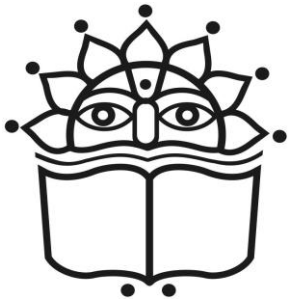


Project Report on

Unveiling Online Hate using Deep Learning Techniques

Submitted in partial fulfillment of the requirements for the Degree of

Bachelor of Engineering in Information Technology

by

Yash Suhas Shukla (B190358574)

Tanmay Dnyaneshwar Nigade (B190358554)

Suyash Vikas Khodade (B190358536)

Prathamesh Dilip Pimpalkar (B190358565)

Under the guidance of

Mr. R. V. Panchal

Department of Information Technology

Vidya Pratishthans

Kamalnayan Bajaj Institute of Engineering and Technology

Baramati-413133, Dist-Pune (M.S.)

India

April 2024

VPKBIET, Baramati

Certificate

This is to certify that the project report on

Unveiling Online Hate using Deep Learning Techniques

submitted by

Yash Suhas Shukla (B190358574)

Tanmay Dnyaneshwar Nigade (B190358554)

Suyash Vikas Khodade (B190358536)

Prathamesh Dilip Pimpalkar (B190358565)

are bonafide of this institute and the work has been carried out by them under the guidance of Mr.R.V.Panchal and it is approved for the partial fulfillment in the partial fulfillment of the requirement for the award of Degree of ***Bachelor of Engineering*** in Information Technology at Vidya Pratishthan's Kamalnayan Bajaj Institute Of Engineering and Technology, Baramati under the Savitribai Phule Pune University, Pune. This work is done during year 2023-24 Sem-II, under our guidance.

Mr. R. V. Panchal	Dr. S. A. Takale	Dr. R. S. Bichkar
Project Guide	Head of Dept.	Principal

Examiner 1:——-

Examiner 2: - - - - - - -

Acknowledgements

With a deep sense of gratitude, we would like to express our sincere thanks to our guide Mr. R. V. Panchal, Assistant Professor, IT Department, Vidya Pratishthan's Kamalnayan Bajaj Institute of Engineering and Technology, Baramati for giving us the opportunity to work under them on the project "Unveiling Online Hate using Deep Learning Techniques".

We truly appreciate and value their esteemed guidance and encouragement from the beginning to end of this project. We are extremely grateful to him. We want to thank to all my teachers for providing a solid background for our studies and research thereafter. They have been great source of inspiration to us, and we thank them from the bottom of our heart.

We would like to thank our department for giving us the opportunity and platform to make our effort a successful one.

Thank You!

Mr. Yash Suhas Shukla
Mr. Tanmay Dnyaneshwar Nigade
Mr. Suyash Vikas Khodade
Mr. Prathamesh Dilip Pimpalkar

Abstract

The proliferation of hate speech across various digital mediums, especially on social media platforms, has become an increasingly concerning issue due to its detrimental effects on individuals and communities. Hate speech targeting attributes such as race, gender, religion, and ethnicity has been linked to societal unrest and harm. Addressing this challenge requires innovative solutions that can detect and categorize hate speech effectively. Our project aims to tackle this problem by developing a comprehensive hate speech detection system capable of analyzing diverse forms of content. In addition to traditional text-based detection, our system extends its capabilities to audio, image, and video content analysis, as well as real-time monitoring of YouTube live stream comments. This versatility enables proactive identification and mitigation of hate speech across various digital channels.

In this project, an innovative approach to hate speech detection is proposed by leveraging state-of-the-art transformer-based language models, particularly Bidirectional Encoder Representations from Transformers (BERT). This technique involves fine-tuning BERT models pretrained on large multilingual corpora to effectively detect hate speech. By utilizing multilingual BERT pretrained models, we aim to address the nuances of hate speech detection in languages like Hinglish, English, and Hindi.

Furthermore, hate speech detection system has been deployed through Telegram bots and a user-friendly web interface, enhancing accessibility and usability for a wide range of users. This project represents a proactive step towards creating a safer and more inclusive online environment, fostering positive social interactions and community engagement.

Contents

List of Figures

List of Tables

Keywords

List of keywords-

- Deep Learning
- Bidirectional Encoder Representations from Transformers (BERT)
- Natural Language Processing (NLP)
- Code-mixed
- Emoticons
- Telegram Bots

Notation and Abbreviations

- BERT - Bidirectional Encoder Representations from Transformers
- DL - Deep Learning
- NLP - Natural Language Processing
- NLU - Natural Language Understanding
- API - Application Programming Interface
- GUI - Graphical User Interface

Chapter 1

Introduction

1.1 Introduction

The rapid spread of social media platforms in recent years has ushered in an era of unprecedented global connectivity and communication. However, this technological revolution has come at a price, with one of the most significant and pressing concerns being the rampant spread of hate speech on these platforms. Hate speech, characterized by its derogatory and offensive nature, poses a serious threat to individuals and communities, as it can incite violence, discrimination, and division based on attributes such as race, gender, religion, and ethnicity. Hate speech can also include nonverbal depictions and symbols. For example, the Covid Crisis, the sensitive issues like Bullibai, and pornography have all been considered hate speech by a variety of people and groups. This scenario has led to real-world consequences, including hate crimes, social unrest, and damage to individual lives and property. The pervasive nature of hate speech, present in various forms such as text, audio, images, GIFs, and videos, poses a significant challenge to maintaining a safe and inclusive online environment.

This research centers on the critical task of hate speech detection using advanced NLP and deep learning models. Specifically, this project leverages the power of the transformer-based language model known as BERT. In addition to conventional text-based hate speech, the research extends its contributions to encompass the detection of hate speech involving emoticons. The HASOC 2023 dataset that contains various

kinds of tweets based on different events such as Political, Religion, Entertainment, etc. Another dataset has also been prepared by scraping Hateful comments from various social media websites based on various sensitive social, political and religion-based issues. By using both of these datasets, model has been trained to classify text as hate or non-hate text. This research represents a significant step towards combating hate speech on social media platforms. By harnessing the power of deep learning models and extending their capabilities to include the detection of hate speech involving emoticons and internet acronyms, the aim is to foster a more positive and respectful online discourse while contributing to the creation of safer digital spaces for all.

Recognizing the urgent need to address this issue, the project focuses on developing a comprehensive hate speech detection system capable of identifying and flagging hateful content across different mediums. By harnessing the power of deep learning and natural language processing (NLP) techniques, the aim is to provide users, social media platforms, and law enforcement agencies with a robust tool to combat the spread of hate speech online. Through a multidimensional approach, our project not only tackles hate speech in traditional text format but also extends its reach to audio, images, GIFs, videos, and live YouTube comments. By analyzing content across these diverse formats, the aim is to create a more inclusive and effective detection system that can adapt to the evolving landscape of online communication.

Furthermore, ethical implications have been recognized inherent in developing such technology. With a commitment to fairness, privacy protection, and bias mitigation, the project strives to ensure that our hate speech detection system operates with integrity and accountability. The project seeks not only to detect hate speech but also to promote understanding, empathy, and dialogue in online communities.

1.2 Motivation

The motivation behind this research comes from the undeniable fact that social media has revolutionized global communication, connecting people from all corners of the world. This connectivity, however, comes at a price, with hate speech becoming a pressing concern. Hate speech can incite violence, discrimination, and division based on race, gender, religion, and ethnicity. It's not limited to text but also includes emoticons and nonverbal depictions. The consequences are tangible and can result in hate crimes, social unrest, and damage to lives and property.

The motivation behind this research is to leverage advanced NLP and deep learning models, such as BERT, to develop a hate speech detection system. This system goes beyond text to detect hate speech involving user inputs like audio, images, GIFs, and videos. By doing so, it aims to combat hate speech effectively and contribute to the creation of safer digital spaces. Ultimately, the motivation is to promote a more positive and respectful online discourse, fostering a sense of responsibility and ethical use of social media platforms. The goal is to mitigate the harm caused by hate speech and encourage global connectivity while minimizing its negative consequences.

Chapter 2

Literature Survey

Jacob Devlin (2019) [1]
In their work, Jacob Devlin, MingWei Chang, Kenton Lee, and Kristina Toutanova introduced BERT, a groundbreaking language model that utilizes masked language models for pre-training. BERT's bidirectional Transformer encoder architecture is demonstrated to excel in a wide range of natural language understanding tasks without extensive task-specific modifications. This approach stands in contrast to traditional left-to-right models or concatenations of unidirectional models. The study also provides a visual comparison of BERT with other popular representation learning models like ELMo and OpenAI GPT. While MLM pre-training converges slightly slower than LTR pre-training, it achieves superior accuracy early in the training process.

Hajung Sohn (2019) [2] In their work, authors Hajung Sohn and Hyunju Lee explore the utilization of BERT models in a finetuning approach for multilingual hate speech detection. They introduce the concept of a multi-channel BERT finetuning model, combining three BERT models for parallel language data. By creating a weighted sum of hidden states from different BERT models, they achieve impressive results with the multi-channel model, surpassing the state-of-the-art in accuracy and F1 score. Their approach showcases the potential of leveraging multilingual BERT models for improved hate speech detection, offering a valuable contribution to the field.

Sreelakshmi Ka (2020) [3] In their study, Sreelakshmi Ka, Premjith Ba, and Soman K.P. investigate hate speech detection in Hindi-English text using pre-trained word embeddings, specifically fastText. They collected a dataset of 10,000 samples from various sources categorized as hate and nonhate. The authors compare the performance of their methodology with word2vec and doc2vec embeddings, finding that word2vec outperforms doc2vec in hate speech detection. Additionally, they mention the development of domain specific word embeddings from Hindi-English tweets for this purpose. The study presents a promising approach for identifying hate speech, with potential extensions for finer grained classification in the future.

Vasu Gupta (2021) [4] In their study, authors Rahul, Vasu Gupta, Vibhu Sehra, and Yashaswi Raj Vardhan explore various model architectures for character-level embedding based multiclass labeling of Hinglish tweets into three categories: non-offensive, abusive, and hate-inducing. They experimented with more than 12 models, finding that the hybridization of GRU with an Attention Model performed the best among them. They also emphasize the importance of monitoring code-mixed language on social media due to its prevalence. Their work demonstrates that combining the features extracted from different models, such as Bi-LSTM and GRU, improves overall model understanding. Additionally, they suggest the potential enhancement of their model's performance through the use of transformer models in the future.

Chayan Paul (2021) [5] Authors Chayan Paul and Pronami Bora explore various aspects of research using data from social networking sites, beyond just sentiment analysis. They delve into detecting users with shared interests in products or services and the detection of abusive language on these platforms. Their work emphasizes the critical need for labeled datasets to train machine learning and deep learning models for hate speech detection. They also highlight the common procedure of data collection, cleaning, and manual annotation by experts to determine if a text contains hateful content. The study discusses investigation of optimal features for hate speech classification using dataset of 16,000 tweets and the application of LSTM and Bi-LSTM models for this purpose.

Snehaan Bhawal (2021) [6] In their research, authors Snehaan Bhawal, Pradeep Kumar Roy, and Abhinav Kumar conducted a comprehensive study on hate speech and offensive language detection, particularly focusing on low-resource and code mixed languages like Tamil and Malayalam. They explored various machine learning, deep learning, and transfer learning models and found that transfer learning models, particularly the MuRIL model, exhibited the best performance. Their work highlights the challenges in detecting hate speech in such languages and the potential of pre-trained models in addressing this challenge, offering valuable insights for NLP applications in code-mixed settings.

Sanjiban Sekhar Roy , Akash Roy, Pijush Samui, Mostafa Gandomi, and Amir H. Gandomi's (2021) [7] research focuses on developing a hate speech detection system for real time tweets. They collected and preprocessed Twitter data using Tweepy API, employing both searching and streaming APIs. The study employed various machine learning algorithms, including LSTM, SVM, NB, XGB, RF, k-NN, LR, ANN, and BERT, to distinguish between hateful and non hateful tweets. Their work showcases the use of advanced ML techniques and visualizations, such as word clouds, to enhance hate speech detection accuracy in multilingual real-time tweets.

Ravindra Nayak (2022) [8] In their research, authors Ravindra Nayak and Raviraj Joshi present a comprehensive study of various BERT model variations pre-trained on the L3Cube-HingCorpus with a focus on code-switching and mixed-script languages. They assess model performance through perplexity scores and evaluate them on downstream NLP tasks using the EN-HI pair from GLUECoS, a code switching benchmark dataset. Their models include HingBERT, HingMBERT, HingRoBERTa, and mixed-script variants, demonstrating the superiority of RoBERTa-based models. They also introduce HingFT, a distributed word representation model, and compare their models to baseline BERT and XLMRoBERTa models, highlighting the efficacy of mixed-script models for such tasks.

Sabit Hassan (2022) [9] In their research, authors Hamdy Mubarak, Sabit Hassan, and Shammur Absar Chowdhury conduct a comprehensive analysis of offensive language on social media platforms, with a focus on Arabic content. They propose a novel method utilizing emojis as anchors to identify offensive tweets, achieving a higher offensive tweet detection rate compared to previous approaches. The study reveals that certain emojis are commonly used in offensive communications, offering insights into linguistic usage patterns. They introduce a valence score to quantify term offensiveness and identify top offensive words. Additionally, the authors investigate offensive content in Bengali tweets, building upon the established understanding of emoji usage in English.

Maha Jarallah Althobaiti's (2022) [10] research paper titled "BERT-based Approach to Arabic Hate Speech and Offensive Language Detection in Twitter: Exploiting Emojis and Sentiment Analysis" addresses the critical issue of identifying offensive language and hate speech in online content. The study focuses on Arabic text and employs a BERT-based approach, incorporating sentiment analysis and emojis to enhance detection accuracy. The dataset used in the research consists of 12,698 tweets annotated for offensive language, including insults, threats, slurs, and violent content. The paper contributes to the growing body of work on Arabic hate speech and offensive language detection, with a focus on the unique challenges posed by the Arabic language in online communication. Various related studies and shared tasks in this domain are referenced, highlighting the importance and potential of this research area.

Arun Kumar (2022) [11] The authors, Yadav, Arun Kumar; Kumar, Abhishek; ., Shivani; ., Kusum; Kumar, Mohit; Yadav, Divakar, address the underexplored domain of hate speech detection in Hindi-English code-mixed data, a challenging task due to the scarcity of appropriate datasets and code-mixing complexities. They adopt a multi-faceted approach involving monolingual embeddings and supervised classifiers with

transfer learning from English datasets to tackle this issue. The creation of a consolidated EnglishHindi code-mix dataset from three publicly available sources significantly enhances the dataset's size, yielding over 20,000 samples. Their CNNBiLSTM-based approach stands out as it surpasses state of-the-art methods, achieving an impressive 87.6on the consolidated dataset, making a valuable contribution to hate speech detection in multilingual code-mixed contexts.

Hind Saleh (2023) [12] In their research, Hind Saleh, Areej Alhothali, and Kawthar Moria address the pressing issue of online hate speech detection using advanced techniques like BERT and domain-specific word embeddings. They utilize a unique hate speech word2vec model, HSW2V, trained on hate speech domains to capture the semantics of code words used in hate speech. The authors emphasize the importance of distinguishing between offensive language and hate speech, collapsing classes for effective binary classification. Their work also includes insightful analysis of misclassifications, shedding light on word contributions to classification decisions. This study contributes to the ongoing efforts in hate speech detection in a nuanced manner.

Ravindra Nayak (2023) [13] In their research, authors Ravindra Nayak and Raviraj Joshi employ a context-aware approach to detect hateful content in tweets. They emphasize the significance of considering the context of a tweet, including parent tweets, to determine the presence of hate speech. The study explores various fine-tuning methods and context-aware models, leveraging previous tweet features as input alongside the current tweet. Notably, they implement a dual encoder approach, utilizing BERT embeddings for both context and tweet, leading to improved F1 scores by capturing contextual nuances effectively. Their work underscores the importance of context in hate speech detection within the Twitter platform.

Qizheng Wang (2023) [14] In this overview, author Qizheng Wang delves into the complex issue of hate speech, particularly its prevalence and detection on social media platforms. The piece highlights the challenges of defining and categorizing hate speech

and discusses the subjectivity involved in labeling it. Wang also mentions the integration of user modeling as a parameter for classifying hate speech and notes the legal differences regarding hate speech in the United States and China. This comprehensive examination serves as a valuable resource for understanding the nuances and challenges of hate speech detection in today's digital landscape.

Khouloud Mnassri (2023) [15] The paper by Khouloud Mnassri, Praboda Rajapaksha, Reza Farahbakhsh, and Noel Crespi discusses hate speech and offensive language detection using multi-task learning (MTL) models. They found that the MTL approach improved hate speech detection performance by 3offensive language detection. The study compared single-task learning (STL) and MTL models, showing that MTL models generally performed better and exhibited fewer false positive errors. The MTL models, particularly the mBERT-MTL model, outperformed STL models in hate speech detection, even on imbalanced datasets. The research highlights the value of emotional knowledge in improving classification for these tasks.

Nirali Arora (2023) [16] The authors Nirali Arora, Aartem Singh, Laik Shaikh, Mawrah Khan, and Yash Devadiga discuss the importance of word representation in natural language processing and highlight the use of word embeddings to convert words into dense vectors. They emphasize the limitations of sparse representations, such as the Bag-of-Words model, which can be inefficient in terms of space usage and fail to capture word semantics effectively. The paper introduces Word2Vec and FastText models as methods for distributed word representation, with a focus on FastText's ability to consider word morphology. The authors also describe a step-by-step process for word preprocessing, including the removal of extra letters from swear words, which can be valuable for offensive language detection.

Umitcan Sahin, Izzet Emre Kucukkaya, Oguzhan Ozcelik, and Cagri Toraman (2023) [17] present a novel approach for multimodal hate speech detection. They propose a model that combines multimodal deep learning with text-based tabular features, such as named entities and syntactical features, to enhance hate speech detection in both image and OCR generated text. Their approach demonstrates superior performance compared to existing methods, including tabular, textual, visual, and multimodal baselines, showcasing the effectiveness of their ensemble learning strategy for hate speech detection in a multimodal setting.

Author	Paper Name	Year	Accurancy	F1 Score	Limitations
Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova.	BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding	2019	GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).	BERTLARGE (Sgl.+TriviaQA) = 91.8 BERTLARGE (Ens.+TriviaQA) = 93.2	Limitations of BERT include its reliance on deep bidirectionality, making alternative models less effective, increased complexity with additional layers, and the inefficiency of concatenating separate LTR and RTL models, emphasizing the importance of BERT's unique pre-training design choices for NLP tasks.
Hajung Sohn, Hyunju Lee.	MC-BERT4HATE: Hate Speech Detection using Multi-channel BERT for Different Languages and Translations	2019	Multilingual BERT = 82.2 % English BERT = 79.8% Chinese BERT = 79.9% multi-channel model = 80.0%	Multilingual BERT = 79.9 English BERT = 77.3 Chinese BERT = 77.5 multi-channel model = 77.5	The limitation in this approach is that it relies on the availability of parallel language data to fine-tune multiple BERT models, which may not be accessible for all language pairs or domains, limiting its applicability in multilingual settings.
Sreelakshmi ka, Premjith B , Soman K.P.	Detection of Hate Speech Text in Hindi-English Code-mixed Data	2020	*Experiment 01* Random Forest = 64.15% *Experiment 02* SVM-RBF = 75.11% *Experiment 03* SVM-RBF = 85.81% *Overall* SVM-RBF = 85.81%	*Experiment 01* Random Forest = 64.14 *Experiment 02* SVM-RBF = 75.09 *Experiment 03* SVM-RBF = 85.80	The limitations in the provided information pertain to the model's language-specific focus, reliance on a relatively small dataset, potential difficulties in cross-language translation, sensitivity to pre-trained embeddings, and the oversimplification of hate speech into a binary classification, emphasizing the need for broader data, more robust cross-lingual models, and nuanced hate speech categorization.
Rahul, Vasu Gupta, Vibhu Sehra, Yashaswi Raj Vardhan.	Hindi-English Code Mixed Hate Speech Detection using Character Level Embeddings	2021	*RESULTS WITH BASIC PROPOSED MODELS* GRU Model = 86% *RESULTS WITH STACKED MODELS* Bidirectional LSTM + GRU Model = 84% *RESULTS WITH MODELS CONTAINING ATTENTION LAYER* GRU with Attention Model = 87%	GRU Model = 86 Bidirectional LSTM + GRU Model = 84 GRU with Attention Model = 87	Implicit limitations include potential complexity and computational costs of the hybrid model, the limited generalizability of results due to a small dataset, and the challenges of combining diverse models. It also highlights concerns about overfitting and the context-specific performance of the models.
Chayan Paul, Pronami Bora.	Detecting Hate Speech using Deep Learning Techniques	2021	*PERFORMANCE MEASURE SCORES FOR LSTM AND BI-LSTM* LSTM = 97.85% Bi-LSTM = 97.81%	LSTM = 97.85 Bi-LSTM = 97.81	The limitations involve the labor-intensive process of manual annotation, data cleaning challenges, the impact of a relatively small dataset, difficulty in feature selection, and potential limitations in classifying hate speech nuances, emphasizing the complexities in this research area.
Snehaan Bhawal , Pradeep Kumar Roy and Abhinav Kumar.	Hate Speech and Offensive Language Identification on Multilingual code-mixed Text using BERT	2021	———	Malayalam Code-Mixed BERT = 72 MuRIL = 73 Tamil Code-Mixed BERT = 63 MuRIL = 64	The limitations include potential challenges in model generalization to different languages, the dataset's limited coverage of hate speech variations, language-specific effectiveness, and the lack of information regarding the selected model's limitations or biases. These factors underscore complexities in hate speech detection, particularly in low-resource language scenarios.
Sanjiban Sekhar Roy, Akash Roy, Pijush Samui, Mostafa Gandomi, and Amir H. Gandomi ,	Hateful Sentiment Detection in Real-Time Tweets: An LSTM-Based Comparative Approach	2021	NB = 92%, SVM = 93%, LR = 93%, XGB = 95%, RF = 92%, k-NN = 94%, ANN = 94%, BERT = 93%, LSTM = 97%	———	Limitations include potential data source biases, the language-specific nature of the method, feature engineering sensitivity, and a lack of clarity regarding evaluation criteria, emphasizing the complexity of effectively detecting hate speech in real-time Twitter data.

