RAJIV GANDHI INSTITUTE OF TECHNOLOGY
GOVERNMENT ENGINEERING COLLEGE
KOTTAYAM-686 501

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

CSD 415 PROJECT PHASE - II REPORT

OFFENSIVE LANGUAGE DETECTION OF MALAYALAM-ENGLISH CODE-MIX TEXT

Submitted by

ASHNA ANNA PHILIPS(KTE21CS014)
SAFANA P S(KTE21CS049)
T S PARVATHY(LKTE21CS077)
AUGASTINA JOHNSON(NSS21CS024)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY
THIRUVANANTHAPURAM

NOVEMBER 2024

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE

KOTTAYAM-686 501

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

Vision of the Department

To be a centre of excellence for inclusively nurturing young minds to become innovative computing professionals committed to sustainable development and societal empowerment.

Mission of the Department

- To offer a solid foundation in computing and technology for crafting competent professionals.
- To promote innovative and entrepreneurial skills of students by exposing them to the forefront of developments in the field of computing.
- To inculcate strong ethical values in the young minds to work with commitment for the progress of the nation.

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE

KOTTAYAM-686 501

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this report entitled ***OFFENSIVE LANGUAGE DETECTION OF MALAYALAM-ENGLISH CODE-MIX TEXT*** *is an authentic report of the project done by the team consisting of* ***ASHNA ANNA PHILIPS(Reg. No: KTE21CS014), SAFANA P S(Reg. No: KTE21CS049), T S PARVATHY(Reg. No: LKTE21CS077), and AUGASTINA JOHNSON(Reg. No: NSS21CS024)*** *during the academic year* ***2024-25****, in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering of APJ Abdul Kalam Technological University, Thiruvananthapuram.*

GUIDE COORDINATOR

HEAD OF THE DEPARTMENT

Acknowledgement

Every successful project is the outcome of hard work and strong support and guidance given by a number of people. We sincerely appreciate the inspiration, support and guidance of all those people who have been instrumental in making this project a success. We express our sincere gratitude to Dr. Prince A., Principal for making the resources available at the right time without which this project would not have been a success. We take this opportunity to express a deep sense of gratitude to Prof. Kavitha N., Associate Professor & Head, Department of Computer Science and Engineering. We also take this opportunity to express our profound gratitude and deep regards to our guide Prof. Raji R Pillai for her exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by her from time to time shall carry us a long way in the journey of life on which we are about to embark. We also express our gratitude to Project Coordinators Prof. Nisha K K and Prof. Anupama Jayan and Prof. Unnikrishnan K for the cordial support, valuable information and guidance, which helped us in completing this task through various stages. Last, but not least, we thank The Almighty, our parents and friends for their constant encouragement without which this project would not have been possible.

Ashna Anna Philips
Safana P S
T S Parvathy
Augastina Johnson

Declaration

We, the undersigned hereby declare that the project report entitled Project **OFFENSIVE LANGUAGE DETECTION OF MALAYALAM-ENGLISH CODE-MIX TEXT**, submitted for partial fulfilment of the requirements for the award of the degree of Bachelor of Technology of the **APJ Abdul Kalam Technological University, Kerala** is a bonafide work done by us under the supervision of **Prof. Raji R. Pillai**. This submission represents our idea in our own words where ideas or words of others have not been included; we have adequately and accurately cited and referenced the original sources. We have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data, idea or on our submission. We understand that any violation of the above can result in disciplinary actions from the institute and/ or University and can evoke penal action from the sources which have not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed as the basis for awarding any degree, diploma or similar title of any other university.

Name of Students: Signatures:

ASHNA ANNA PHILIPS (Reg. No.: KTE21CS014)

SAFANA P S (Reg. No.: KTE21CS049)

T S PARVATHY (Reg. No.: LKTE21CS077)

AUGASTINA JOHNSON (Reg. No.: NSS21CS024)

Batch: 2021 - 2025 April 6, 2025

Abstract

This project aims to detect offensive language in Malayalam-English code-mixed text, particularly in social media contexts where language mixing and evolving trends are common. Using the DravidianCodemix Malayalam dataset, which contains labeled comments, we developed multiple classification approaches to improve detection accuracy despite significant class imbalance.

Initially, BERT embeddings with class weights were applied to four classifiers—Random Forest, Logistic Regression, KNN, and Naïve Bayes—using majority voting. However, due to imbalance issues, we implemented a Multilayer Perceptron (MLP) trained with BERT embeddings, class weights, and focal loss, achieving 87%.

These models are integrated into a simple frontend implementation for real-time comment testing. The four-classifier approach retains majority voting, while the MLP model operates independently, enhancing offensive language detection in multilingual social media text.

Contents

List of Abbreviations

AI Artificial Intelligence

NLP Natural Language Processing

ML Machine Learning

SVM Support Vector Machine

HASOC Hate Speech and Offensive Content Identification

BERT Bidirectional Encoder Representations from Transformers

List of Figures

List of Algorithms

Chapter 1

Introduction

The Offensive Language Detection in Malayalam-English Code-Mix Text project aims to effectively monitor and classify offensive language within multilingual online conversations, specifically Malayalam-English code-mixed text. Utilizing Natural Language Processing (NLP) and Machine Learning (ML) techniques, this system processes comments from the DravidianLangTech 2021 dataset [**?**], identifying offensive and non-offensive language in social media contexts.

After preprocessing the text, embeddings are generated through BERT, which are then passed through various classifiers to decide which class a particular comment falls under. These embeddings are then passed to a classification unit, testing models like Support Vector Machine (SVM) and Random Forest to identify the most effective approach. This project offers a practical solution for detection and management of offensive language in code-mixed environments.

