resample:
        X_train, y_train = get_resampled(X_train, y_train)
    class_weights = None
    if use_class_weights:
        class_weights = get_class_weights(y_train)
    model, tokenizer = get_model_tokenizer()
    history = train_model(
        model, X_train, y_train,
        X_val, y_val,
        tokenizer,
        epochs=epochs,
        batch_size=batch_size,
        class_weights=class_weights
    )
    plot_training_history(history, model_name=exp_name)
    results = evaluate_model(model, X_test, y_test, tokenizer, model_name=exp_name)
    auc_score = plot_multiclass_roc(model, X_val, y_val, tokenizer, model_name=exp_name)
    clear_cache()
    return {"results": results, "auc_score": auc_score}

def collect_results(all_results):
    records = []
    for exp_name, exp_data in all_results.items():
        results = exp_data["results"]
        auc_score = exp_data["auc_score"]
        records.append({
            "Experiment": exp_name,
            "Accuracy": results.get("accuracy", None),
            "Kappa": results.get("kappa", None),
            "AUC": auc_score
        })
    return pd.DataFrame(records
```

APPENDIX B: COLAB PROGRAM

```
!pip install transformers
!pip install -q keras-tuner

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import json
import pipeline

df = pd.read_csv('processed_tweets.csv')
all_results = {}

all_results["Experiment 1"] = pipeline.run_experiment(
    df,
    exp_name="DistilBERT-CNN-BiLSTM",
    get_data_fn=pipeline.get_train_test_data
)

all_results["Experiment 2"] = pipeline.run_experiment(
    df,
    exp_name="DistilBERT-CNN-BiLSTM, Resampled",
    get_data_fn=pipeline.get_train_test_data,
    resample=True
)

all_results["Experiment 3"] = pipeline.run_experiment(
    df,
    exp_name="DistilBERT-CNN-BiLSTM, with Class Weights",
    get_data_fn=pipeline.get_train_test_data,
    use_class_weights=True
)

all_results["Experiment 4"] = pipeline.run_experiment(
    df,
    exp_name="DistilBERT-CNN-BiLSTM, Resampled + Class Weights",
    get_data_fn=pipeline.get_train_test_data,
    resample=True,
    use_class_weights=True
)

all_results["Experiment 5"] = pipeline.run_experiment(
    df,
    exp_name="DistilBERT-CNN-BiLSTM, Ablation Study",
    get_data_fn=pipeline.get_ablation_study_data
)
```

```
data = {
   "Model": ["Base", "Resampled", "Class Weights", "Resampled + Weights", "Ablation"],
   "F1_Class0": [0.3587, 0.3917, 0.4091, 0.4066, 0.2886],
   "F1_Class1": [0.9322, 0.9257, 0.9199, 0.9276, 0.9328],
   "F1_Class2": [0.8036, 0.8147, 0.8000, 0.8000, 0.7957],
   "Accuracy": [0.8832, 0.8751, 0.8648, 0.8767, 0.8834],
   "Kappa": [0.6805, 0.6706, 0.6541, 0.6685, 0.6636],
   "AUC": [0.9260, 0.9148, 0.9197, 0.9159, 0.9236]
}
df = pd.DataFrame(data)

df_f1 = df.melt(id_vars="Model", value_vars=["F1_Class0", "F1_Class1", "F1_Class2"],
             var_name="Class", value_name="F1 Score")

df_acc_kappa = df.melt(id_vars="Model", value_vars=["Accuracy", "Kappa"],
                    var_name="Metric", value_name="Score")

plt.figure(figsize=(10,6))
sns.barplot(data=df_f1, x="Model", y="F1 Score", hue="Class", palette="Set2")
plt.title("Class-wise F1 Scores per Model")
plt.ylim(0.25, 1.0)
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()

plt.figure(figsize=(10,6))
sns.barplot(data=df_acc_kappa, x="Model", y="Score", hue="Metric", palette="Paired")
min_val = df_acc_kappa["Score"].min()
max_val = df_acc_kappa["Score"].max()
margin = 0.05
plt.ylim(min_val - margin, max_val + margin)
plt.title("Accuracy and Cohen's Kappa per Model")
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```

APPENDIX C POSTER

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

DEVELOPING A DEEP LEARNING MODEL TO DETECT SOCIAL MEDIA HATE SPEECH TEXT

UNIVERSITI TUNKU ABDUL RAHMAN

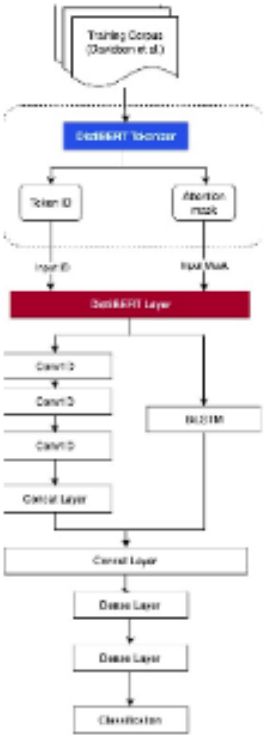
Project Author BESTER LOO MAN TING
Project Supervisor DR RAMESH KUMAR AYYASAMY

INTRODUCTION

- Social media fosters free expression but also spreads hate speech.
- Manual moderation alone can't handle the vast content volume.
- Hate speech definitions vary by context, culture, and law.
- Hate and offensive content are often linguistically similar.
- Current filters are rule-based, rigid, and easy to bypass.
- Deep learning offers promise but lacks robustness and adaptability.
- Detecting subtle, implicit hate remains a major research challenge.

OBJECTIVE

- Experiment on hybrid DistilBERT + Deep Learning Hybrid for detecting hate, offensive, and neutral content.
- Perform Ablation Study on Textual Pre-Processing
- Address Data Imbalance using Resampling & Class Weight
- Utilize multi-class classification with English unstructured text (Davidson et al. Dataset)
- Evaluate performance using precision, recall, F1-score, and accuracy.

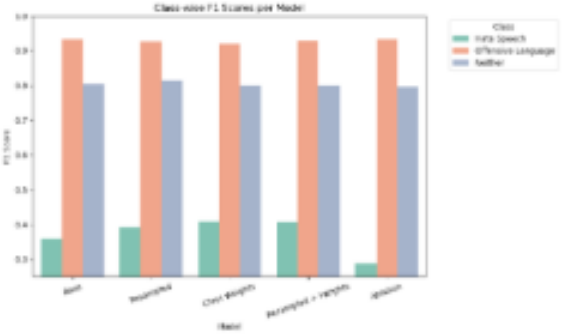


METHODOLOGY

- Data Preprocessing: Cleaned dataset by removing noise, symbols, URLs, and normalizing text. Utilized DistilBERT tokenizer.
- Ablation Study: Preprocessing step was removed in separate experiment to evaluate system performance
- Data Imbalance Resolution: Employed resampling technique
- Data Splitting: Split data into 6:2:2
- Model Development: DistilBERT Feature Extraction, CNN Branch + BiLSTM Branch
- Hyperparameters: ReLU, AdamW Optimizer, EarlyStopping, 52 token sequence length

RESULT

- Ablation Study: Critically reduce model performance in generalizing hate speeches removing preprocessing
- Key Achievement:
  - Obtained 88% accuracy score across all experiments
  - F1 score for Hate Speech of Model trained with Class Weight achieved 5% higher than existing model Advanced BERT-CNN.
  - Used DistilBERT with CNN and BiLSTM which has significantly reduced number of parameters



FINAL YEAR PROJECT