Table 2.1 : Literature Survey

Author	Paper Name	Year	Accurancy	F1 Score	Limitations
Ravindra Nayak and Raviraj Joshi.	L3Cube-HingCorpus and HingBERT: A Code Mixed Hindi-English Dataset and BERT Language Models	2022	_____	F1 scores of test sets after fine-tuning on various downstream tasks of the GLUECoS dataset in Roman script. HingBERT-LID :- HingLID = 98.77	Potential limitations include context-specific model applicability, data quantity impacts, unclear evaluation details, code-switching specificity, and resource-intensive BERT models, emphasizing considerations for broader utility and resource constraints.
Hamdy Mubarak,, Sabit Hassan , and Shammur Absar Chowdhury.	Emojis as Anchors to Detect Arabic Offensive Language and Hate Speech	2022	*Macro-averaged (P)recision, (R)ecall and F1 for Offensive language classification. * represent results obtained with different learning rate (2e-5) instead of 8e-5. C: Character n-grams W: Words n-grams.* QARiB = 84.02 % *Macro-averaged (P)recision, (R)ecall and F1 for hate speech classification. * represent results obtained with different learning rate (2e-5) instead of 8e-5. C: Character n-grams W: Words n-grams.* AraBERT = 92.64%	QARiB = 83.31 AraBERT = 80.14	The limitations include context-dependent emoji interpretations, language-specific focus, emoji redundancy challenges, and potential variability in valence score accuracy, emphasizing the complexity of using emojis for universal offensive content detection.
Maha Jarallah Althobaiti.	BERT-based Approach to Arabic Hate Speech and Offensive Language Detection in Twitter: Exploiting Emojis and Sentiment Analysis	2022	*Offensive language detection* Proposed model = 93.30% *Hate speech detection* Proposed model = 85.90%	*Offensive language detection* Proposed model = 81.80 *Hate speech detection* Proposed model = 84.30	The study's limitations include a focus on Arabic language, a relatively small dataset, subjectivity in task definitions, and limited exploration of model applicability to broader contexts. These constraints emphasize the need for caution when applying findings to other languages and platforms.
yadav, arun kumar; Kumar, Abhishek; ., Shivani; ., Kusum; Kumar, Mohit; Yadav, Divakar	Hate Speech Recognition in multilingual text: Hinglish Documents	2022	CNN-BiLSTM = 87.6%	CNN-BiLSTM = 83.5	The limitations of this study include the lack of prior research on hate speech in Hindi-English code-mixed data, potential concerns about dataset quality from merging multiple sources, and a lack of exploration regarding the model's applicability beyond this specific language context, emphasizing the need for further research and cautious data handling.
Hind Saleh, Areej Alhothali & Kawthar Moria.	Detection of Hate Speech using BERT and Hate Speech Word Embedding with Deep Model	2023	_____	BERT achieved F1 Score = 96	This study faces limitations due to the scarcity of prior research in Hindi-English code-mixed hate speech detection, dataset compilation challenges, and the lack of exploration into broader applicability beyond this specific context. Further research, rigorous dataset handling, and consideration of generalizability are needed.
Ravindra Nayak , Raviraj Joshi.	Contextual Hate Speech Detection in Code Mixed Text using Transformer Based Approaches	2023	_____	m-BERT + FE + C-Avg + Dictionary = 68.61 Indic-BERT + FE + C-Avg + Dictionary = 70.37 Ensemble 4 (Indic-BERT + Indic C-Avg + m-BERT + m-BERT C-Avg) = 73.07	The limitations include a small amount of training data, heavy reliance on context, and model specificity, emphasizing the need for data caution, context relevance, and model adaptability in hate speech detection.
Qizheng Wang	Advanced Deep Learning Approaches for Hate Speech Detection	2023	_____	_____	Challenges in hate speech detection include complex definitions of hate speech, the absence of a standardized and continuously updated dataset, language-specific regulations, and the need for models capable of handling various media forms beyond text. Researchers must stay current with natural language analysis models and adapt to diverse and evolving expressions of hate speech.

Table 2.2 : Literature Survey

Author	Paper Name	Year	Accurancy	F1 Score	Limitations
Khouloud Mnassri, Praboda Rajapaksha, Reza Farahbakhsh, Noel Crespi	Hate Speech and Offensive Language Detection using an Emotion-aware Shared Encoder	2023	*Hate speech detection HS* BERT MTL = 93.85% mBERT MTL = 94.13% *Offensive language detection OFF* BERT MTL = 96.91% mBERT MTL = 96.74%	*Hate speech detection HS* BERT MTL = 91.46 mBERT MTL = 92.04 *Offensive language detection OFF* BERT MTL = 94.89 mBERT MTL = 94.59	Limitations include varying effectiveness of multi-task learning, an imbalanced dataset impacting hate speech detection, and a lack of detailed error analysis, emphasizing the need for further research on MTL, class balance, and error insights.
Nirali Arora, Aartem Singh , Laik Shaikh, Mawrah Khan, Yash Devadiga.	Hinglish Profanity Filter and Hate Speech Detection.	2023	———	———	The limitations are a narrow focus on word embedding, fragmented descriptions, and a lack of coverage regarding challenges and biases, indicating a need for a more comprehensive and context-rich discussion in natural language processing.
Umitcan Sahin, Izzet Emre Kucukkaya, Oguzhan Ozcelik, Cagri Toraman.	ARC-NLP at Multimodal Hate Speech Event Detection 2023: Multimodal Methods Boosted by Ensemble Learning, Syntactical and Entity Features	2023	*Subtask A: Hate Speech Detection* ELECTRA + Swin + Tabular = 84.9% *Subtask B: Target Detection* CLIP + NER = 80.3%	*Subtask A: Hate Speech Detection* ELECTRA + Swin + Tabular = 84.8 *Subtask B: Target Detection* CLIP + NER = 79.7	The limitations are: Lack of detailed model descriptions, limited explanation of named entity usage, and absence of comparative data, making it challenging to fully understand the proposed models and their performance compared to existing methods.

Table 2.3 : Literature Survey

Chapter 3

Proposed Work

3.1 Problem Definition

In today's digital age, the proliferation of social media platforms has revolutionized the way we communicate, share information, and connect with others. However, amidst the myriad of benefits offered by these platforms, a pressing issue threatens to undermine the very essence of online interaction: hate speech. A hate speech detection system is essential in today's digital landscape to identify and mitigate harmful content that promotes discrimination, hostility, and violence. Hate speech, defined as any form of communication that disparages or targets individuals or groups based on attributes such as race, ethnicity, religion, gender, sexual orientation, disability, or nationality, has become increasingly prevalent in online spaces. This toxic phenomenon not only perpetuates harm and discrimination but also fosters division, polarization, and hostility within communities.

3.2 Project Objectives

1. Develop a hate speech detection system using advanced NLP and deep learning models.
2. Extend the system's capabilities to detect hate speech involving various forms of

input like Text (along with emoticons), Audios, Images, GIFs and Videos all in real-time.

3. Apply Hate Speech Detection System on the real-time comments made on social media by leveraging streaming APIs.
4. Develop Social Media Bots which are capable of taking various forms of input like Text, Audio, Image, GIF, Video and Video ID of social media live stream video.
5. Develop a Frontend website which is capable of taking various forms of input like Text, Audio, Image, GIF, Video and Video ID of social media live stream video.
6. Rigorously evaluating the performance of the system across diverse demographic groups and linguistic contexts to identify and mitigate any biases or disparities in detection accuracy.

3.3 Scope of Project

1. The system will target Hindi, English and Hinglish addressing code- mixed content within a single text.
2. The system will encompass a diverse set of existing emoticons and will be designed to identify hate speech conveyed through emoticons.
3. The system will process various kinds of input such as Text, Audio, Image, GIF, Video in real-time to identify hate speech conveyed through them.
4. The system will process live comments made on a live stream YouTube video in real-time to identify hate speech conveyed through them.

3.4 Project Constraints

1. The proposed hate speech detection system can only detect hate speech if the data entered is only in Hinglish, English or Hindi language.

2. Emoticons are often used in conjunction with text. The model should analyze the combined effect of emoticons and surrounding text to accurately gauge intent.

3. Hate speech can be context-dependent. The model needs to understand nuances and varying interpretations, as certain words may be offensive in one context but neutral in another.

Chapter 4

Systems Architecture

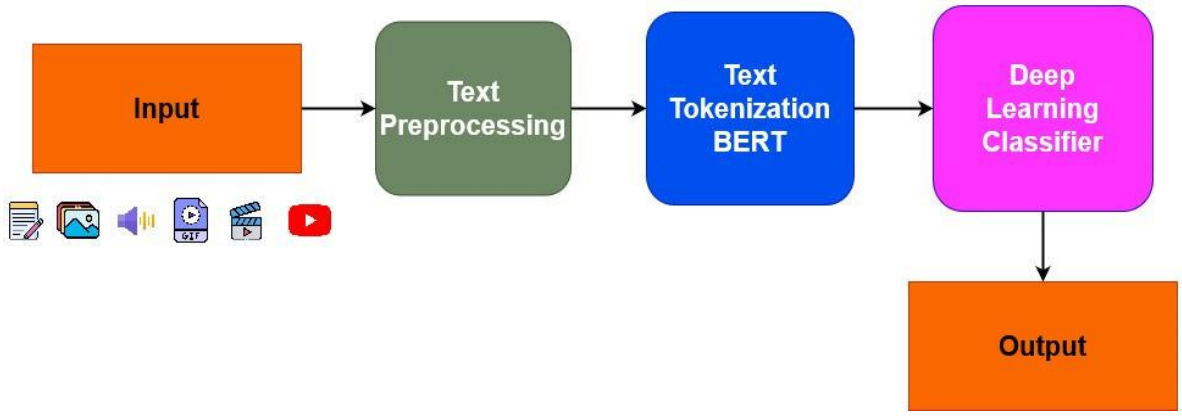


Fig 4.1 : System Architecture

Hate Speech Detection on Various Inputs:

- **Text Input:** The user can input text for hate speech detection directly.
- **Audio Input:** Hate speech prediction is performed on text extracted from audio content using Google Speech-to-Text API.
- **Image Input:** Hate speech detection is performed on text extracted from images using Google Cloud Vision API.
- **Image Input:** Hate speech detection is performed on text extracted from GIFs using Google Video Intelligence API.
- **Image Input:** Hate speech detection is performed on text extracted from Videos using both Google Speech-to-Text API and Google Video Intelligence API.
- **Live YouTube Video Input:** Live comments from a YouTube video can be classified for hate speech in real-time using the Python PytChat Library.

Text Cleaning and Data Pre-processing:

- The system begins by loading the dataset from a CSV file using Pandas.
- The dataset is split into training and testing sets.
- The dataset is cleaned by removing hyperlinks and removing stop-words,
- Text preprocessing is performed, including encoding labels using LabelEncoder and tokenizing text using the BERT tokenizer.

Model Building and Training:

- A pre-trained BERT model for sequence classification is utilized as the core model architecture.
- The model is configured with specific parameters and compiled using TensorFlow.
- AdamWeightDecay optimizer is used for optimizing the model during training.

- SparseCategoricalCrossentropy loss function is employed for calculating the loss during training.
- The model is trained on the preprocessed data for a specified number of epochs and batch size.

Model Evaluation:

- The trained model is evaluated on the testing dataset to assess its performance.
- Metrics such as test loss and accuracy are calculated.
- Classification report is generated to analyze model performance in detail.

Additional Components:

- The system incorporates various additional components such as text cleaning, audio-to-text conversion, text extraction from images, and live YouTube comment retrieval.

Chapter 5

Project Requirement Specification

5.1 Performance Requirements

1. The model must accurately distinguish hate speech from non-hate speech, minimizing errors for effective content filtering.
2. The system should efficiently handle large volumes of Text, Audio, Image, GIF and Video data in real-time to keep up with the constant flow of content on social media.
3. The system should efficiently handle large volumes of Text Comments from YouTube Live Stream Video in real-time to keep up with the constant flow of content on social media.
4. The hate speech detection system must deliver fast results for timely identification and intervention against hate speech.
5. The model should be proficient in detecting hate speech in Hindi, Hinglish and English language text, promoting safety across diverse online communities.

5.2 Software Quality Attribures/Requirements

1. Quality / Accuracy of system is totally dependent on dataset used for training.
2. The software should log and maintain records of its actions and decisions for auditing and transparency purposes.
3. The software should incorporate ethical considerations in its algorithms and decision-making processes to avoid bias and promote fairness.
4. The system should respond in real-time or near-real-time to detect and mitigate hate speech, ensuring prompt action to prevent harm.

5.3 Safety Requirements

1. Conduct periodic security audits to identify and address vulnerabilities.
2. Implement robust mechanisms for obtaining user consent before collecting and processing data.
3. Audios, Images, GIFs and Videos should be clear otherwise it will anticipate the wrong classification.

5.4 Security Requirements

1. The system should adhere to data protection regulations and ensure the privacy of users while monitoring and processing content.
2. The system should keep the information secured from Ethical Hackers.
3. The system should be able to prevent unauthorised access, modification, or deletion of the data used for training and testing deep learning model.

5.5 Hardware Requirements

1. Processor - Intel i9 core and above
2. GPU - NVIDIA
3. Speed - 1.1 GHz and above
4. RAM - Min 8 GB
5. Key Board - Standard Windows Keyboard
6. Mouse - Two or Three Button Mouse
7. Monitor - SVGA

5.6 Software Requirements

1. Operating system: Windows 10 and above
2. Coding Language: Python, HTML, CSS, JavaScript
3. IDE: Jupyter Notebook/Google Colab, VS Code and Pycharm
4. Database: Firebase
5. API: Speech Recognition API, Google Vision API, Video Intelligence API

Chapter 6

Project Planning

6.1 Project Code Estimates

Sr No	Task Name	Lines of Code
1	Hate Speech Detection Model Training	100
2	Applying Various Input Forms	300
3	Frontend Development	400
4	Telegram Bots Development	400
	Total	1200

Table 6.1 : Project Code Estimates

6.2 Project Cost Estimates

Parameter Estimation

Cost Estimation using **COCOMO** Model

Softare Projects	a	b	c	d
Organic	1.2	1.05	2.5	0.38
Semi-Detached	1.5	1.12	2.5	0.35
Embedded	1.8	1.20	2.5	0.32

Table 6.2 : Project Code Estimates

As,

a = 1.8

b = 1.2

c = 2.5

d = 0.32

KLOC = 1.200

Effort(E) = $a(KLOC)^b$

= $1.8 * (1.200)^{1.2}$

= 2.52 person-month

= 3 persons-month

Development Time(D) = c(Effort)d

= 2.5 * (2.52)^0.32

= 3.36 months

= 3 months

Productivity(P) = KLOC/Effort

= 1/1

= 1 KLOC/Person-month

= 100 LOC/Person-month

Phases involved in Project Cost Estimation:

- Research/Planning
- Design
- Copy-writing
- Backend / Frontend / Bot Development
- Testing / Bug Fixes
- Launch

6.3 Team Structure

Sr No	Name Of Student	Worked Assigned
1	Yash Suhas Shukla	Research, Documentation, Data Collection, Dataset Preparation, Model Building, Bot Building, Testing, Project Funding Management
2	Tanmay Dnyaneshwar Nigade	Research, Frontend Website Building, Data Collection, Dataset Preparation, Model Building, Testing
3	Suyash Vikas Khodade	Research, Frontend Website Building, Data Collection, Dataset Preparation, Model Building
4	Prathamesh Dilip Pimpalkar	Research, Documentation, Data Collection, Dataset Preparation, Model Building

Table 6.3 : Team Structure

Chapter 7

Project Schedule

7.1 Project Breakdown Structure

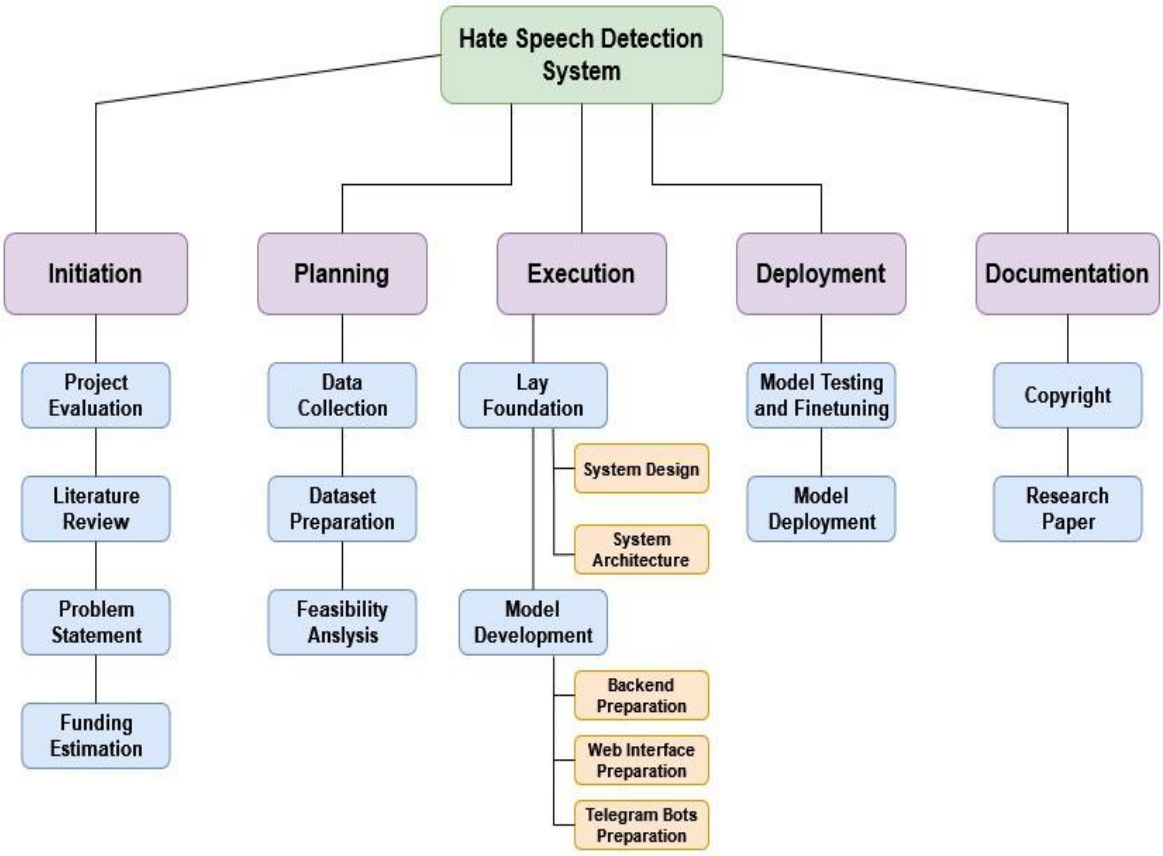


Fig 7.1 : Project Breakdown Structure

7.2 Task Network

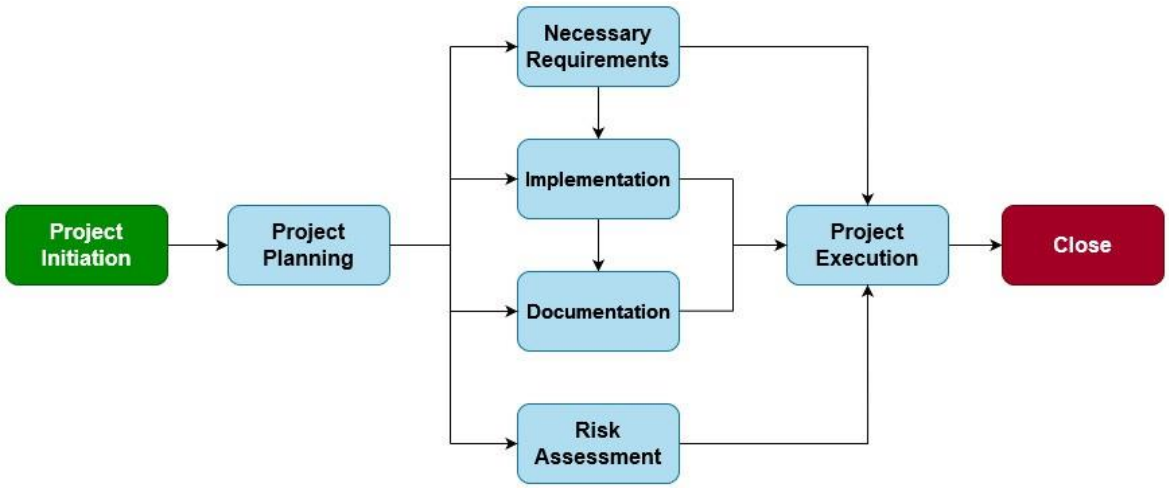


Fig 7.2 : Task Network

7.3 Gantt Chart

Fig 7.3 : Gantt Chart

Chapter 8

Project Design

8.1 Use Case Diagram

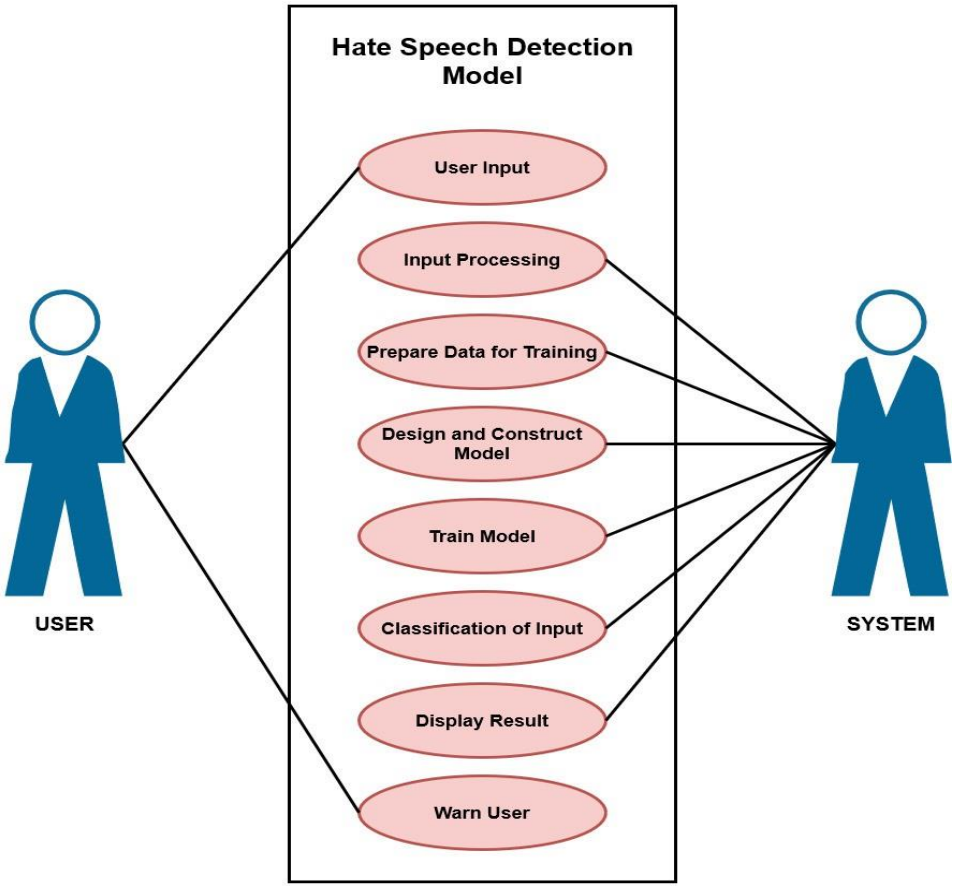


Fig 8.1 : Use Case Diagram

8.2 Activity Diagram

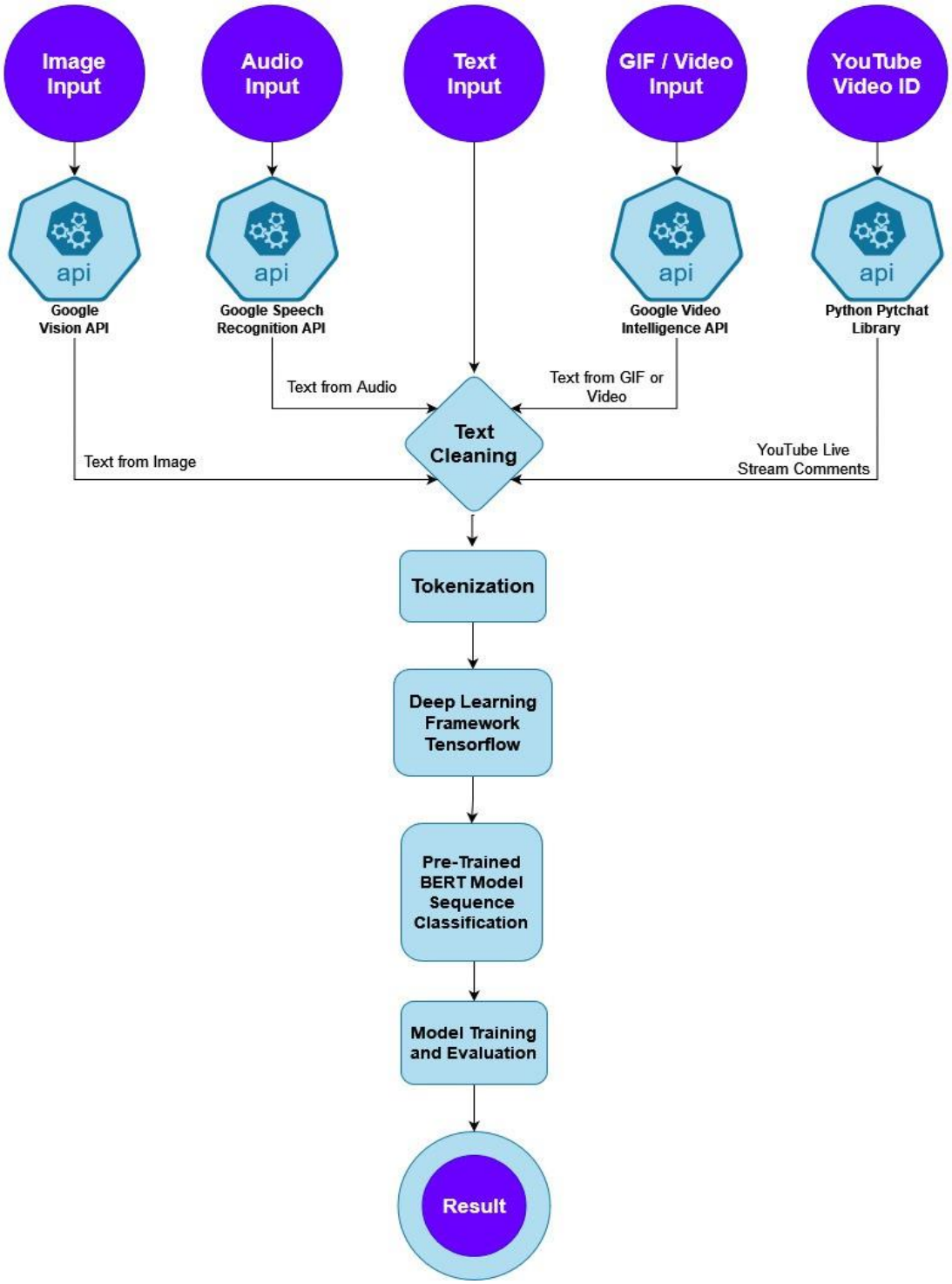


Fig 8.2 : Activity Diagram

8.3 Class Diagram

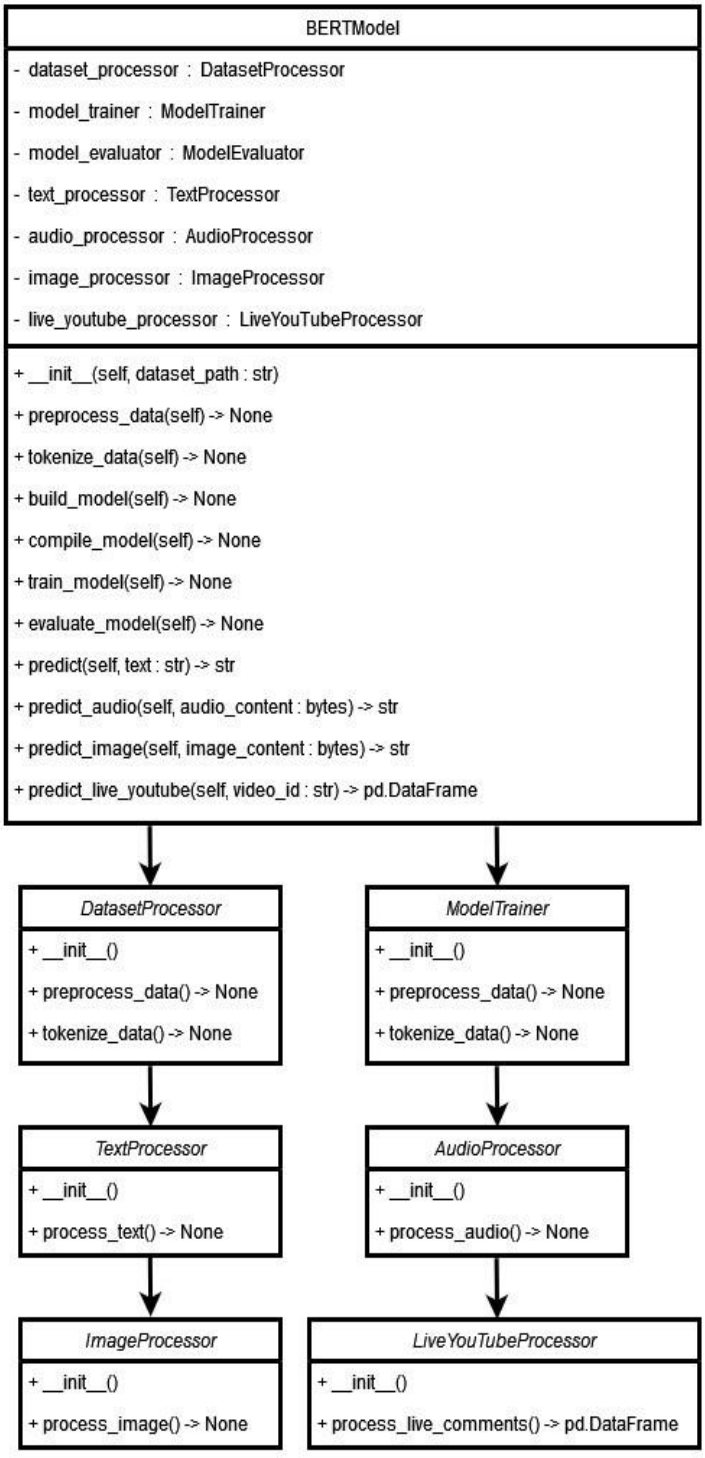


Fig 8.3 : Class Diagram

8.4 Data Flow Diagram Level 0

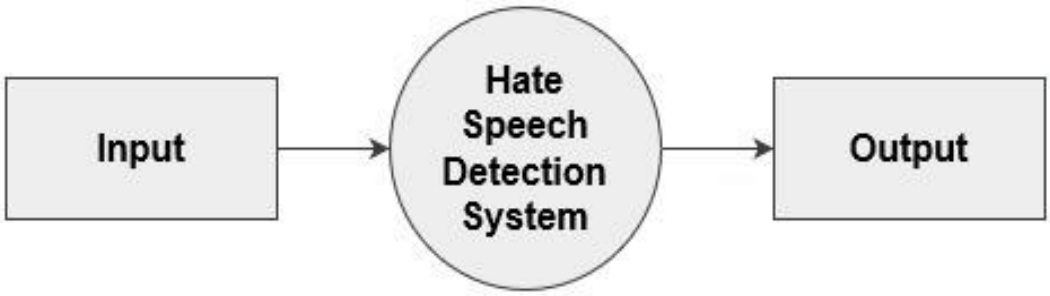


Fig 8.4 : DFD Level 0

8.5 Data Flow Diagram Level 1

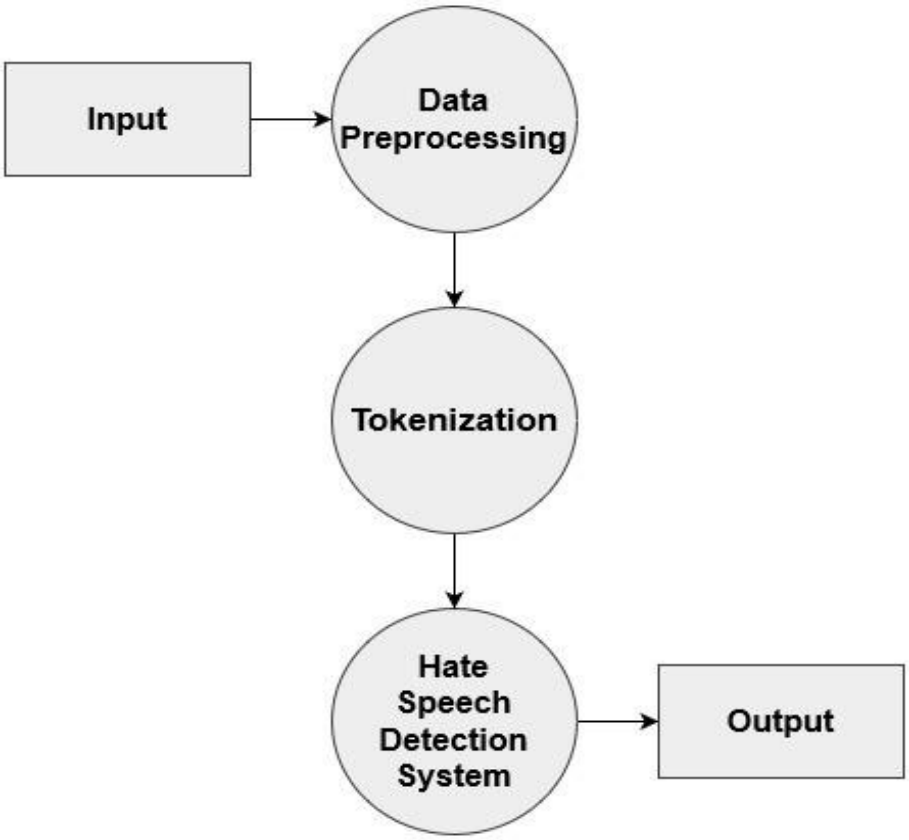


Fig 8.5 : DFD Level 1

8.6 Data Flow Diagram Level 2

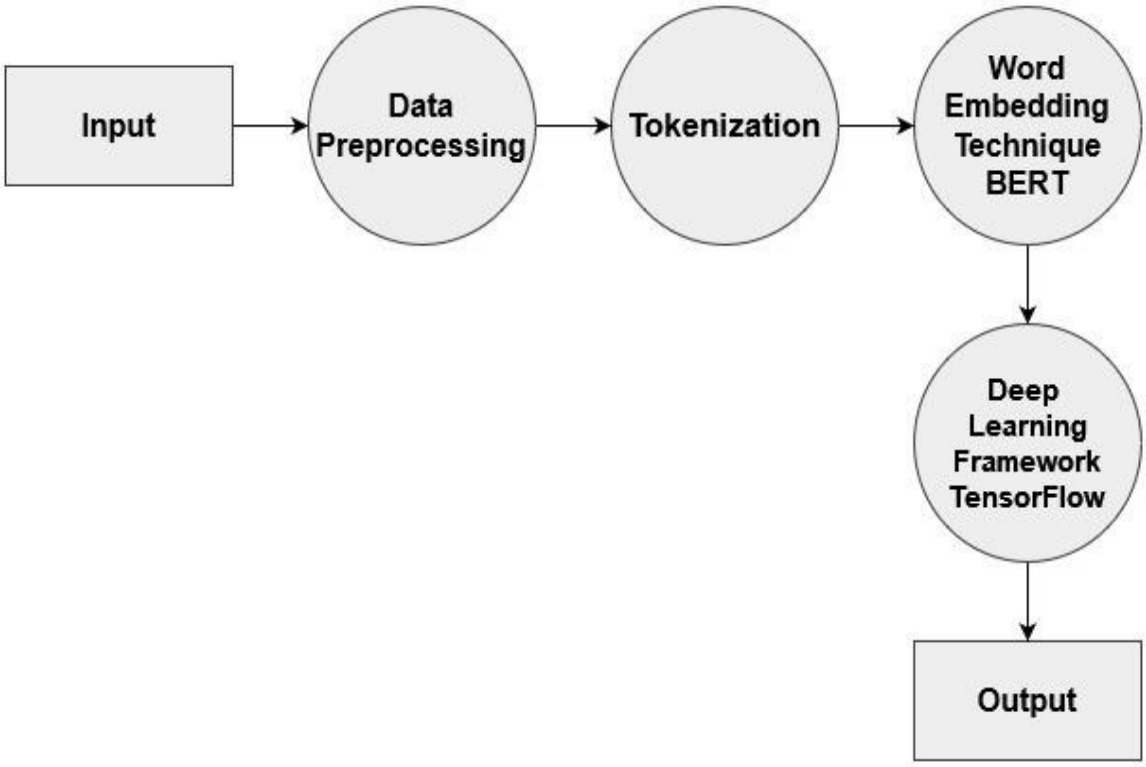


Fig 8.6 : DFD Level 2

Chapter 9

System Algorithms, Framework Design and System API

9.1 BERT(Bidirectional Encoder Representations from Transformers) Algorithm

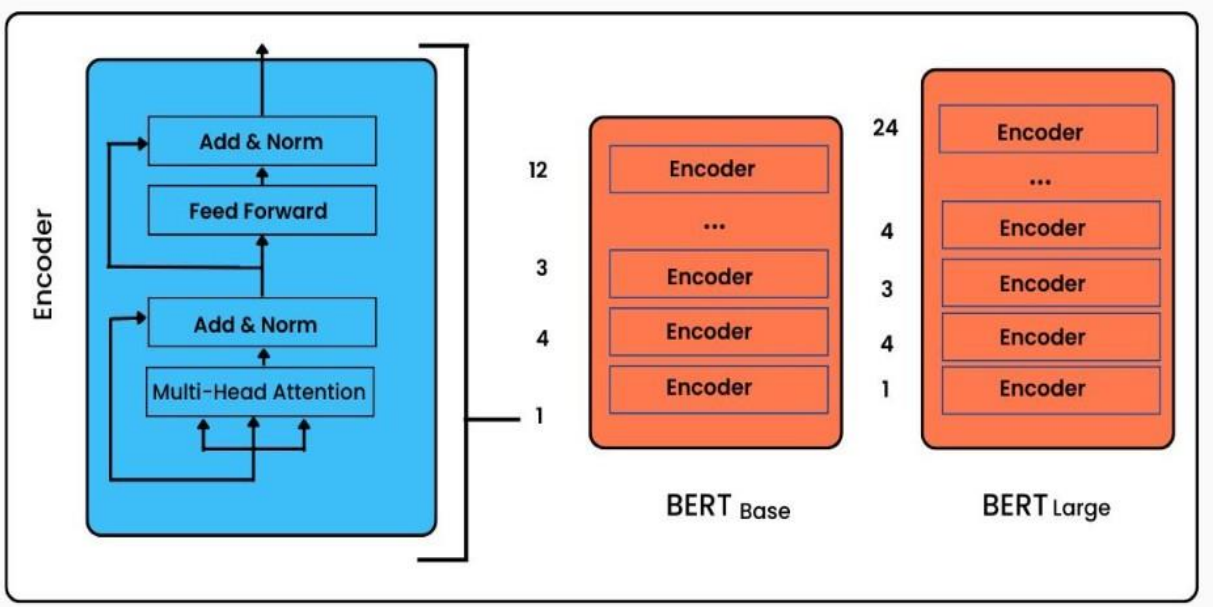


Fig 9.1 : BERT Architecture

BERT, an acronym for Bidirectional Encoder Representations from Transformers, represents a groundbreaking advancement in the field of natural language understanding. Developed by a team at Google in 2018, this powerful model has transformed the landscape of NLP.At its core, BERT is constructed from a series of stacked Transformer encoders, each consisting of a self-attention layer and a feed-forward layer.

There are two different BERT models: **BERT Base:** Think of it as a language model with 12 layers, where each layer can understand different aspects of language. It has 12 ”attention heads” that help it focus on different parts of a sentence simultaneously. It’s like having 768 small language detectives working together to understand text. BERT Base has 110 million parameters, which are like the switches and dials that fine-tune its understanding.**BERT Large:** This is like a more advanced version of BERT with 24 layers, making it even better at understanding language. It has 16 attention heads, so it can pay attention to even more details in text. Its hidden size is 1024, meaning it can remember more about what it reads. BERT Large is like having a team of 340 experts, each with their own special skills, working together to make sense of language.

These components work in harmony to analyze and comprehend contextual relationships between words in text. One of BERT’s distinguishing features is its bidirectional nature, enabling it to capture the entire sentence’s context simultaneously. This holistic approach results in superior accuracy compared to its predecessors. BERT’s application extends to a wide range of NLP tasks, from sentiment analysis to question answering.

Multilingual BERT, often referred to as mBERT, is an extension of the original BERT model specifically designed to understand and process text in multiple languages. Introduced by Google AI researchers, mBERT is trained on a diverse corpus of texts from over 100 languages, allowing it to effectively capture the nuances and intricacies of language across different cultures and regions.

One of the key advantages of mBERT is its ability to perform cross-lingual transfer learning. This means that the model can leverage knowledge gained from one language to improve its understanding and performance in another language. For example, if mBERT is trained on English and Spanish data, it can transfer the knowledge learned

from English to improve its performance on Spanish tasks, and vice versa. This makes mBERT a highly versatile and efficient tool for multilingual natural language processing tasks.

One specific variant of mBERT is bert-base-multilingual-uncased. This model is part of the BERT family and is trained on lowercased text from multiple languages. It includes a shared vocabulary across all supported languages, allowing for seamless integration and interoperability in multilingual environments. With over 100 languages represented in its training data, bert-base-multilingual-uncased is capable of understanding and processing text in a wide range of languages, making it an invaluable resource for researchers, developers, and practitioners working on multilingual NLP applications.

9.1.1 Layers

Input Embedding Layer:

- The input text data is converted into numerical representations using token embeddings and positional embeddings.
- Token embeddings represent the meaning of each word in the context of the sentence.
- Positional embeddings encode the position of each word in the sequence.

Transformer Encoder Layers:

- The BERT base model comprises multiple transformer encoder layers.
- Each layer consists of sub-layers, such as multi-head self-attention mechanisms and position-wise feed-forward networks.
- The self-attention mechanism enables the model to weigh the importance of different words in the context of the entire sequence.
- The feed-forward networks process the output of the attention mechanism to capture complex patterns in the data.

Classification Layer:

- The BERT base multilingual model has an additional classification layer for sequence classification tasks.

- This layer maps the outputs from the transformer layers to the appropriate class labels.

Dropout Layer (Regularization):

- Dropout layers are incorporated within the model for regularization purposes, which helps prevent overfitting during the training process.
- These layers randomly drop a fraction of the input units to zero,preventing complex co-adaptations during training and improving generalization.

9.1.2 Optimization Techniques

Adam (Adaptive Moment Estimation):

Adam is an optimization algorithm used in training deep learning models. It combines the advantages of two other popular optimization algorithms, namely AdaGrad and RMSProp. It dynamically adjusts the learning rate for each parameter based on past gradients and magnitudes, enabling faster convergence and improved performance. By combining first-order moments (mean) and second-order moments (uncentered variance) of the gradients, Adam adapts learning rates for different parameters, making it well-suited for optimizing complex neural network architectures like those used in hate speech detection.

Sparse Categorical Crossentropy:

Sparse Categorical Crossentropy is a loss function commonly used in multi-class classification tasks, particularly when the target variable is represented as integers. It calculates the cross-entropy loss between the true distribution and the predicted probability distribution. In the context of hate speech detection, where the goal is to classify text into multiple categories (e.g., hate speech, non-hate speech), Sparse Categorical Crossentropy helps quantify the discrepancy between the predicted probabilities assigned to each class and the actual labels. By minimizing this loss during model training, the algorithm guides the model to make more accurate predictions, ultimately improving its performance in classifying hate speech.

9.1.3 BERT in Hate Speech Detection System

1. **Importing Libraries:** BERT-related libraries such as transformers and tokenization are imported at the beginning of the code.
2. **Loading Pre-trained BERT Model:** The pre-trained BERT model is loaded using the TFBertForSequenceClassification class from the transformers library. This model is specifically designed for sequence classification tasks like hate speech detection.
3. **Tokenization:** The BertTokenizer class from the transformers library is used to tokenize the input text data. Tokenization converts each text input into a sequence of tokens suitable for BERT processing.
4. **Preprocessing Text Data:** The input text data is preprocessed using the BERT tokenizer. This involves padding and truncating the sequences to a fixed length, and converting them into input IDs and attention masks.
5. **Fine-tuning BERT Model:** The pre-trained BERT model is fine-tuned on the hate speech detection task. This involves training the BERT model with your hate speech dataset to adapt it to recognize hate speech patterns effectively.
6. **Model Compilation and Training:** The BERT model is compiled using an optimizer (e.g., Adam) and a loss function (e.g., Sparse Categorical Crossentropy). Then, it is trained on the hate speech dataset using the compiled configuration.
7. **Model Evaluation:** After training, the performance of the BERT model is evaluated using test data. Metrics such as accuracy, precision, recall, and F1-score may be calculated to assess its performance.
8. **Prediction:** Finally, the trained BERT model is used to make predictions on new text inputs. These predictions determine whether the input text is classified as hate speech or non-hate speech based on the learned patterns during training.

9.2 Tensorflow Framework

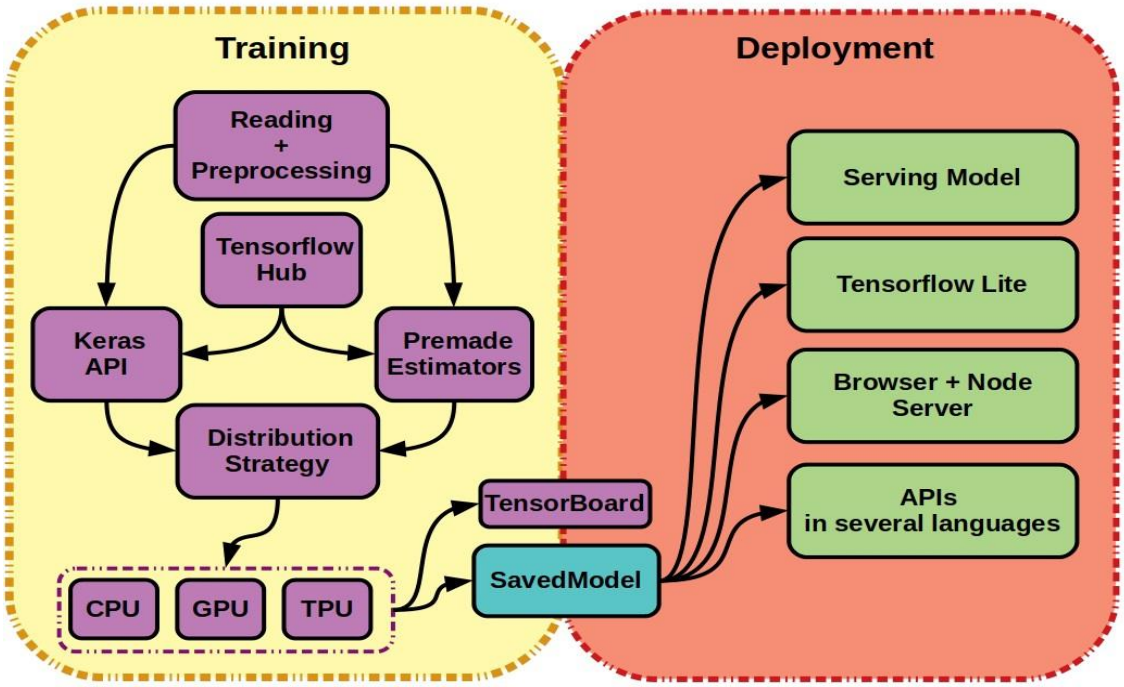


Fig 9.2 : Tensorflow Framework

TensorFlow, an open-source framework developed by Google, has emerged as a leading tool in the realm of deep learning, succeeding Google's earlier platform, DistBelief. It serves as a robust foundation for the implementation of neural networks, particularly in tasks related to language and image processing. Offering versatility, TensorFlow facilitates the creation and execution of both custom and pre-trained models across various platforms. Implemented primarily in Python and C++, TensorFlow stands out for its extensive hierarchy of toolkits and dynamic resource allocation capabilities.

TensorFlow's Hierarchy and APIs: TensorFlow provides a hierarchical structure comprising low-level APIs tailored for CPU, GPU, or TPU utilization, ensuring optimal hardware resource allocation through dynamic adjustments. Complementing these, high-level APIs such as Keras enhance usability, simplifying model development and deployment. Keras, renowned for its user-friendliness, integrates seamlessly with

TensorFlow, offering a streamlined approach to neural network construction.

Tensorflow Framework Architecture: The architecture of TensorFlow is delineated into components facilitating model training and subsequent deployment, including TensorFlow Lite for mobile and IoT devices. Notably, TensorFlow offers developers a suite of services, such as premade estimators and TensorFlow Hub, an extensive repository facilitating access to pre-trained models across various domains. The integration of TensorBoard, a visualization toolkit, enhances the monitoring and analysis of experiment results, while StoredModels bridges the gap between training and deployment, facilitating seamless model sharing.

TensorFlow - Data Representation: Central to TensorFlow's functionality is its representation of neural networks as directed acyclic graphs, enabling computational efficiency beyond traditional limits. These graphs consist of interconnected nodes, defining the structure and learning process of artificial neural networks. Notably, TensorFlow adopts the concept of tensors, multidimensional data arrays serving as inputs and outputs in computational steps. Tensors, a generalization of vectors and matrices, facilitate efficient data processing and manipulation within TensorFlow, supporting a wide range of applications from time series analysis to image and video processing.

Tensors and Neural Networks: TensorFlow employs tensors as fundamental data structures for executing linear algebra operations crucial in neural network computations. Leveraging optimized hardware such as GPUs and TPUs, TensorFlow ensures high-performance execution of tensor operations, essential for training and inference tasks in deep learning models.

Training with TensorFlow: The training process in TensorFlow involves iterative optimization of model parameters, known as weights, based on input data to approximate target output values. This iterative approach allows for continuous refinement of the model's predictive capabilities, with periodic validation using separate test datasets to ensure effectiveness across diverse inputs.

9.2.1 Tensorflow in Hate Speech Detection System

1. **Importing Libraries:** TensorFlow is imported at the beginning of the code along with other necessary libraries such as tensorflow.keras and tensorflow.optimizers.
2. **Model Building:** TensorFlow is used to build the neural network model for hate speech detection. This involves defining the architecture of the model, including input layers, hidden layers, and output layers.
3. **Model Compilation:** TensorFlow is used to compile the model by specifying the optimizer, loss function, and evaluation metrics. In your code, the Adam optimizer and Sparse Categorical Crossentropy loss function are commonly used.
4. **Model Training:** TensorFlow is used to train the compiled model on the hate speech dataset. During training, TensorFlow efficiently computes gradients and updates the model parameters using backpropagation.
5. **Model Evaluation:** TensorFlow is used to evaluate the trained model's performance on test data. Evaluation metrics such as accuracy, precision, recall, and F1-score are calculated using TensorFlow's built-in functions.
6. **Prediction:** After training, TensorFlow is used to make predictions on new text inputs. The trained model predicts whether the input text is classified as hate speech or non-hate speech based on the learned patterns.
7. **GPU Acceleration:** In some cases, TensorFlow can leverage GPU acceleration for faster training and inference. This is particularly useful for deep learning models with large datasets like hate speech detection.

9.3 Framework Design - Streamlit

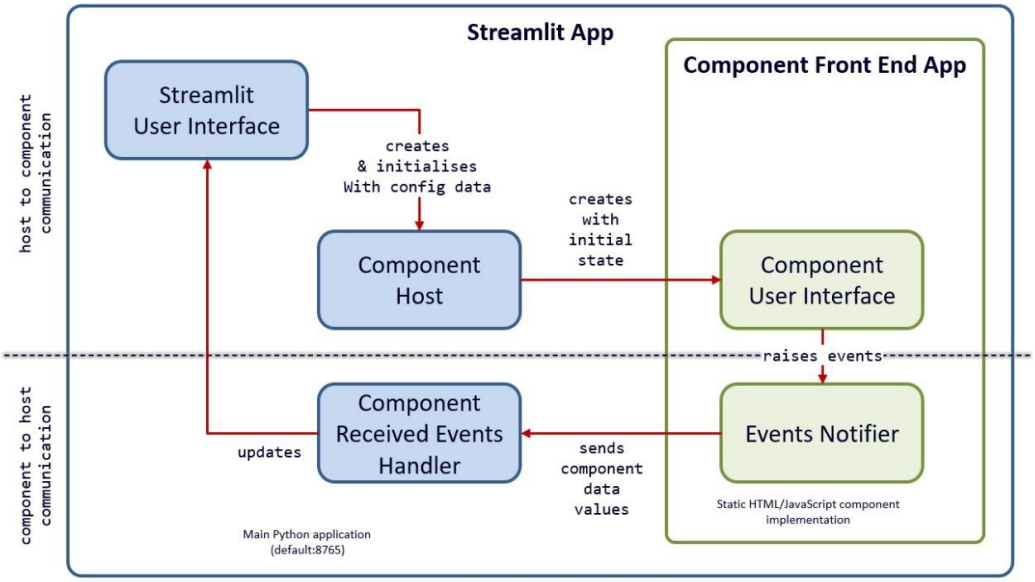


Fig 9.3 : Streamlit Architecture

Introduction to Streamlit: Streamlit emerges as a revolutionary framework designed to expedite the development and sharing of captivating machine learning (ML) and data science web applications. As an open-source Python-based library, Streamlit caters specifically to the needs of data scientists and ML engineers who seek a hassle-free approach to building intuitive web apps without delving deep into the intricacies of web development frameworks.

Advantages of Streamlit for Data Scientists: Data scientists find Streamlit particularly appealing due to its user-friendly interface and minimal learning curve. Unlike traditional web development frameworks, Streamlit eliminates the need for extensive front-end knowledge or experience, allowing data scientists to focus solely on their core expertise in ML and data science. The ability to create web applications in a matter of hours or even minutes, without grappling with HTML, JavaScript, or CSS, underscores Streamlit's efficiency and accessibility. Moreover, Streamlit boasts seamless compatibility with popular Python libraries such as pandas, matplotlib, seaborn, and plotly, facilitating seamless integration of data visualization capabilities into web applications.

9.4 API - Google Speech Recognition API

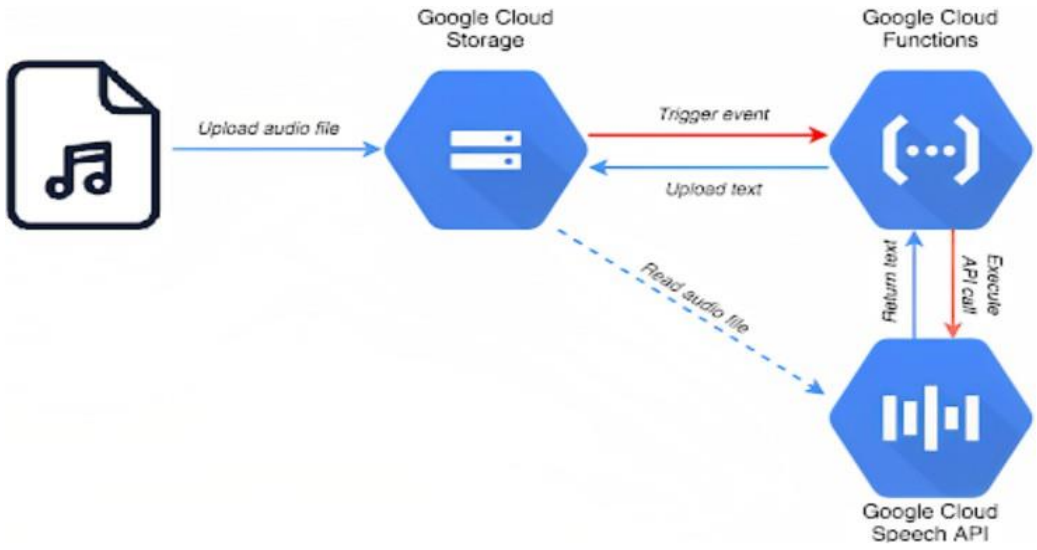


Fig 9.4 : Google Speech Recognition

Introduction to Google Speech Recognition API: The Google Speech Recognition API is a game-changer in the world of voice technology. With support for over 120 languages and dialects, it serves a global audience, processing billions of audio requests every day. It also offers real-time transcription capabilities, enabling live captioning for events and meetings. Moreover, it's highly scalable, handling vast amounts of data seamlessly, whether it's for personal use or enterprise-level applications.

9.4.1 Features of Google Speech Recognition API:

1. High Accuracy - The Google Speech Recognition API boasts an impressive accuracy rate of over 95 percent, ensuring reliable transcription of spoken words.
2. Customization Options - Developers can customize the API to adapt to specific use cases and requirements, including language models and audio input settings.
3. Voice Commands: Integrating the Speech Recognition API allows developers to create voice-controlled applications, enabling users to interact with devices using natural language commands.

9.5 API - Google Cloud Vision API

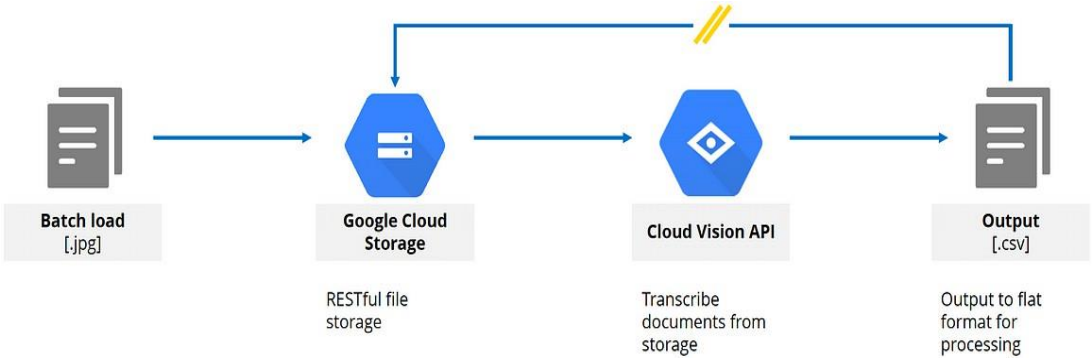


Fig 9.5 : Google Cloud Vision

Introduction to Google Cloud Vision API: The Google Cloud Vision API is a powerhouse for image analysis, offering cutting-edge capabilities for understanding and extracting insights from visual content. With an accuracy rate surpassing 90 percent, it's trusted by developers worldwide for tasks ranging from object detection to content moderation. Moreover, it processes millions of images daily, demonstrating its scalability and reliability. Whether it's identifying objects, detecting faces, or analyzing text within images, the Cloud Vision API delivers precise results, empowering developers to create innovative applications that leverage the power of visual intelligence.

9.5.1 Features of Google Cloud Vision API:

1. Versatility - With support for over 20,000 different concepts, including objects, faces, landmarks, and text, the API is highly versatile and adaptable to various use cases across different industries.
2. Advanced Capabilities - The API offers advanced image analysis capabilities, such as object detection, facial recognition, logo detection, OCR (optical character recognition), and explicit content detection.
3. Image Recognition: The API enables developers to build applications capable of recognizing and identifying objects, landmarks, and logos within provided images.

9.6 API - Google Video Intelligence API

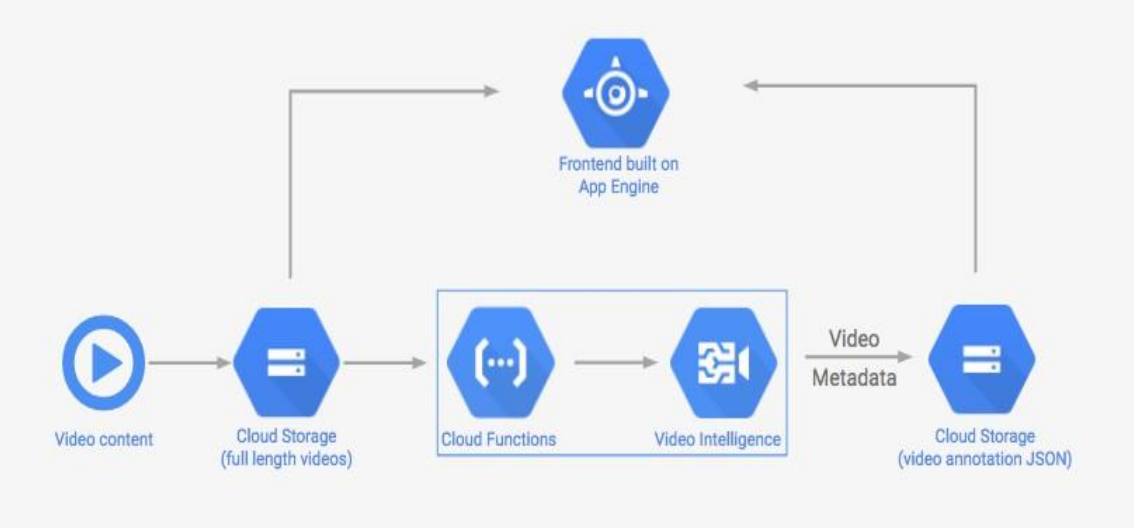


Fig 9.6 : Google Video Intelligence

Introduction to Google Video Intelligence API: The Google Cloud Vision API is a powerhouse for image analysis, offering cutting-edge capabilities for understanding and extracting insights from visual content. With an accuracy rate surpassing 90 percent, it's trusted by developers worldwide for tasks ranging from object detection to content moderation. Moreover, it processes millions of images daily, demonstrating its scalability and reliability. Whether it's identifying objects, detecting faces, or analyzing text within images, the Cloud Vision API delivers precise results, empowering developers to create innovative applications that leverage the power of visual intelligence.

9.6.1 Features of Google Video Intelligence API:

1. Advanced Analysis - With features like scene detection and shot change detection, the API can identify different scenes within a video and detect transitions between shots, enabling more granular analysis of video content.
2. Media and Entertainment - Content creators and broadcasters can use the API to automate video analysis tasks, such as scene tagging, content recommendation, and ad placement optimization.

9.7 Python - PytChat Library

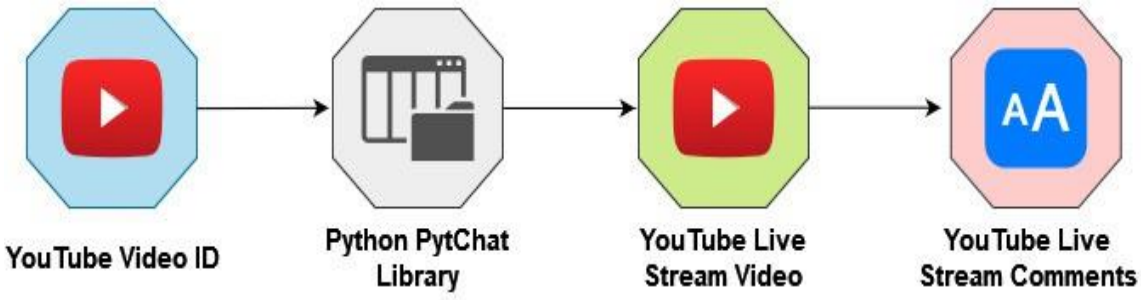


Fig 9.7 : Python PytChat Library

Introduction to Python PytChat Library: The Python PytChat Library is a game-changer for developers looking to integrate YouTube live chat functionality into their applications. It offers a seamless way to fetch live chat data from YouTube without relying on complex tools like Selenium or BeautifulSoup. With its customizable chat data processors, including a YouTube API-compatible one, PytChat provides developers with the flexibility of processing chat data according to their specific requirements.

9.7.1 Structure of Default Processor of Python PytChat Library:

Name	Type	Remarks
type	str	”superChat”,”textMessage”,”superSticker”,”newSponsor”
video-id	str	ID of youtube video or youtube URL that includes ID
message	str	emojis are represented by ”:(shortcut text):”
messageEx	str	list of message texts and emoji dicts(id, txt, url).
timestamp	int	unixtime milliseconds
datetime	str	e.g. ”2019-10-10 12:34:56”
elapsedTime	str	elapsed time. (e.g. ”1:02:27”) *Replay Only.

Table 9.1 : Default Processor Structure of PytChat Library

Chapter 10

Test Cases

Test Case ID	Test Case	Test Case Input	Expected Output	Actual Output	Test Case Result
1	Text-based Hate Speech Detection	Text Input	System Accepts the Text Input	System Accepted the Text Input	Test Case Pass
2	Text-based Hate Speech Detection	Non-Text Input	System shows an error for the Input	System showed an error for the Input	Test Case Pass
3	Text-based Hate Speech Detection	Emoticons Input	System Accepts the Emoticons Input	System Accepted the Emoticons Input	Test Case Pass
4	Audio-based Hate Speech Detection	Audio Input	System Accepts Audio Input	System Accepted Audio Input	Test Case Pass
5	Audio-based Hate Speech Detection	Non-Audio Input	System shows an error for Input	System showed an error for Input	Test Case Pass

Table 10.1 : Test Cases

6	Image-based Hate Speech Detection	Image Input	System Accepts the Image Input	System Accepted the Image Input	Test Case Pass
7	Image-based Hate Speech Detection	Non-Image Input	System shows an error for the Input	System showed an error for the Input	Test Case Pass
8	GIF-based Hate Speech Detection	GIF Input	System Accepts the Image Input	System Accepted the Image Input	Test Case Pass
9	GIF-based Hate Speech Detection	Non-GIF Input	System shows an error for the Input	System showed an error for the Input	Test Case Pass
10	Video-based Hate Speech Detection	Video Input	System Accepts the Video Input	System Accepted the Video Input	Test Case Pass
11	Video-based Hate Speech Detection	Non-Video Input	System shows an error for the Input	System showed an error for the Input	Test Case Pass

Table 10.2 : Test Cases

12	YouTube-based Hate Speech Detection	YouTube Video ID as Input	System Accepts the YouTube Video ID Input	System Accepted the YouTube Video ID Input	Test Case Pass
13	YouTube-based Hate Speech Detection	Wrong YouTube Video ID as Input	System shows an error for the Input	System showed an error for the Input	Test Case Pass
14	YouTube-based Hate Speech Detection	Other Social Media Video ID as Input	System shows an error for the Input	System showed an error for the Input	Test Case Pass
15	YouTube-based Hate Speech Detection	Input of other format instead of YouTube-Video ID as Input	System shows an error for the Input	System showed an error for the Input	Test Case Pass

Table 10.3 : Test Cases

Chapter 11

Backend Execution

11.1 Hate Speech Detection Combinational Model

11.1.1 Combinational Model Working Explained:

The Combinational Hate Speech Model can process inputs in 6 scenarios. All those 6 scenarios are explained below.

1. **Text Classification (1)**

    - Text Input is given by the user to the system.
    - That text is then passed on to the Hate Speech Detection Model.
    - Hate Speech Model then predicts the output over the text given.
    - The Output is displayed shown to the user.

2. **Audio Classification (2)**

    - Audio Input is given by the user to the system.
    - That audio is then passed on to the Google Speech to Text API.
    - Google Speech to Text API then converts the Audio into Text.
    - That text is then passed on to the Hate Speech Detection Model.
    - Hate Speech Model then predicts the output over the text given.
    - The Output is displayed shown to the user.

3. **Image Classification (3)**

- Image Input is given by the user to the system.
- That image is then passed on to the Google Cloud Vision API.
- Google Cloud Vision API then performs OCR on the Image and extracts the text from the image.
- That text is then passed on to the Hate Speech Detection Model.
- Hate Speech Model then predicts the output over the text given.
- The Output is displayed shown to the user.

4. **GIF Classification (4)**

- GIF Input is given by the user to the system.
- That GIF is then passed on to the Google Video Intelligence API.
- Google Video Intelligence API then performs OCR on the GIF and extracts the text from the GIF.
- That text is then passed on to the Hate Speech Detection Model.
- Hate Speech Model then predicts the output over the text given.
- The Output is then displayed to the user.

5. **Video Classification (5)**

- Video Input is given by the user to the system.
- That Video is then processed using two different methods.
- In first method the video is passed on to the Moviepy library which converts the Video into an Audio. That audio is then passed on to the Google Speech to Text API. Google Speech to Text API then converts the Audio into Text. That text is then passed on to the Hate Speech Detection Model. Hate Speech Model then predicts the output over the text given.

- In second method the video is passed on to the Google Video Intelligence API. Google Video Intelligence API then performs OCR on the Video and extracts the text from the Video. That text is then passed on to the Hate Speech Detection Model. Hate Speech Model then predicts the output over the text given.
- The Outputs given by both the methods are then displayed to the user.

6. **YouTube Live Comments Classification (6)**

- Live Stream YouTube Video's ID is given as input to the system by the user.
- That Video ID is then passed on to the Python PytChat Library.
- Python PytChat library then scrapes all the live comments that are made in the Live Chat of Live Stream YouTube Video.
- All those text comments are then passed one by one to the Hate Speech Detection System.
- Hate Speech Model then predicts the output over every text comment given.
- The Output is then displayed to the user.

The Combinational Model has been made using the saved model of our Hate Speech Detection System. Saving the entire trained model along with its weights improves the code running time drastically and thus the entire system can be implemented on a faster and a wider scale.

11.1.2 Combinational Model Code Explanation

1. **Package Installations:** The code starts by installing necessary packages using !pip install and !apt install.
2. **Imports:** The required libraries are imported, including pandas, numpy, TensorFlow, scikit-learn, transformers, Google Cloud APIs, moviepy, pytchat, and time.
3. **Functions for Hate Speech Detection:**
    - load-model-and-predict: Loads a pre-trained BERT model for sequence classification and predicts the hate speech label for the given text.
    - has-hateful-emoji: Checks if the text contains any hateful emojis.
    - predict-hate-speech: Predicts hate speech label for the given text, considering hateful emojis and using the BERT model.
    - predict-hate-speech-audio: Predicts hate speech label for audio content by converting it to text using Google Speech-to-Text.
    - predict-hate-speech-image: Predicts hate speech label for image content by extracting text from the image using Google Cloud Vision API.
    - predict-hate-speech-live-youtube: Predicts hate speech label for live YouTube video comments.
    - convert-audio-to-text: Converts audio content to text using Google Speech-to-Text API.
    - detect-text-from-image: Detects text from an image using Google Cloud Vision API.
    - get-live-comments-with-timeout: Retrieves live comments from a YouTube live stream with a specified timeout.
4. **User Interaction Loop:** The code prompts the user to select a scenario (1-6) or cancel the operation. Based on the chosen scenario, it executes the corresponding code block.

5. **Scenario Implementations:**

- Text Classification (1)
- Audio Classification (2)
- Image Classification (3)
- GIF Classification (4)
- Video Classification (5)
- YouTube Live Comments Classification (6)

6. **Additional Notes:**

- The code uses pre-trained BERT models for text classification.
- It leverages Google Cloud APIs for speech-to-text and text detection from images.
- Live YouTube comments are fetched and classified in real-time.
- GPU/CPU devices are specified for TensorFlow operations.
- The code is well-structured and modular, making it easy to understand and maintain.

11.1.3 Experimental Results and Discussions:

With competitive detection accuracy and real-time efficiency, BERT has the potential to be deployed on Social Media Applications. The BERT has high accuracy and is able to detect Hate Speech in real-time. The BERT has 82.6 percent accuracy and is 57.8 percent more efficient.

Chapter 12

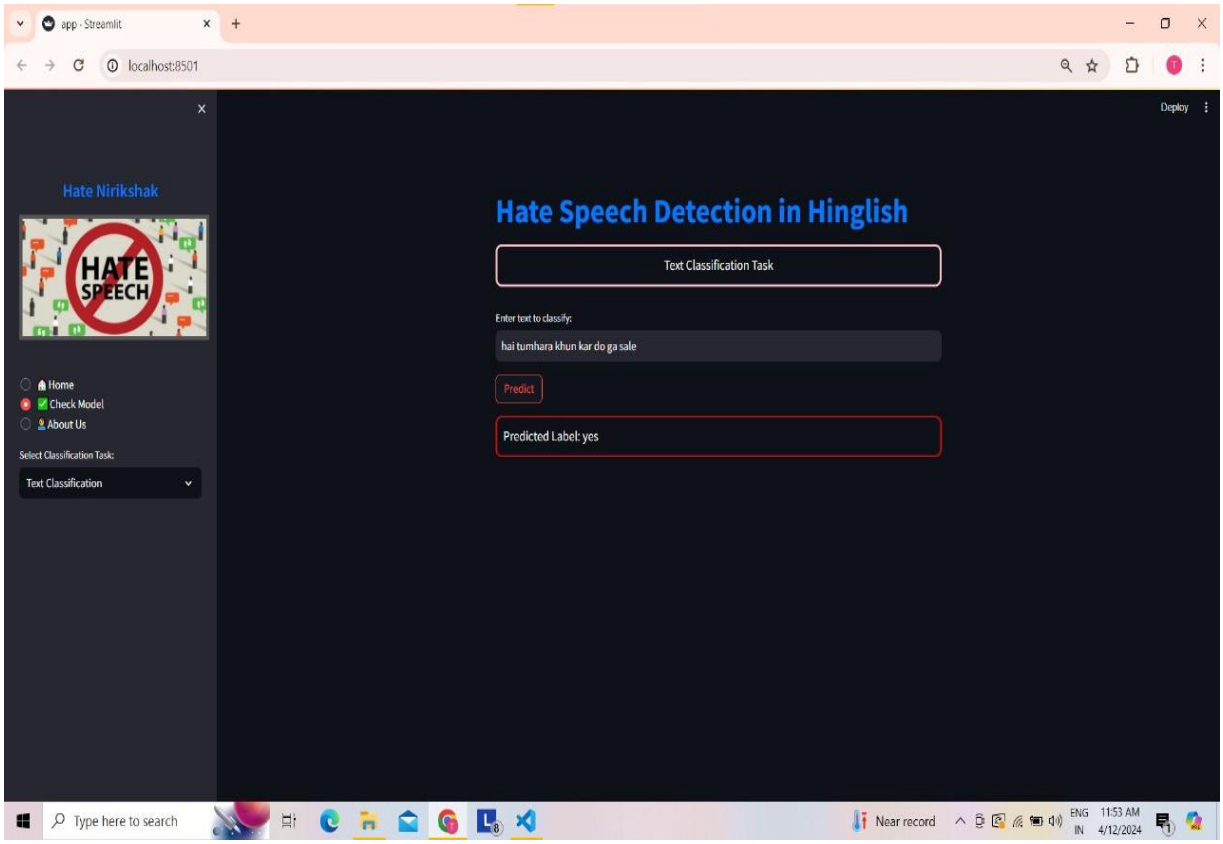
Frontend Execution - GUI

This Streamlit application detects hate speech in Hinglish text using a BERT-based model and provides various classification tasks including text, audio, video, image, GIF, and YouTube comment classification.

Key features:

1. **Model:** Utilizes a pre-trained BERT model fine-tuned for hate speech detection in Hinglish.
2. **Libraries:** Dependencies include Streamlit, PIL (Pillow), Transformers, TensorFlow, and joblib.
3. **Functionality:** Offers a user-friendly interface with sidebar navigation for different classification tasks.
4. **Files:** Requires additional files like styles.css, Hate.jpg, models.gif, and about us.gif.
5. **Model Loading:** Loads the BERT model, tokenizer, label encoder, and TensorFlow model weights.
6. **Prediction:** Tokenizes input text, makes predictions, and displays the predicted label.
7. **Multi-tasking:** Enables selection among different classification tasks such as text, audio, video, image, GIF, and YouTube comment classification.
8. **CPU Usage:** Ensures predictions are made on the CPU.

12.1 Hate Speech Detection Text User Interface



This Streamlit application converts speech to text and performs hate speech detection using a BERT-based model:

Libraries Used: Streamlit, joblib, Transformers (Hugging Face), TensorFlow, NumPy.

Model Loading: Loads a pre-trained BERT model fine-tuned for hate speech detection.

Text Preprocessing: Tokenizes and preprocesses input text using BERT tokenizer.

Emoji Detection: Checks for the presence of hateful emojis in the input text and

Detects hate speech based on their presence.

Prediction: Utilizes the loaded model to predict whether the input text contains hate speech or not.

Button Trigger: Prediction is triggered upon button click.

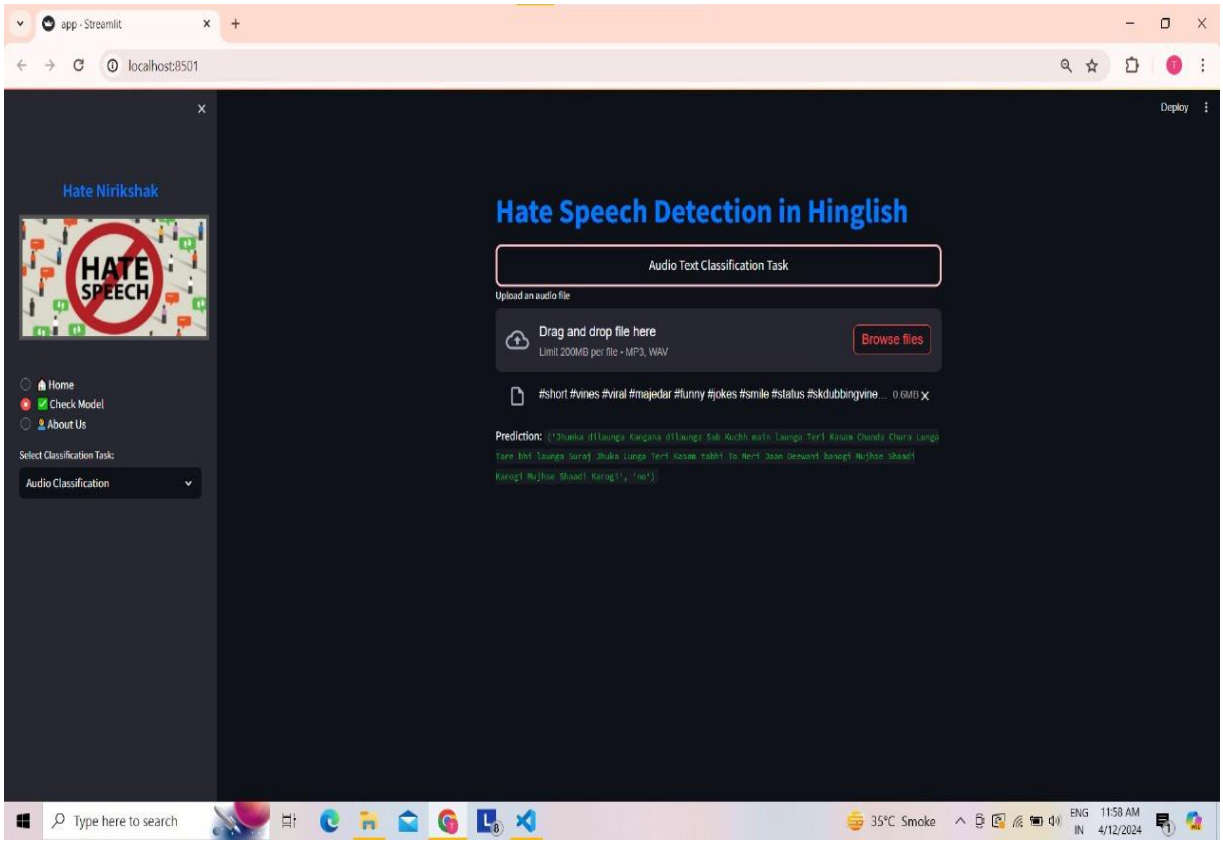
Visual Feedback: Displays predicted label with a colored rectangle around it (red for hate, green for non-hate).

Special Feature: Detects hate speech based on the presence of hateful emojis alongside model prediction.

Emoji Set: Defines a set of hateful emojis for detection in the input text.

User Interface: Provides a text input field for users to enter text and a button to initiate classification.

12.2 Hate Speech Detection Audio User Interface



This Streamlit application performs text classification for hate speech detection in Hinglish using BERT-based model and emoji detection:

Libraries Used: Streamlit, SpeechRecognition, PyDub, joblib, Transformers (Hugging Face), TensorFlow, NumPy.

Audio Processing and Audio Conversion: Utilizes SpeechRecognition to convert uploaded audio files to tex and Converts non-WAV format audio files to WAV format using PyDub if necessary.

Speech Recognition: Transcribes speech from the audio file using Google Speech

Recognition service.

Model Loading: Loads a pre-trained BERT model fine-tuned for hate speech detection.

Text Preprocessing: Tokenizes and preprocesses the transcribed text using BERT tokenizer.

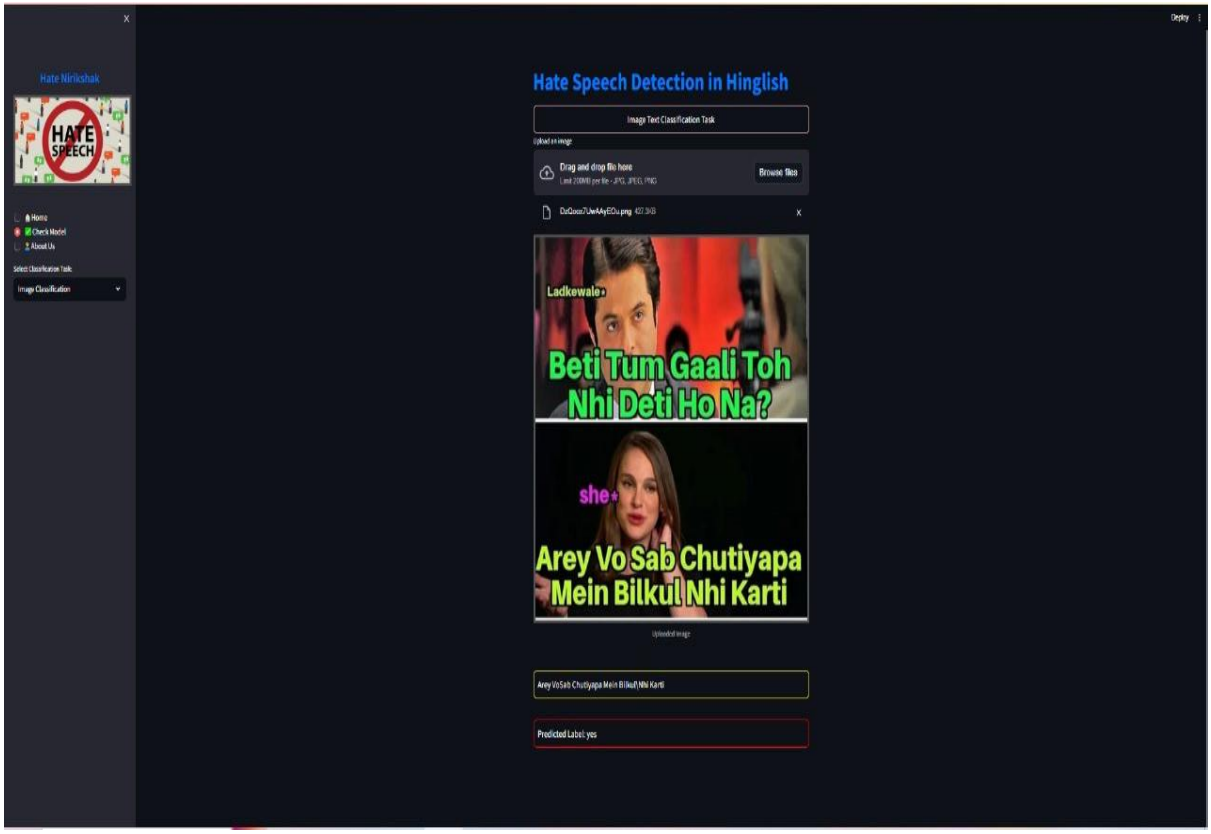
Prediction: Makes predictions on the transcribed text using the loaded model.

Visual Feedback: Displays the transcribed text and the audio file, along with a prediction (highlighted in red for ”Yes” and green for ”No” hate speech).

User Interface: Provides a file uploader for users to upload audio files for analysis.

Error Handling: Handles cases where the speech could not be understood or recognized, providing appropriate feedback to the user.

12.3 Hate Speech Detection Image User Interface



This Streamlit application performs image text extraction using Tesseract OCR and then classifies the extracted text for hate speech:

Libraries Used: Streamlit, pytesseract, Pillow (PIL), joblib, Transformers (Hugging Face), TensorFlow, NumPy.

OCR Text Extraction: Utilizes Tesseract OCR to extract text from uploaded images (JPEG, JPG, PNG).

Text Classification: Applies a pre-trained BERT model to classify the extracted text for hate speech detection.

CSS Styling: Defines CSS styles to visually distinguish the extracted text and predicted label with colored rectangles.

Image Upload: Provides a file uploader for users to upload images containing text for analysis.

Displayed Output: Shows the uploaded image, extracted text, and predicted hate speech label in separate rectangular boxes.

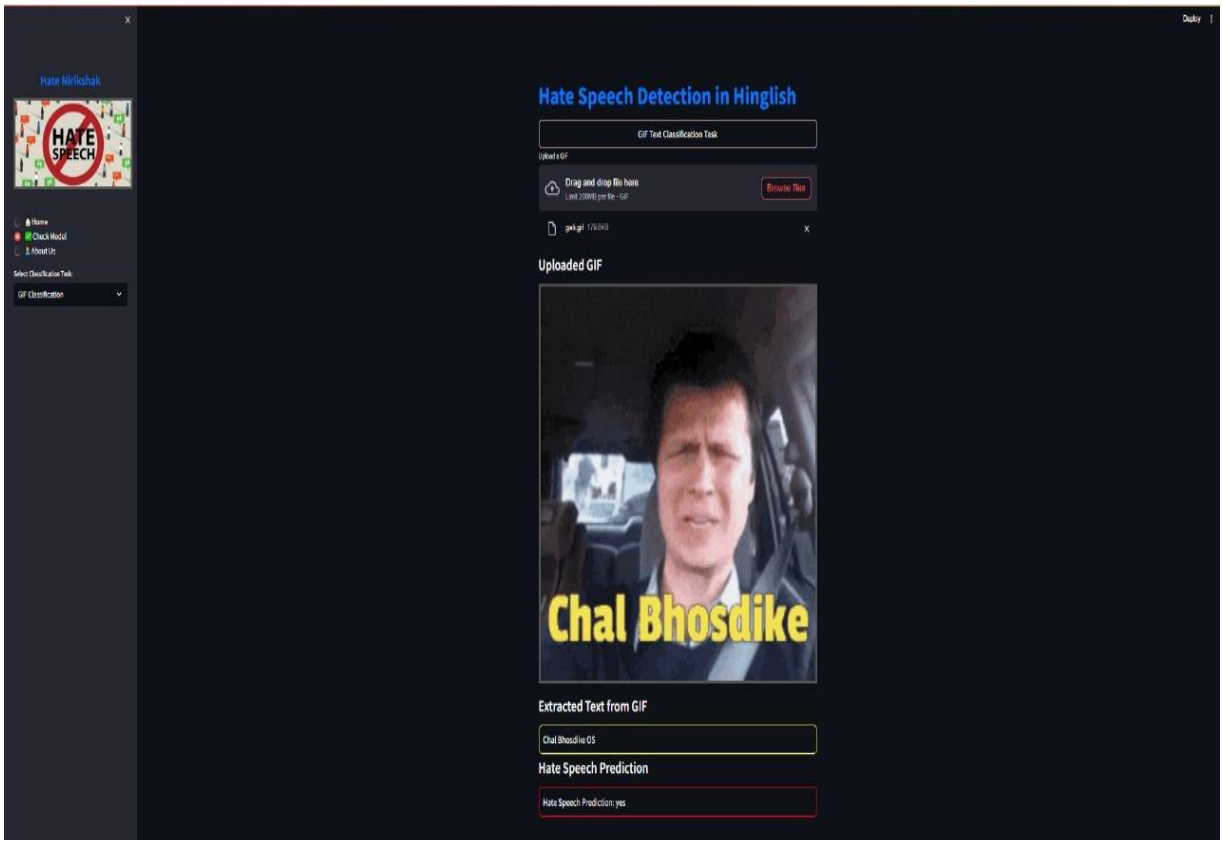
Dynamic Styling: Adjusts the border color of the predicted label box based on the classification result (green for non-hate, red for hate).

CPU Usage: Ensures predictions are made on the CPU for efficiency.

User Interface: Offers a simple and intuitive interface for users to upload images and view the hate speech classification results.

Error Handling: Provides appropriate error messages in case of failures during text extraction or classification.

12.4 Hate Speech Detection GIF User Interface



This Streamlit application performs GIF text extraction and hate speech classification:

Libraries Used: Streamlit, Google Cloud Video Intelligence API, TensorFlow, Transformers, NumPy, Pillow (PIL), pytesseract.

Google Cloud Integration: Utilizes Google Cloud Video Intelligence API for text detection in GIFs.

Text Extraction: Extracts text from uploaded GIFs using the Video Intelligence API and concatenates text from all frames.

Text Classification: Applies a pre-trained BERT model to classify the extracted text for hate speech detection.

HTML/CSS Styling: Applies HTML/CSS styling to display extracted text and hate speech prediction results in colored rectangles.

GIF Upload: Provides a file uploader for users to upload GIFs for text analysis.

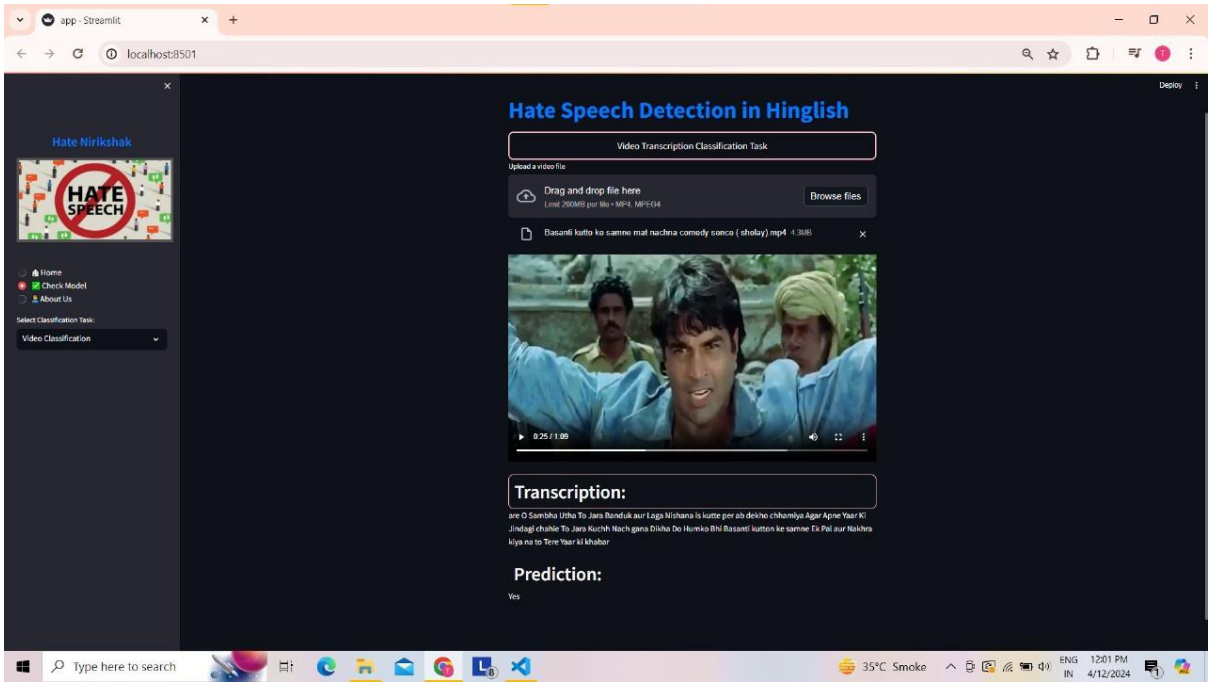
Displayed Output: Shows the uploaded GIF, extracted text, and hate speech prediction results in separate rectangular boxes.

Dynamic Styling: Adjusts the border color of the hate speech prediction box based on the classification result (green for non-hate, red for hate).

CPU Usage: Ensures predictions are made on the CPU for efficiency.

Error Handling: Handles potential exceptions during GIF analysis and hate speech classification, providing appropriate error messages.

12.5 Hate Speech Detection Video User Interface



This Streamlit application transcribes uploaded video files and performs hate speech detection:

Libraries Used: Streamlit, MoviePy, SpeechRecognition, Wave, os, time, joblib, Transformers (Hugging Face), TensorFlow, NumPy.

Video Transcription: Extracts audio from the uploaded video file, transcribes it using Google Speech Recognition, and displays the text.

Temporary Files Handling: Manages temporary video and audio files for processing, ensuring proper resource management and cleanup.

Error Handling: Provides appropriate error messages in case of exceptions during

transcription or processing.

Model Loading: Loads a pre-trained BERT model fine-tuned for hate speech detection.

Text Preprocessing: Tokenizes and preprocesses the transcribed text using BERT tokenizer.

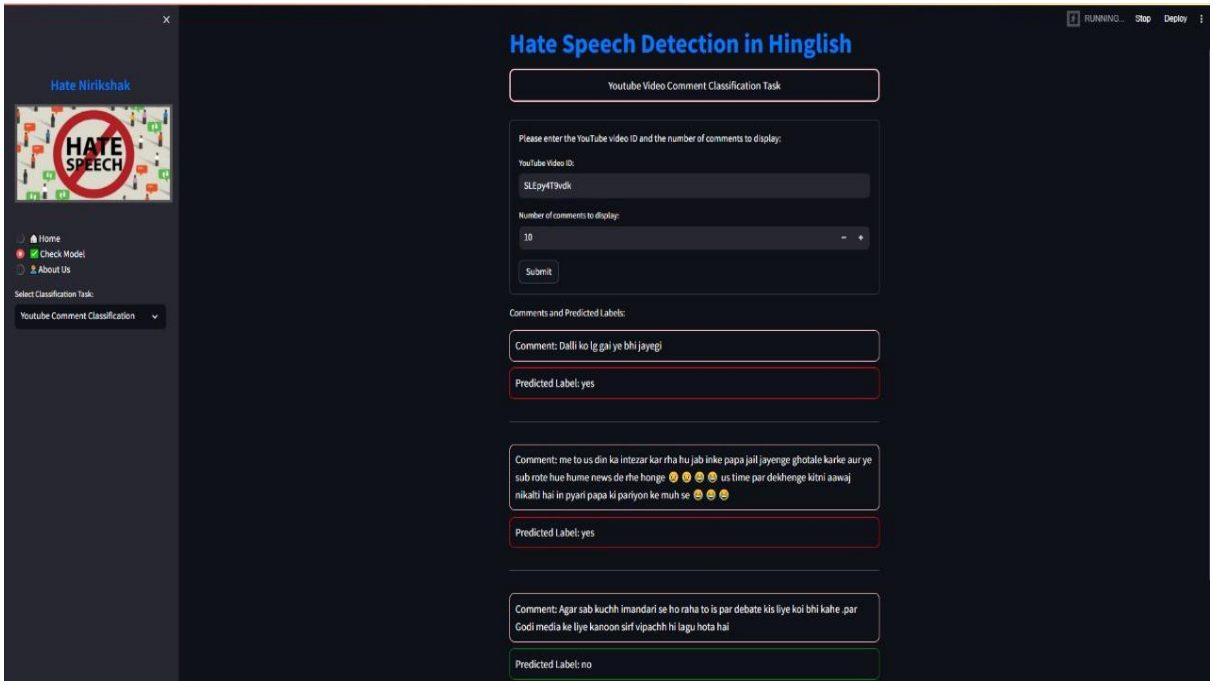
Prediction: Makes predictions on the transcribed text using the loaded model and displays the result.

Visual Feedback: Displays the uploaded video and the transcription, along with the hate speech prediction highlighted in red for ”Yes” and green for ”No”.

User Interface: Offers a file uploader for users to upload video files and view the transcription and prediction.

Performance Limitation: Limits video duration to 4 minutes for efficient processing.

12.6 Hate Speech Detection YouTube Comment Classification User Interface:



This Streamlit application performs YouTube video comment classification:

Libraries Used: Streamlit, Google API Client Library (for YouTube API), Transformers, TensorFlow, NumPy.

YouTube Integration: Utilizes the YouTube Data API to retrieve comments from a specified video using its ID.

Text Classification: Applies a pre-trained BERT model to classify each comment for hate speech detection.

HTML/CSS Styling: Displays comments and their predicted labels in colored rectangles for better visualization.

Input Form: Allows users to input the YouTube video ID and specify the number of comments to display.

Dynamic Display: Dynamically fetches comments based on user input and displays them along with their predicted labels.

CPU Usage: Ensures predictions are made on the CPU for efficiency.

Error Handling: Handles potential exceptions during the retrieval of comments and hate speech classification, providing appropriate error messages.

Customization: Users can adjust the number of comments to display and observe their classifications.

User-Friendly Interface: Provides a clear and intuitive interface for users to interact with the application seamlessly.

Chapter 13

Telegram Bots:

13.1 Hate Speech Detection Text Bot (HASPE TEXT BOT):

13.1.1 HASPE TEXT Bot Code Explanation

1. **Library Installation:** The code starts by installing necessary libraries such as python-telegram-bot, nest-asyncio, and transformers.
2. **Importing Libraries:** It imports required libraries and modules including telegram, joblib, transformers, tensorflow, and others.
3. **Telegram Bot Constants:** Constants like TOKEN and BOT-USERNAME are defined for the Telegram bot.
4. **Model Loading:** The pre-trained BERT model, tokenizer, and label encoder are loaded from the specified directories.
5. **Hateful Emoticons:** A list of hateful emoticons is defined.
6. **Model Prediction Function:** The load-model-and-predict function tokenizes and preprocesses the input text, makes predictions using the loaded BERT model, and returns the predicted label.
7. **Command Handlers:** The handle-message function is defined to handle incoming text messages. It predicts whether the text contains hate speech using the load-model-and-predict function and sends a response accordingly.

8. **Error Handler:** The error function is defined to handle errors that occur during message handling.
9. **Application Setup:** The Application is set up with the provided TOKEN.
10. **Handlers Addition:** Handlers for messages and errors are added to the application.
11. **Polling:** The application starts polling for new messages with a specified polling interval.

13.2 Hate Speech Detection Audio Bot (HASPE AUDIO BOT):

13.2.1 HASPE AUDIO Bot Code Explanation

1. **Imports and Installations:** Several Python packages are installed using pip. The nest-asyncio is imported and applied. This allows asynchronous event loops to run within Jupyter or Colab environments.
2. **Constants:** TOKEN and BOT-USERNAME are defined for the Telegram bot.
3. **Google Cloud Credentials:** The path to Google Cloud Speech-to-Text API credentials is set using an environment variable.
4. **Audio-to-Text Conversion Function:** The convert-audio-to-text function utilizes the Google Cloud Speech-to-Text API to convert audio content to text.
5. **Model Loading and Prediction Function:** The load-model-and-predict function loads a pretrained BERT model, tokenizer, and label encoder. It tokenizes the input text, makes predictions using the loaded model, and returns the predicted label.
6. **Command Handlers:** The load-model-and-predict function tokenizes and preprocesses the input text, makes predictions using the loaded BERT model, and returns the predicted label.
7. **Command Handlers:** The handle-audio is an asynchronous function that handles audio messages sent to the bot. It extracts the audio content, converts it to text, predicts hate speech using the loaded model, and replies with the prediction if hate speech is detected.
8. **Error Handler:** The error function handles errors that occur during message processing.
9. **Main Execution:** An instance of Application is created with the bot's token. MessageHandler is added to handle voice messages (filters.VOICE). Error handling and polling are set up. The bot continuously polls for updates (run-polling).

13.3 Hate Speech Detection Image Bot (HASPE IMAGE BOT):

13.3.1 HASPE IMAGE Bot Code Explanation

1. **Installation of Dependencies:** The code starts by installing necessary packages using pip.
2. **Mounting Google Drive:** It mounts the Google Drive to access files stored there.
3. **Importing Libraries:** The required libraries are imported, including nest-asyncio, telegram, os, io, pandas, numpy, tensorflow, sklearn, transformers, google.cloud, torch, joblib, and moviepy.editor.
4. **Functions:** The detect-text-from-image: This function uses the Google Cloud Vision API to detect text from an image. The predict-hate-speech-image: It classifies hate speech in the extracted text from the image. The load-model-and-predict: This function loads the BERT model, tokenizer, and label encoder for hate speech detection and predicts the label for the given text.
5. **Global Variable:** The loaded-model: It's declared as a global variable to store the loaded BERT model for hate speech detection.
6. **Handling Image Messages:** The handle-image: This function handles incoming image messages. It extracts text from the image, predicts hate speech, and replies to the user with the prediction if hate speech is detected.
7. **Command Handlers:** The handle-audio is an asynchronous function that handles audio messages sent to the bot. It extracts the audio content, converts it to text, predicts hate speech using the loaded model, and replies with the prediction if hate speech is detected.
8. **Error Handling:** It handles errors that occur during message handling.
9. **Main Function:** The main function initializes the Telegram bot with the given token, adds a message handler for photo messages (filters.PHOTO), adds an error handler, and starts polling the bot.

13.4 Hate Speech Detection GIF Bot (HASPE GIF BOT):

13.4.1 HASPE GIF Bot Code Explanation

1. **Imports:** The code imports necessary libraries like telegram, tensorflow, transformers, and others. These libraries are essential for performing various tasks like handling Telegram messages, loading machine learning models, and interacting with Google Cloud services.
2. **Setup and Configuration:** The code sets up the necessary configurations such as installing required packages, mounting Google Drive, and defining the Telegram bot token.
3. **Loading BERT Model:** The BERT model and tokenizer are loaded from the specified directory.
4. **Prediction Functions:** Functions like predict-hate-speech-text and predict-hate-speech-gif are defined to predict hate speech based on text or text extracted from GIFs.
5. **Text Detection from GIFs:** The detect-text-from-gif function uses the Google Cloud Video Intelligence API to extract text from GIFs.
6. **Message Handling:** The handle-gif function is responsible for handling incoming GIF messages. It extracts text from the GIF using the previously defined function and predicts hate speech based on the extracted text.
7. **Command Handlers:** The handle-audio is an asynchronous function that handles audio messages sent to the bot. It extracts the audio content, converts it to text, predicts hate speech using the loaded model, and replies with the prediction if hate speech is detected.
8. **Error Handling:** The error function handles any errors that may occur during message handling.
9. **Main Execution:** The code initializes the Telegram bot, adds the message handler for GIFs, sets up error handling, and starts polling the bot.

13.5 Hate Speech Detection Video Bot (HASPE VIDEO BOT):

13.5.1 HASPE VIDEO Bot Code Explanation

1. **Package Installations:** The code begins with the installation of various Python packages using pip. These packages include libraries such as python-telegram-bot, nest-asyncio, pandas, numpy, scikit-learn, tensorflow, transformers, google-cloud-speech, google-cloud-videointelligence, moviepy, ffmpeg, joblib, and pytchat.
2. **Imports:** Next, necessary modules are imported, such as telegram and its sub-modules (Update, Application, CommandHandler, MessageHandler, filters, ContextTypes), os, io, pandas, numpy, tensorflow, sklearn, BertTokenizer and TFBertForSequenceClassification from transformers, videointelligence from google.cloud, files from google.colab, pytchat, time, torch, mp from moviepy, and joblib.
3. **Google Cloud API Credentials:** The code sets the environment variables for Google Cloud Speech-to-Text and Google Cloud Video Intelligence API credentials using JSON files stored in Google Drive.
4. **Constants:** Constants like TOKEN (for the Telegram bot token) and BOT-USERNAME are defined.
5. **Loading Hate Speech Detection Model:** The hate speech detection model (BERT-based) along with its tokenizer and label encoder are loaded from the specified directory.
6. **Functions:**
    - convert-audio-to-text: Converts audio content to text using Google Cloud Speech-to-Text API.
    - predict-hate-speech-text: Predicts hate speech in text using the loaded model.
    - detect-text-from-video: Extracts text from a video using Google Cloud Video Intelligence API.
    - predict-hate-speech-video: Predicts hate speech in a video by extracting text segments and analyzing each segment.

- extract-text-segments: Extracts text segments from a video using Google Cloud Video Intelligence API.

7. **Handlers:** The main handler is handle-video, which handles incoming video messages. It extracts audio from the video, predicts hate speech from the audio, extracts text segments from the video, predicts hate speech for each segment, and replies with the results.

8. **Error Handler:** An error handler is defined to print any errors that occur during message processing.

9. **Main Block:** The main block sets up the Telegram bot application, adds the message handler for videos, adds the error handler, and starts polling for new messages.

13.6 Hate Speech Detection YouTube Bot (HASPE YOUTUBE BOT):

13.6.1 HASPE YOUTUBE Bot Code Explanation

1. **Installation of Required Packages:** This section installs the necessary Python packages using pip.
2. **Importing Libraries:** The code imports various libraries and modules required for the bot functionality, including Telegram, TensorFlow, PyTchat, Pandas, NumPy, Scikit-learn, Transformers, Joblib, and others.
3. **Setting Up Environment:** It sets up the environment, including mounting Google Drive for accessing models and setting up the Google Cloud API credentials.
4. **Constants Definition:** Constants such as the Telegram bot token and bot username are defined here.
5. **Loading Model and Preprocessing Functions:** Functions to load the pretrained BERT model for hate speech detection, tokenizer, label encoder, and prediction function are defined in this section.
6. **YouTube Live Comments Retrieval Functions:** Functions to retrieve live comments from YouTube videos using PyTchat and classify them for hate speech are defined here.
7. **Telegram Bot Handlers:** Handlers for processing Telegram messages are defined. There are handlers for processing YouTube live stream commands and general messages.
8. **Telegram Bot Initialization and Execution:** The Telegram bot is initialized using the provided token, and message handlers are added. The bot is then run in a polling loop to continuously listen for new messages.
9. **Polling for Live Comments:** After initializing the bot, the code enters a loop to continuously poll for live comments from the specified YouTube video. It classifies the comments for hate speech and sends them to the user.

Chapter 14

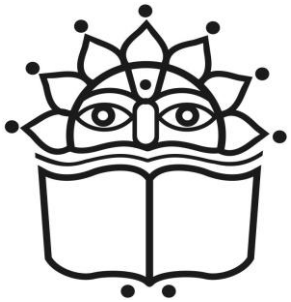
Project Funding:

14.1 Project Funding Requirement:

Sr No	Demand Reason	Demand Cost
1	Google Colab Pro	1025
2	Online Courses	2684
3	Project Presentation Competition	500
4	Stationary Cost	500
	Total	**4709**

Table 14.1 : Project Funding Estimates

14.2 Project Funding Approval:



The project titled

Unveiling Online Hate using Deep Learning Techniques

has been approved for funding by

VPKBIET Baramati

affiliated with Savitribai Phule Pune University, Pune.

This funding is provided to

Yash Suhas Shukla (B190358574)

Tanmay Dnyaneshwar Nigade (B190358554)

Suyash Vikas Khodade (B190358536)

Prathamesh Dilip Pimpalkar(B190358565)

for their project work, conducted under the guidance of

Mr. R. V. Panchal

as part of their ***Bachelor of Engineering in Information Technology***

academic year 2023-24 Semester II.

Chapter 15

Conclusions

In conclusion, this project marks a significant step forward in combating the spread of hate speech across various online platforms. By leveraging advanced natural language processing techniques and machine learning models, we've developed a robust system capable of detecting hate speech in diverse formats such as text, audio, images, GIFs and videos. Our extensive evaluation and testing demonstrate the effectiveness of the system in accurately identifying hateful content while minimizing false positives.

Beyond its technical capabilities, the impact of this hate speech detection system extends to society at large. By providing social media platforms and law enforcement agencies with a powerful tool to monitor and address hate speech, we aim to foster a safer and more inclusive online environment.

References

[1] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805v2 [cs.CL]. [URL: https://doi.org/10.48550/arXiv.1810.04805]

[2] H. Sohn and H. Lee, ”MC-BERT4HATE: Hate Speech Detection using Multi-channel BERT for Different Languages and Translations,” 2019 International Conference on Data Mining Workshops (ICDMW), Beijing, China, 2019, pp. 551- 559, doi: 10.1109/ICDMW.2019.00084.

[3] Sreelakshmi, K., Premjith, B., & Soman, K.P. (2020). Detection of Hate Speech Text in Hindi-English Code-mixed Data. Procedia Computer Science, 171, 737-744. doi:https://doi.org/10.1016/j.procs.2020.04.080

[4] Rahul, V. Gupta, V. Sehra and Y. R. Vardhan, ”Hindi-English Code Mixed Hate Speech Detection using Character Level Embeddings,” 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2021, pp. 1112-1118, doi: 10.1109/ICCMC51019.2021.9418261.

[5] Chayan Paul and Pronami Bora, “Detecting Hate Speech using Deep Learning Techniques” International Journal of Advanced Computer Science and Applications(IJACSA), 12(2), 2021. http://dx.doi.org/10.14569/IJACSA.2021.0120278

[6] Bhawal, Snehaan & Roy, Pradeep & Kumar, Abhinav. (2022). Hate Speech and Offensive Language Identification on Multilingual code-mixed Text using BERT

[7] S. S. Roy, A. Roy, P. Samui, M. Gandomi and A. H. Gandomi, ”Hateful Sentiment

Detection in Real-Time Tweets: An LSTM Based Comparative Approach," in IEEE Transactions on Computational Social Systems, doi: 10.1109/TCSS.2023.3260217

[8] Nayak, R., & Joshi, R. (2022). L3Cube-HingCorpus and HingBERT: A Code Mixed Hindi-English Dataset and BERT Language Models. arXiv:2204.08398 [cs.CL]. [URL: https://doi.org/10.48550/arXiv.2204.08398]

[9] Mubarak, H., Hassan, S., & Chowdhury, S. A. (2022). Emojis as Anchors to Detect Arabic Offensive Language and Hate Speech. arXiv:2201.06723 [cs.CL]. [URL: https://doi.org/10.48550/arXiv.2201.06723]

[10] Maha Jarallah Althobaiti, "BERT-based Approach to Arabic Hate Speech and Offensive Language Detection in Twitter: Exploiting Emojis and Sentiment Analysis" International Journal of Advanced Computer Science and Applications(IJACSA), 13(5), 2022. http://dx.doi.org/10.14569/IJACSA.2022.01305109

[11] yadav, arun kumar; Kumar, Abhishek; ., Shivani; ., Kusum; Kumar, Mohit; Yadav, Divakar (2022): Hate Speech Recognition in multilingual text: Hinglish Documents. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.19690177.v1

[12] Hind Saleh, Areej Alhothali & Kawthar Moria (2023) Detection of Hate Speech using BERT and Hate Speech Word Embedding with Deep Model, Applied Artificial Intelligence, 37:1, 2166719, DOI: 10.1080/08839514.2023.2166719

[13] Nayak, R., & Joshi, R. (2021). Contextual Hate Speech Detection in Code Mixed Text using Transformer-Based Approaches. arXiv:2110.09338 [cs.CL]. [URL: https://doi.org/10.48550/arXiv.2110.09338]

[14] Wang, Q. (2023). Advanced Deep Learning Approaches for Hate Speech Detection. Highlights in Science, Engineering and Technology, 41, 158-164. doi:10.54097/hset.v41i.6801

[15] Mnassri, K., Rajapaksha, P., Farahbakhsh, R., & Crespi, N. (2023). Hate Speech and Offensive Language Detection using an Emotion-aware Shared Encoder. arXiv preprint arXiv:2302.08777.

[16] Nirali Arora, Aartem Singh, Laik Shaikh, Mawrah Khan, Yash Devadiga, ”Hinglish Profanity Filter and Hate Speech Detection,” International Journal of Computer Trends and Technology, vol. 71, no. 2, pp. 1-7, 2023. Crossref, https://doi.org/10.14445/22312803/IJCTT-V71I2P101

[17] Sahin, U., Kucukkaya, I. E., Ozcelik, O., & Toraman, C. (2023). ARC-NLP at Multimodal Hate Speech Event Detection 2023: Multimodal Methods Boosted by Ensemble Learning, Syntactical and Entity Features. arXiv preprint arXiv:2307.13829.

[18] COCOMO Analysis :- https://strs.grc.nasa.gov/repository/forms/cocomo-calculation/

[19] Project Diagrams :- https://app.diagrams.net/?src=about

[20] BERT Architecture :- https://www.turing.com/kb/how-bert-nlp-optimization-model-works

[21] Tensorflow Framework :- https://starship-knowledge.com/

[22] Tensorflow Framework :- https://starship-knowledge.com/

[23] Streamlit Architecture :- https://auth0.com/blog/introduction-to-streamlit-and-streamlit-components/

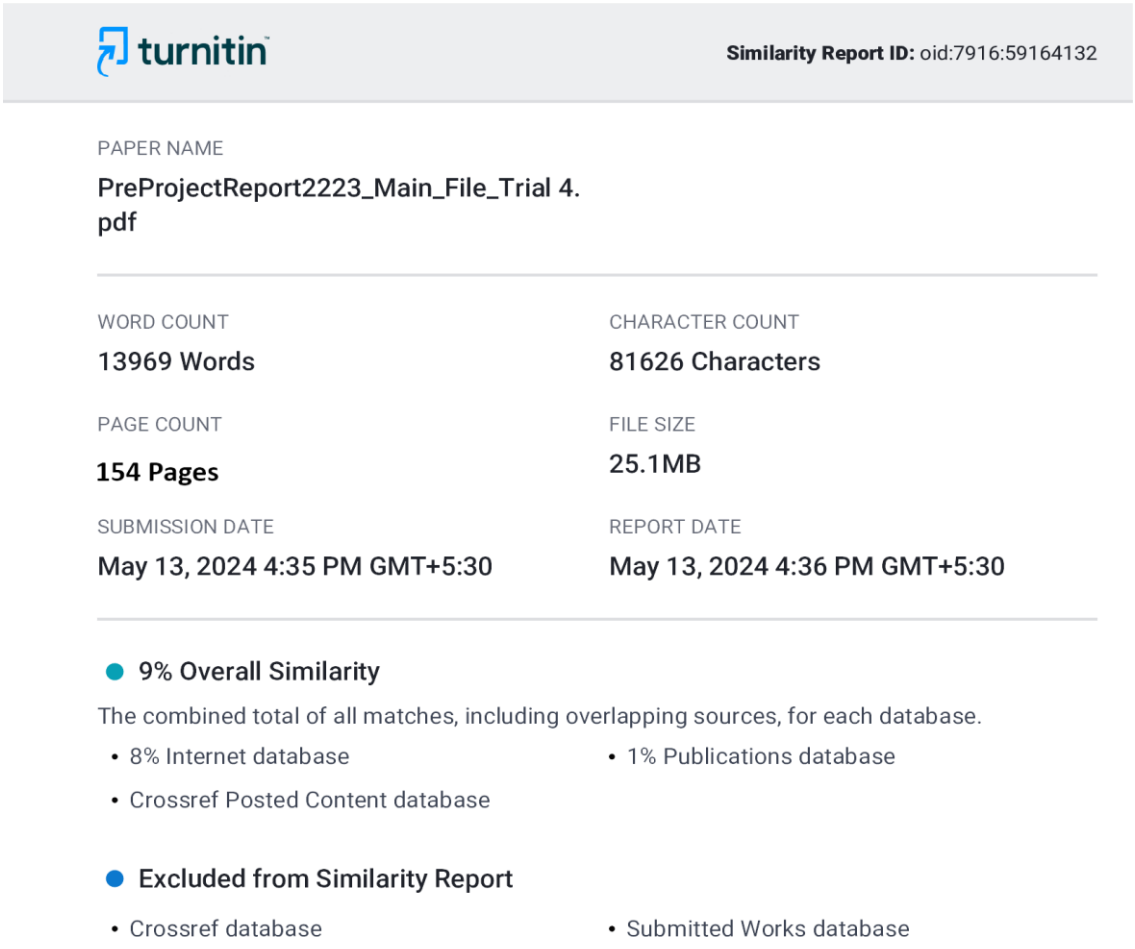
[24] Google Speech Recognition API :- https://www.evonence.com/blog/googles-speech-to-text-api-features-and-reviews

[25] Google Cloud Vision API :- https://medium.datadriveninvestor.com/how-to-build-an-automated-claims-processing-pipeline-using-google-cloud-platform-and-ocr-166a9dc9f5d7

[26] Google Video Intelligence API :- https://medium.com/hackernoon/get-the-code-for-the-video-intelligence-api-demo-794e7675effe

Appendix A

Appendix



turnitin

Similarity Report ID: oid:7916:59164132

PAPER NAME

PreProjectReport2223_Main_File_Trial 4.pdf

WORD COUNT

13969 Words

CHARACTER COUNT

81626 Characters

PAGE COUNT

154 Pages

FILE SIZE

25.1MB

SUBMISSION DATE

May 13, 2024 4:35 PM GMT+5:30

REPORT DATE

May 13, 2024 4:36 PM GMT+5:30

● 9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 8% Internet database
- 1% Publications database
- Crossref Posted Content database

● Excluded from Similarity Report

- Crossref database
- Submitted Works database

Appendix B

Base Paper

Rahul, V. Gupta, V. Sehra and Y. R. Vardhan, ”Hindi-English Code Mixed Hate Speech Detection using Character Level Embeddings,” 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2021, pp. 1112-1118,
doi: 10.1109/ICCMC51019.2021.9418261.

Proceedings of the Fifth International Conference on Computing Methodologies and Communication (ICCMC 2021)
IEEE Xplore Part Number: CFP21K25-ART

2021 5th International Conference on Computing Methodologies and Communication (ICCMC) | 978-1-6654-0360-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICCMC51019.2021.9418261

Hindi-English Code Mixed Hate Speech Detection using Character Level Embeddings

Rahul
Department of Computer Science and Engineering
Delhi Technological University
New Delhi, India
rahul@dtu.ac.in

Vasu Gupta
Department of Computer Science and Engineering
Delhi Technological University
New Delhi, India
vasugupta_2k17co366@dtu.ac.in

Vibhu Sehra
Department of Computer Science and Engineering
Delhi Technological University
New Delhi, India
vibhusehra_2k17co368@dtu.ac.in

Yashaswi Raj Vardhan
Department of Computer Science and Engineering
Delhi Technological University
New Delhi, India
yashaswirajvardhan_2k17co386@dtu.ac.in

***Abstract*—Hinglish is a portmanteau word for 'Hindi' and 'English', and refers to the informal "language" predominantly used in the South-Asian (Indian) Sub-Continent, a blend of the two languages it derives its name from. It considerably differs from the English language in grammar, syntax, punctuations, phonetics and accent, as well as in sentiments.**

As it is more convenient to use English for certain technical words, sports events, scientific phenomena, and other things, mixed usage of English and regional languages has gained considerable prominence in day-to-day conversations and Social Media. This research aims to create an independent and self-sufficing model that classifies Hingish texts as Hate Speech, Abusive or Non-Offensive.

The prevalent use of code-mixed language in the subcontinent, the sensitive nature of hate speeches, and the need of a self-sufficient model for Hinglish, together serve as the motivation for this research.

We have used character level embeddings for Hinglish Language which has the potential to most efficiently extract the context from Hinglish sentences given the level of variation in syntax and semantics of the code-mixed (a language that is a combination of two or more languages) language. Later we trained various deep learning classifier models. Hybridisation of GRU with Attention Model performed best among more than 12 models experimented with. The use of Character Level Embeddings, GRU, and attention layer are novel to Hate Speech Detection in Hinglish Code-Mixed Language.

***Index Terms*—Hate Speech, Character-level Embeddings, Hinglish Sentiment Analysis, Deep Learning, Sequential Models, Code-Switched**

I. Introduction

Social media is gaining pace, and with ever increasing users and the way in which it is being used is also changing. Users find it very convenient to blend a ubiquitous language (like English) with a native language (Hindi in this context), as it is much easier and relatable within the regional context. It is challenging to create a standard model that can classify tweets as hate-inducing or abusive, given a large number of regional languages.

Hate tweets may target a single person or express prejudice against a particular group[1], especially on the basis of race, colour, ethnicity, religion, nationality or sexual orientation, other identity factors that may extend to include a person's disability (mental or physical).

The mere presence of hate speech in public space has the potential to disrupt public order, harm communal harmony, or instigate mobs. Thus, removing these tweets is necessary and with the use of code-switched languages such as HINGLISH, it becomes difficult to remove these tweets using models designed for English tweets.

Abusive text on the other hand may be offensive in a vague sense with some degree of profanity [1]. While these texts are hurtful as well, they do not call for direct intervention. Most platforms deal them with when reported.

In 2019, India ranked first on the Social Hostilities Index, published by Pew Research Center[2], among the 25 most populous countries with an index value of 9.5 out of 10, rising from 8.7 (4th position) in 2014. This expresses the pressing need for a more vigilant social media moderation of hate inducing microblogs. It is also necessary that the mechanism efficiently differentiates between merely offensive from hate tweets so as to not curtail the freedom of expression, as it is guaranteed under most of the constitutions around the world [2], or create a social media policing system.

Hinglish language differs from English as well as Hindi due to the following reasons:

- Use of Roman script instead of Devanagari script (Script for Hindi)
- Absence of fixed grammatical use and high dependence on the region.
- Liberal use of more or fewer punctuations.

[1]https://www.un.org/en/genocideprevention/

[2]https://www.pewforum.org/essay/a-closer-look-at-changing-restrictions-on-religion/

TABLE I
SAMPLE TWEETS

Tweet	Label
#teamIndia congrats on your win. #JaiHo	Non-Offensive
OMG. Jaldi ye offer use krlo. 80% off #Bachat www.abc.com	Non-Offensive
Hindu Muslim Bhai bhai #unity	Non-Offensive
Neem ka patta kadva hai, Salman s**la bha**a hai.	Abusive
@maj–tic_b–ra @ka–nj-h– R*ND* K PILLE TERI M** B*H*N KO K*THE PE BECH K AAYA HU AB	Abusive
@ai-t–in–a Bho**de k b**nch*d,4g ki offer dikhata hai!khud k ga**d k 2g v nhi hai,ch*t**a ulimited t**i	Abusive
Vi–t and A–sh–'s future kid An–h-a: Mamma bolo beta, mammaaaa Kid: Mm.. Ma.. Maa.. M*d*rc**d!	Hate-Inducing
ye mlmaano ko is desh se nikaalo	Hate-Inducing
@dasr–hu–r @na–n–amo-i @A–tSh– @BJP– M*d*rc**d brahmno se mafi mango	Hate-inducing

- Multiple variations in the transliteration of the same word.

The high magnitude of variations in the phonetics makes it challenging to interpret with reference to English speech. E.g., The slang "b*tch" can be translated to 'k*tiy*',' k*ttiy*," k*tiyaa' etc. This causes a challenging situation during the preprocessing step as we want to avoid redundancies.

Merely translating the Hindi text or Hindi portions of Hinglish texts to English and then using an English Offensive Text Classification model might not always be successful as there is vast variation in the grammatical structures, and thus translations may fail to be adequate.

E.g.: 'Muje iske baare mei nhi sunna' original English translation should be 'I do not want to hear about it' but mere conversion into English leads to the sentence 'I it about in no hear' which is grammatically incorrect.

Our model classifies tweets as:

- Hate Inducing
- Abusive
- Non-offensive

Sample tweets of these categories can be seen in Table I. The degree of profanity varying in these categories can be clearly inferred from these examples.

The following models have provided us with the best robust model, capable of gaining all the relationships in the sentences:

1) Only GRU
2) Only GRU with Attention
3) Bidirectional LSTM with Attention + GRU: Attention after LSTM Layer

Our work can be condensed into 5 major steps:

- Preprocessing
- Creation of Character Embeddings
- Model Building
- Training & Hypertuning
- Testing trained models

Here, we have used character-level embedding models. Character level models process neither the word's semantic information nor the ecosystem of pre-trained word vectors. Instead, deep learning models working at the character level come with two key advantages: the vocabulary related issues in the input text of the model are circumvented, and on the output side, a computational bottleneck is averted.

Spelling mistakes, distorted vocabulary, and use of rare words are more common issues in Hinglish texts, as compared to any monolingual text, especially English. The use of character-level deep learning has made our model resilient to such problems and dramatically enhanced the vocabulary it can deal with. Along with this, the use of smaller tokens has made the output less computationally extensive.

The primary contributions of our work may be condensed as follows:

- Character level embeddings were used to deal with the variations in transliteration and grammatical liberties taken in Hinglish, as well as a computational bottleneck caused by word-level embedding
- An exhaustive survey of various Deep Learning architectures for training the model, including sequential stacked as well as unstacked architectures using GRU, Bi-directional LSTM, and Attention Layers, which has not been used for Hinglish Text.

The further sections of this paper contain related work, detailed methodologies, evaluation and lastly conclusion and possible future work in section 2 through 5.

II. RELATED WORK

In [3] analysis was carried out of posts on Facebook by Hinglish Bilingual users, which showed the prevalence of Code Mixed Language on Social Media, and thereby a need to monitor it. One common approach to the classification of Hinglish text has been simple translation of the text into the English language and then use a classifier suitable for English offensive texts. A breakthrough in this method came with [1] with the introduction of a Multi-Channel Transfer Learning-based model that uses Word Embedding of single words and combinations. It also uses sentimental scoring, 67 dimensions of LIWC features, and 210 dimensions of profanity vector. It utilised a hybrid model of CNN and BLSTM to carry out transfer learning and gives f1 score of 0.895.

The study in [4] compare CNN - 1D, LSTM, and Bi-directional LSTM, using domain-specific embeddings creating a 300 Dimension Vector for each word, of which CNN-1D gives the best results at F1-score of 0.8085. [5] created a dataset of Hinglish Text classified into two categories and used a supervised learning approach attaining the highest accuracy of 71.7% by use of SVM Classifier. A similar supervised classification and lexical baselining approach was used in [6], which uses character n-grams, word n-grams, and word skip grams. They were able to attain an accuracy of 78% by using three labels, separating hate speech from offensive language.

The study [7] put forward that hate speech has a presence in the 'long tail' of the dataset, and the lack of peculiar and distinct features make its detection a challenging task. They have used TF-IDF weighted word 1-gram, 2-gram, and 3-gram, number of mentions, hashtags, characters and words, alongside Part-of-Speech (PoS) tag and carried out a removal of candidates having document frequency under five. CNN+ sCNN has performed better than CNN+GRU in all the tests, but the difference in F1 scores is only 1-5%.

Taking an example of work on non-English languages other than Hindi/Hinglish, [8] worked on Greek, presenting the first Offensive Greek Tweet Dataset (OGTD) containing 4,779 posts, with tweets annotated as Offensive, Not Offensive, and Spam. Similar to [7], the TF-IDF approach was taken, as well as part-of-speech (POS), and dependency relation tags and a 300-dimensional vector were used. They also experimented with some stacked and unstacked deep learning architectures obtaining the best results from LSTM and GRU with attention model. Work in [9] was carried out on a Dutch corpus from a popular question-answer-based social media platform Ask.fm, while [10] worked on a corpus of Facebook posts from anti-Islamic groups. [11] has worked on a corpus of hate tweets in German, targeting refugees.

The approach used in [12] for hate speech detection explore using Bag of words, N-grams, Character level n-grams, which might help with spelling problems as well as the frequency of @ mentions, punctuations, token length, words not found in English dictionary, variety of characters that are non-alpha-numeric in tokens. As hate speech is applied to a small chunk of text, we might face the problem of sparsity with Bag of Words, and in that case, Word Generalisation comes into play. It may be done by word clustering or using the Latent Dirichlet Allocation(LDA) topic distribution technique. Thus, word embedding emerges as a feature. Sentiment analysis, lexical analysis, as well as linguistic features like POS and Dependency Relationships were examined.

Multi-tiered pipeline was created in [13]. The first being profanity modeling, second being deep graph embeddings, and the last one, author profiling. Their work uses targeted hate embeddings combined with social network-based features on various baselines and two real-world datasets. They have included an expert-in-the-loop algorithm within the pipeline framework for debiasing.

III. Methodology

A. Preprocessing

The data obtained was streamed through a series of pre-processing steps. Preprocessing is a crucial step as raw text often contains data that may contain errors and redundant information that can deter the model's training.

The first step in the preprocessing was as follows:

1) Converting tweets to lowercase
2) Removing all the @ mentions.
3) Removing hashtags (#) from the tweet. E.g., this is a # tweet → this is a tweet

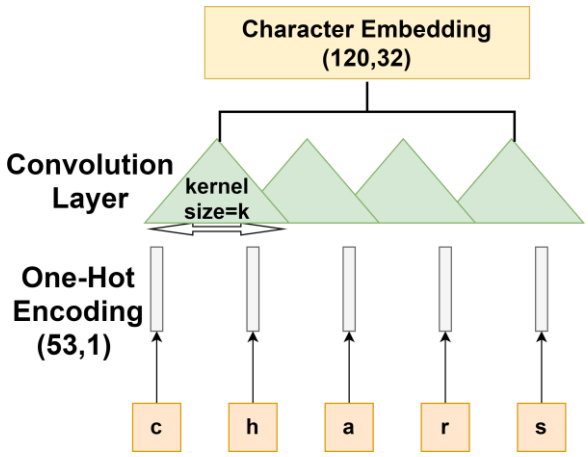


Fig. 1. Character-Level Embeddings Generation

4) Removing web links.
5) Removing digits

B. Character-Level Embeddings Generation

After preprocessing, the tweets are converted into character embeddings through the following process:

1) The average length of a tweet is determined. In this dataset, it is found that most of the preprocessed tweets are no longer than 120 characters. Therefore, the maximum length is taken as 120.
2) The total number of unique characters is determined in the tweets. We have 53 distinct characters after preprocessing.
3) The tweets are converted to one-hot vectors. Each tweet is of shape (MAXIMUM_LENGTH, TOTAL_ CHARS) = (120,53)
4) These are then passed through a 1D-Convolutional Neural Network (CNN) layer with 32 kernels and kernel size being three, to obtain the character embeddings as shown in Figure 1.

C. Model Building

The output from the embedding layer is passed through our deep learning models, which predicts the probability of belonging to each class of this imbalance class classification scenario.

We have used Convolutional 1D Layers (same as in embedding layer), Batch Normalization Layer for faster convergence as discussed in [14], and Dense layers. We used ReLU activation with Convolutional Layers and the Dense Layers, the exception being the last dense layer. This layer's activation function was softmax.

The basic architecture for CNN, Bi-LSTM, and GRU are described as below:

CNN Model (Figure 2)

EMB_OUPUT → CONV1D(16,3) → BATCH--NORM() → CONV1D(12,3) → BATCHNORM()

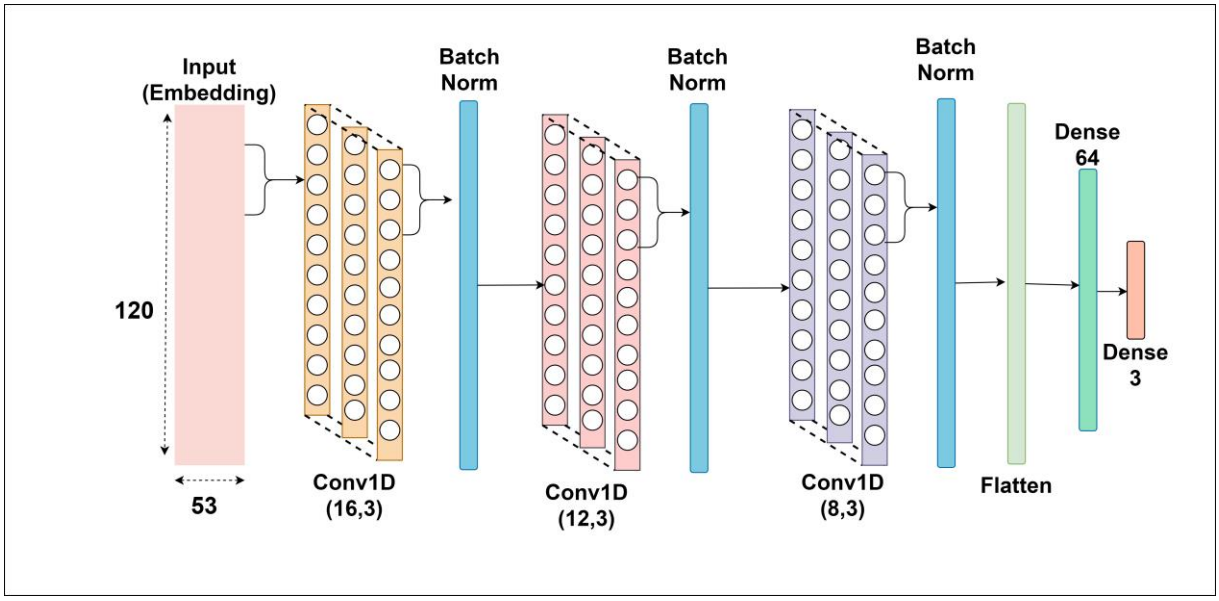


Fig. 2. Proposed CNN Model

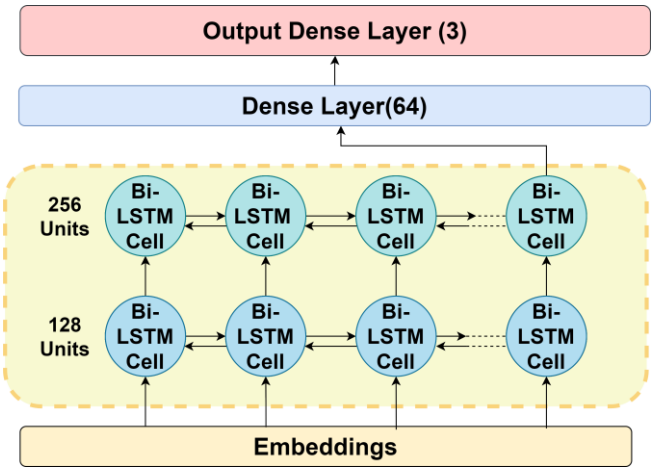


Fig. 3. Proposed Bi-LSTM Model

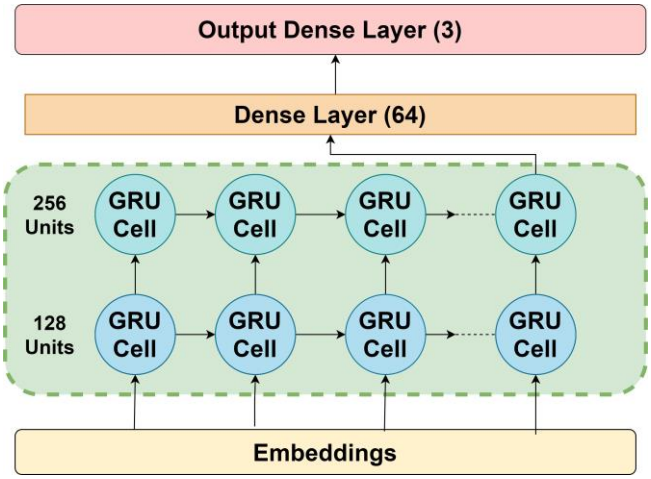


Fig. 4. Proposed GRU Model

→CONV1D(8,3) → BATCHNORM() → DROPOUT(0.2) → FLATTEN() → DENSE(64) → DENSE(3)

Bidirectional LSTM Model (Figure 3)

EMB_OUPUT → BLSTM(128) → DROPOUT(0.25) → BLSTM(256) → DROPOUT(0.25) → DENSE(64) → DENSE(3)

GRU Model (Figure 4)

EMB_OUPUT → GRU(128) → DROPOUT(0.25) → GRU(256) → DROPOUT(0.25) → DENSE(64) → DENSE(3)

Bidirectional GRU Model

EMB_OUPUT → BGRU(128) → DROPOUT(0.25) → BGRU(256) → DROPOUT(0.25) → DENSE(64) → DENSE(3)

We then used the above three basic architectures to make a combination of different models. We concatenated the output of the dense(64) layer of individual models and finally added a final dense output layer. The intent behind this was to see if combining the two models could give us an added advantage over the use of these models separately. CNN being a non-sequential model, whereas, Bi-LSTM and GRU being sequential models have their own set separate of advantages. All the different models created are listed below:

- CNN + GRU
- Bi-LSTM + GRU

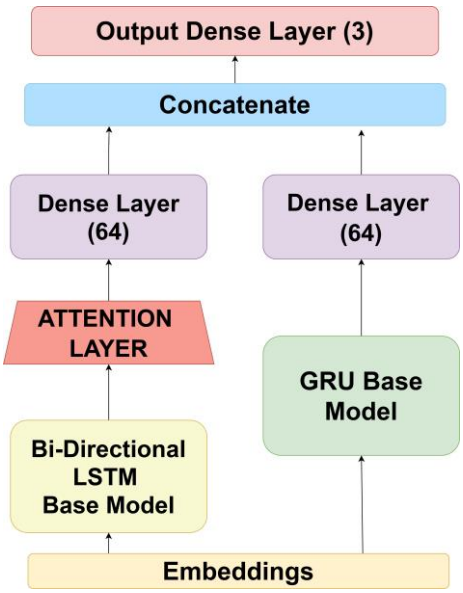


Fig. 5. Proposed Bi-LSTM(with Attention) + GRU Model

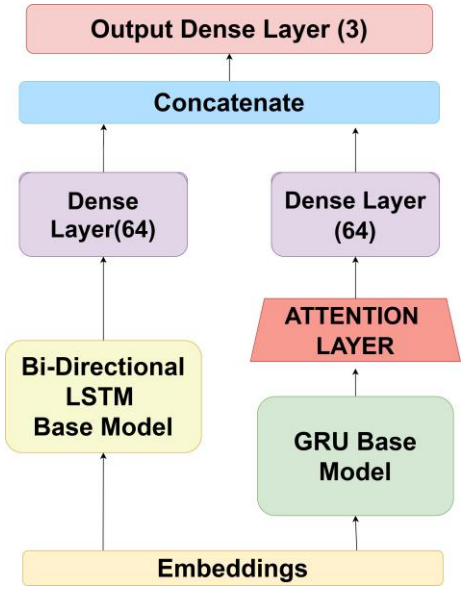


Fig. 6. Proposed Bi-LSTM + GRU Model(with Attention)

- CNN + Bi-LSTM

The idea behind stacking the dense layers from the two models (Bi-LSTM and GRU) is that when we concatenate the two layers, we are able to use the features extracted from the two models and add them to improve the overall understanding of the model.

Further, an attempt was made to enhance the results with the help of the attention layer [15], as it helps to provide weightage to output of the previous layer and pass on that weighted output to the next layer for better learning because some parts of the data are of greater importance for the prediction as compared to others. attention layer. We introduced attention in the following models:

- Bi-LSTM
- GRU
- Bi-GRU
- Bi-LSTM + GRU (with attention layer after Bi-LSTM layer) (Figure 5)
- Bi-LSTM + GRU (with attention layer after GRU layer) (Figure 6)

IV. Evaluation

A. Dataset

The dataset is obtained from [1], and it contains 3189 tweets. This dataset contains tweets written in the Hinglish text, classified into three categories based on the profanity in the text. The three categories were namely: Hate Inducing, Abusive, and Non-Offensive. After preprocessing, some of the tweets were reduced to length zero. The results of baseline models like SVM and Random Forest using Character n-grams, Bag of Words and TF-IDF give an F1-score in the range of 0.574-0.723. The best performing baseline model is SVM classifier with TF-IDF feature with F1-score of 0.723.

TABLE II
Dataset Description

Label	Tweets Count
Non-Offensive	1018
Abusive	1764
Hate-Inducing	303
Total	**3085**

So, Table II is the final dataset size after preprocessing.

Further, we split the formed dataset with the train-test size ratio being 80:20. We used sklearn's train test split with random state 6. The final division of tweets belonging to various classes can be seen in Table VI.

B. Training Details

For the creation of model architectures, Keras Library with Tensorflow backend was used. To train each model, we used a loss function named Categorical Cross Entropy. We used Adam optimizer [16] with learning rate = 10^{-3}. All the model architectures were trained using ten-fold cross-validation. The batch size was kept as 8, and the model was trained for 50 epochs.

We used L2 regularization [17] with $\lambda = 10^{-5}$ in each layer as a kernel regularizer to prevent the model from overfitting as our dataset is relatively small, and there is a high probability of the model being overfitted.

C. Results and Discussion

As stated in the previous section, a train-test split of 80:20 had reserved 20% of the dataset for testing, which was used for the preparation of our results. A common problem in a classification scenario is when the ratio of observations in each class is disproportionate; that is, the distribution is biased or skewed, as is the case with our dataset, making it an imbalanced class. Therefore, accuracy is not an apt measure

TABLE III
RESULTS WITH BASIC PROPOSED MODELS

Model Name	Accuracy	F1-Score	Precision	Recall
CNN Model	0.72	0.71	0.70	0.72
Bidirectional LSTM Model	0.85	0.85	**0.86**	0.85
GRU Model	**0.86**	**0.86**	**0.86**	**0.86**
Bidirectional GRU Model	0.85	0.85	**0.86**	0.85

TABLE IV
RESULTS WITH STACKED MODELS

Model Name	Accuracy	F1-Score	Precision	Recall
CNN + GRU Model	0.72	0.71	0.71	0.72
Bidirectional LSTM + GRU Model	**0.84**	**0.84**	**0.84**	**0.84**
CNN + Bidirectional LSTM Model	0.73	0.71	0.70	0.73

TABLE V
RESULTS WITH MODELS CONTAINING ATTENTION LAYER

Model Name	Accuracy	F1-Score	Precision	Recall
Bidirectional LSTM with Attention Model	0.85	0.86	0.86	0.85
GRU with Attention Model	**0.87**	**0.87**	**0.87**	**0.87**
Bidirectional GRU with Attention Model	0.85	0.85	0.85	0.85
Bidirectional LSTM with Attention + GRU Model	0.86	0.86	**0.87**	0.86
GRU with Attention + Bidirectional LSTM Model	0.84	0.84	0.84	0.84

TABLE VI
DISTRIBUTION AFTER TRAIN-TEST SPLIT

Label	Train	Test
Non-Offensive	815	203
Abusive	1406	358
Hate-Inducing	247	56
Total	**2538**	**651**

for the performance of the models; instead, f1-score was taken for this purpose. F1-score is the harmonic mean of Precision, that is the positive predictive value, and Recall, the measure of sensitivity of the model.

The predictions were generated by passing the 20% data, previously reserved as test-data, through the models. The results were then tabulated using classification reports of the SciKitLearn library of python, which takes predicted and actual values of classes as function parameters.

The predictions were generated by passing the 20% data, previously reserved as test-data, through the models. The results were then tabulated using classification reports of the SciKitLearn library of python, which takes predicted and actual values of classes as function parameters.

The focus of this work was on implementing Character Level Embedding. This was done to avert the demerits of using word-level embeddings with Hinglish text, such as flawed grammar and vocabulary, as well as a computational bottleneck.

An exhaustive trial of various deep learning models and their layered combinations was carried out. It started with basic model architectures like CNN, Bi-LSTM and GRU. Table III shows results for the same. It can be seen, CNN did not perform well which was expected as CNN is too non sequential to suit in this case. Whereas, sequential models like GRU and LSTM performed significantly better. GRU models perform the best, with F1-Score of 0.86. This depicts that sequential architecture is able to extract suitable features from the sentiments and categorize them accordingly.

Even though, the individual performance of CNN model was below par with an f1-score of 0.71, , we stacked it with other sequential models with an idea to combine their advantages, however, it can be seen from Table IV, that stacked CNN Models did not perform well either, giving similar accuracy and f1-score.

For further experimentation, we drop the usage of CNN and introduce Attention Layer to our previously best performing architectures. Table V depicts the results for the same. The introduction of Attention layer enhances the performance in nearly all cases. This was expected as contribution of all parts is not uniform in building the meaning and sentiments of the sentence, and attention layer mechanism facilitates by giving more focus to those tokens which contribute more. The GRU model with Attention layer is the best performing model, among the various combinations we have attempted. It has the highest f1-score, accuracy, precision and recall at 0.87 each.

This was followed by Bidirectional LSTM with Attention layer and GRU, with an f1-score, accuracy and recall of 0.86 and precision at the same level as the previous model at 0.87.

In comparison, these results surpass the related work done in Hinglish Language like Bohra et al, which uses Random Forest Classifier and Support Vector Machine to achieve a maximum accuracy of 0.699 and 0.717 respectively. Kamble et al also remain limited to a highest f1-score of 0.808 and 0.804 with the use of CNN and Bi-LSTM respectively.

Our approach also significantly outperforms other works on Code-Mixed languages. Tulkens et al's detection of racism in Dutch social media using a dictionary based approach has a maximum f1-score of 0.50 and a highest AUC of 0.63 only.

V. Conclusion and Future Work

We experimented with 12 model architectures to train our model on Character Level Embedding for multiclass labeling of Hinglish tweets into three categories(non-offensive, abusive, hate-inducing tweets). Among them, three models have performed considerably better than the rest and are recommended for use in this work: Only GRU, Only GRU with Attention, Bidirectional LSTM + GRU: with Attention after LSTM Layer. Our approach is robust and capable of learning complex dependencies known today or which may arrive in the near future.

The use of Character-Level Embedding instead of word embeddings has made our model resilient to the common defect in most Hinglish Code Mixed projects, which is the use of distorted vocabulary and a multitude of spelling mistakes. A possible enhancement to our work can be done if a larger corpus of tweets in Hinglish is available for the training of models. Another possibility is using an architecture that combines word-level embeddings and character-level embeddings while still averting the computational bottleneck caused by word-level embeddings and resolving the shortcomings of the character level approach. The use of transformer models has been proposed by various authors [18], [19], [20] and it may also enhance the working of our model.

References

[1] P. Mathur, R. Sawhney, M. Ayyar, and R. Shah, "Did you offend me? Classification of Offensive Tweets in Hinglish Language," 2019, pp. 138–148.

[2] C. O'Regan, "Hate speech Online: An (intractable) contemporary challenge?" *Current Legal Problems*, 2018.

[3] K. Bali, J. Sharma, M. Choudhury, and Y. Vyas, ""I am borrowing ya mixing ?" An Analysis of English-Hindi Code Mixing in Facebook," 2015.

[4] S. Kamble and A. Joshi, "Hate speech detection from code-mixed Hindi-english tweets using deep learning models," 2018.

[5] A. Bohra, D. Vijay, V. Singh, S. S. Akhtar, and M. Shrivastava, "A Dataset of Hindi-English Code-Mixed Social Media Text for Hate Speech Detection," 2018.

[6] S. Malmasi and M. Zampieri, "Detecting hate speech in social media," in *International Conference Recent Advances in Natural Language Processing, RANLP*, 2017.

[7] Z. Zhang and L. Luo, "Hate speech detection: A solved problem? The challenging case of long tail on Twitter," *Semantic Web*, 2019.

[8] Z. Pitenis, M. Zampieri, and T. Ranasinghe, "Offensive language identification in Greek," in *LREC 2020 - 12th International Conference on Language Resources and Evaluation, Conference Proceedings*, 2020.

[9] C. Van Hee, E. Lefever, B. Verhoeven, J. Mennes, B. Desmet, G. De Pauw, W. Daelemans, and V. Hoste, "Automatic detection and prevention of cyberbullying," *International Conference on Human and Social Analytics (HUSO 2015)*, 2015.

[10] S. Tulkens, L. Hilte, E. Lodewyckx, B. Verhoeven, and W. Daelemans, "The Automated Detection of Racist Discourse in Dutch Social Media," in *Computational Linguistics in the Netherlands Journal*, 2016.

[11] B. Ross, M. Rist, G. Carbonell, B. Cabrera, N. Kurowsky, and M. Wojatzki, "Measuring the Reliability of Hate Speech Annotations: The Case of the European Refugee Crisis," 2017.

[12] A. Schmidt and M. Wiegand, "A Survey on Hate Speech Detection using Natural Language Processing," 2017.

[13] S. Chopra, R. Sawhney, P. Mathur, and R. Ratn Shah, "Hindi-English Hate Speech Detection: Author Profiling, Debiasing, and Practical Perspectives," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *32nd International Conference on Machine Learning, ICML 2015*, 2015.

[15] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv*, vol. 1409, 09 2014.

[16] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[17] J. Schmidhuber, "Deep Learning in neural networks: An overview," 2015.

[18] K. Pant and T. Dadu, "Towards Code-switched Classification Exploiting Constituent Language Resources," in *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*. Suzhou, China: Association for Computational Linguistics, dec 2020, pp. 37–43. [Online]. Available: https://www.aclweb.org/anthology/2020.aacl-srw.6

[19] M. Mozafari, R. Farahbakhsh, and N. Crespi, "A bert-based transfer learning approach for hate speech detection in online social media," 12 2019.

[20] R. Mutanga, N. Naicker, and O. O, "Hate speech detection in twitter using transformer methods," *International Journal of Advanced Computer Science and Applications*, vol. 11, 01 2020.

Appendix C

Tools Used

1. **Overleaf:** Overleaf is free to use. One Can Create, Edit and Share projects with a sign up method. Overleaf is a real time editor for used to research paper and projects. Overleaf is a cloud based LaTeX editor used for writing,editing and publishing scientific documents. Overleaf can be a access multiple user at a time.
2. **Anaconda Navigator:** Anaconda is an open source tool. Anaconda Navigator also includes a graphical user interface. It can be used for python and R programming language for a data science that aims to simplify package management and deployment. Anaconda Navigator can launch any applications and manage anaconda package without using command line interface.
3. **Google Colab:** Google Colab is a powerful cloud-based platform for creating, sharing, and collaborating on documents, particularly in the realm of data science and machine learning. Colab, short for Colaboratory, extends the functionality of Jupyter Notebooks to a cloud environment, allowing users to write and execute code in Python and other languages like R seamlessly. It provides access to powerful computational resources and facilitates collaborative work through easy sharing and real-time editing capabilities. With Google Colab, you can leverage the versatility and scalability of Jupyter Notebooks in a convenient online environment.

4. **VS Code:** Visual Studio Code (VS Code) by Microsoft is a lightweight, cross-platform source code editor. It is a popular choice among devel opers because it combines the simplicity of a text editor with powerful coding features. Because of its simple interface and many customiza tion options, VS Code delivers a pleasant writing experience across multiple programming languages. Syntax highlighting, code comple tion, debugging, version control integration, and a strong ecosystem of extensions to increase capabilities are all included. Its versatility, speed, and breadth of capability make it a go-to tool for develop- ers of all levels, allowing them to build and debug code quickly and efficiently.

5. **Draw.io:** Draw.io is free online diagram software. It can be used for making flowchart,process diagram, DFD diagram, UML diagram and network diagram.

Appendix D

Certificates:

Unveiling Online Hate using Deep Learning Techniques

Participated in State Level Final Year Project Competition

PROJECT NEXUS 1.0

at Army Institute of Technology

by

Yash Suhas Shukla (B190358574)

Tanmay Dnyaneshwar Nigade (B190358554)

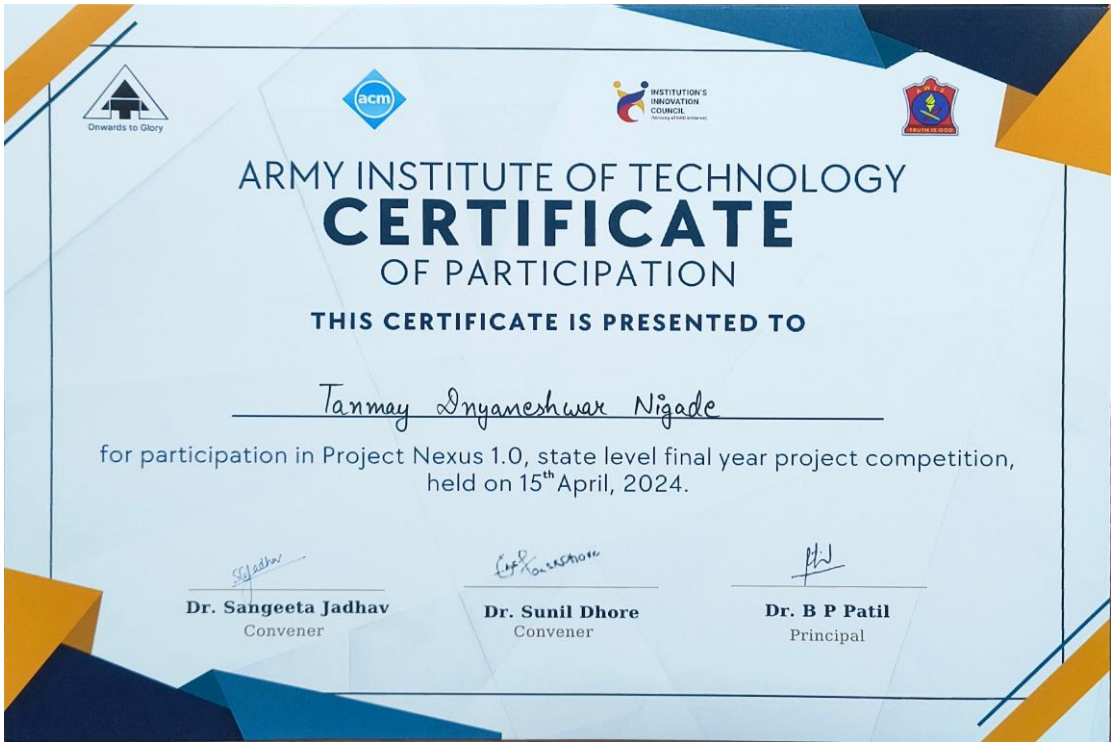
Suyash Vikas Khodade (B190358536)

Prathamesh Dilip Pimpalkar (B190358565)

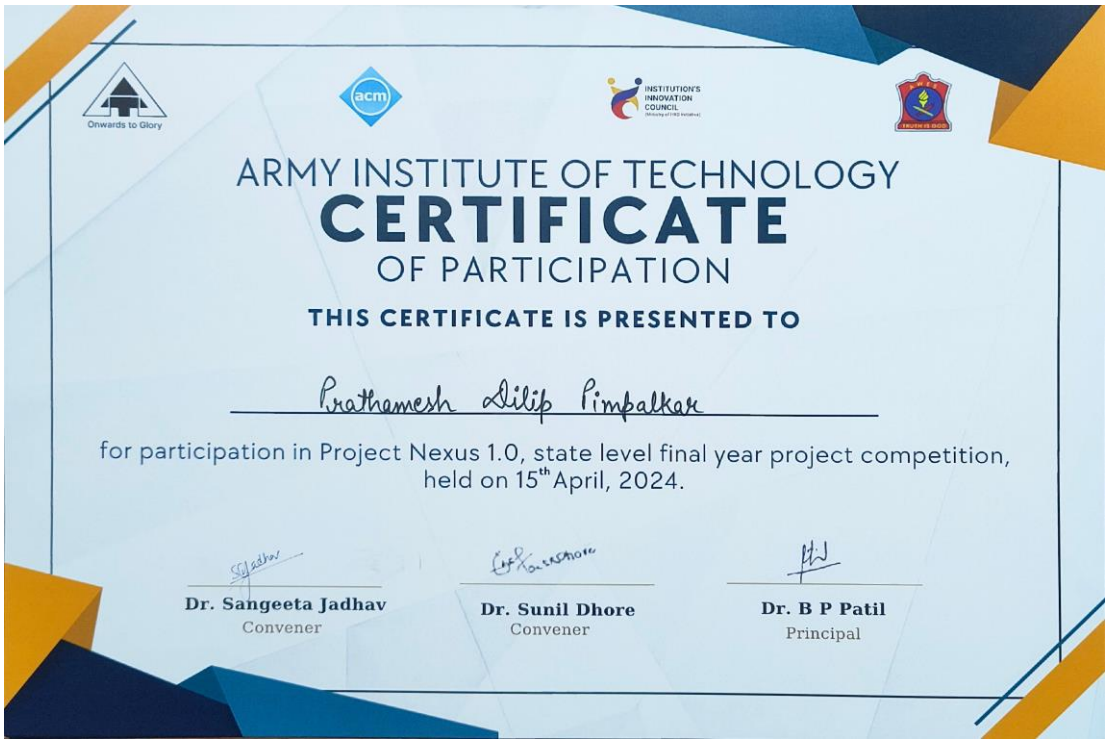
Under the guidance of

Mr. R. V. Panchal


Onwards to Glory
acm
INSTITUTION'S INNOVATION COUNCIL
ARMY INSTITUTE OF TECHNOLOGY
CERTIFICATE
OF PARTICIPATION
THIS CERTIFICATE IS PRESENTED TO
Yash Suhas Shukla
for participation in Project Nexus 1.0, state level final year project competition, held on 15th April, 2024.
Dr. Sangeeta Jadhav
Convener
Dr. Sunil Dhore
Convener
Dr. B P Patil
Principal


Onwards to Glory
acm
INSTITUTION'S INNOVATION COUNCIL
ARMY INSTITUTE OF TECHNOLOGY
CERTIFICATE
OF PARTICIPATION
THIS CERTIFICATE IS PRESENTED TO
Tanmay Dnyaneshwar Nigade
for participation in Project Nexus 1.0, state level final year project competition, held on 15th April, 2024.
Dr. Sangeeta Jadhav
Convener
Dr. Sunil Dhore
Convener
Dr. B P Patil
Principal


Onwards to Glory
acm
INSTITUTION'S INNOVATION COUNCIL
ARMY INSTITUTE OF TECHNOLOGY
CERTIFICATE
OF PARTICIPATION
THIS CERTIFICATE IS PRESENTED TO
Suyash Vikas Khodade
for participation in Project Nexus 1.0, state level final year project competition, held on 15th April, 2024.
Dr. Sangeeta Jadhav
Convener
Dr. Sunil Dhore
Convener
Dr. B P Patil
Principal


Onwards to Glory
acm
INSTITUTION'S INNOVATION COUNCIL
ARMY INSTITUTE OF TECHNOLOGY
CERTIFICATE
OF PARTICIPATION
THIS CERTIFICATE IS PRESENTED TO
Prathamesh Dilip Pimpalkar
for participation in Project Nexus 1.0, state level final year project competition, held on 15th April, 2024.
Dr. Sangeeta Jadhav
Convener
Dr. Sunil Dhore
Convener
Dr. B P Patil
Principal