1.1 Objectives

This project aims to achieve the following objectives:

1. To design and implement a machine learning model for detecting offensive language in Malayalam-English code-mixed text, with a focus on effective NLP and ML techniques. The process involves dataset preprocessing by handling stop words, tokenization, and retaining punctuation and emoticons to support sentiment and tone detection. Additionally, BERT-based embeddings are generated with context-aware enhancements boosting classification accuracy.

2. To expand the dataset with recent comments to help the model learn evolving slang and offensive terms, enabling it to adapt to emerging trends in language use and maintain relevance in real-time social media contexts.

3. To develop a user-friendly interface that allows users to input comments and instantly receive feedback on whether the content is offensive or non-offensive.

1.2 Motivation

The facts that motivated the choice of the project are the following.

1. Addressing Offensive Language in Code-Mixed Text: Offensive language in Malayalam-English code-mixed text on social media can impact the well-being of online communities. Developing a model that understands the nuances of code-mixed language can help reduce harmful content in these spaces.

2. Supporting Safe Online Interactions: By accurately identifying and classifying offensive language, this project aims to foster a safer online environment for Malayalam-speaking communities, ensuring respectful and inclusive interactions.

3. Filling a Gap in NLP Research: Existing NLP models often struggle with multilingual and code-mixed text. This project seeks to contribute to the growing field of multilingual NLP by focusing specifically on Malayalam-English language processing.

1.3 Scope of the Project

- **Need:** To detect and classify offensive language in Malayalam-English code-mixed comments to promote a safe online environment and mitigate the impact of harmful content on users.

- **Deliverables:** A functional web-based application with a user-friendly interface for inputting comments, including a text input field and a display area for results. The backend system will handle processing and classifying comments using a pre-trained ML model. This system will encompass preprocessing of input data, embedding generation using BERTb, and classification of comments as offensive or non-offensive.

- **Exclusions:** The project will not include real-time processing of comments and will not feature any functionality for user reporting or feedback mechanisms.

- **Assumptions:** It is assumed that users of the web application have basic internet connectivity and access to devices capable of running web browsers for inputting comments.

1.4 Prerequisites for the Reader

To understand the contents of this report, the reader is expected to meet the following prerequisites:

- Readers should be familiar with web applications and have access to a device capable of running a web browser for practical experimentation and exploration of the developed application.

- A basic understanding of Natural Language Processing (NLP) concepts will facilitate comprehension of the project's methodologies, particularly in relation to text processing and embedding generation.
- Familiarity with machine learning concepts, including classification algorithms and model training, will aid in understanding the classification unit of the project.
- Knowledge of web development, including frontend (HTML, CSS, JavaScript) and backend (e.g., Flask or Node.js) technologies, is beneficial for grasping the overall architecture and communication flow of the application.

1.5 Organization of the Report

Chapter 2 begins with the System Study, which includes an exploration of the Existing Systems (section 2.1) and continues with a Gap Analysis (section 2.2). It then proceeds to detail the Proposed System (section 2.3) and concludes with the Requirements Engineering (section 2.4). Chapter 3 focuses on the design of the project, starting with the System Design (section 3.1) that provides an overview of the different modules involved, detailed in sections 3.1.1, 3.1.2, 3.1.3 and 3.1.4. Section 3.2 presents the Detailed Design, including algorithms for text preprocessing and classification in section 3.2.1. Chapter 4 outlines the implementation of the system covering the Tools Used (section 4.1) for building the project, with System Implementation (section 4.2) covering the details of Data Preprocessing (section 4.2.1) and Model Development (section 4.2.2), followed by the Results Analysis(section 4.2.3). Chapter 5 focuses on the system's testing phase explaining the Testing Strategies Used(section 5.1) and the presenting the Testing Results (section 5.2). Chapter 6 presents the Results of the project detailing the achieved objectives in Results(section 6.1) and showcasing the finalized system through screenshots. Chapter 7 concludes with a summary of the entire project.

Chapter 2

Literature Survey

A literature survey is a fundamental phase in research that involves a comprehensive review of existing studies, methodologies, and findings related to the problem domain. This process helps in understanding the current state of research, identifying gaps, and establishing a foundation for the proposed work. By analyzing relevant studies, we gain insights into effective techniques, challenges, and advancements in the field, ensuring that our approach is well-informed and builds upon existing knowledge.

2.1 Introduction

Hate speech refers to the attack on a person or a group through communications based on their race, religion, gender, or other factors. The rise of social media platforms has led to an increase in hate speech and offensive language more common, as people often feel free to express their hatred towards others through comments. This behavior can cause significant harm, affecting both individuals and society. Therefore, there is a need for offensive language detection to mitigate the spread of harmful content on social media.

Most research on offensive language detection has focused on the English language [1–4], primarily due to the availability of datasets that facilitate the study and easier detection of offensive comments. However, when it comes to regional languages such as Malayalam, Tamil, Kannada, etc., [5] there is a lack of datasets [6] available for effective detection.

Additionally, people often mix languages while communicating, which is particularly common in Indian languages, like Malayalam-English code-mixed text (Manglish). Because of the frequent use of highly code-mixed words detecting offensive comments using traditional methods becomes challenging, making this a significant area of study. Researchers are increasingly focusing on detecting offensive speech in Dravidian code-mixed languages, and this field continues to evolve. In this paper, we focus on hate speech and offensive language detection in Malayalam-English code-mixed text (Manglish), we examine key challenges such as linguistic nuances and data scarcity as well as analyse the performance of different classifiers.

The following sections provide a detailed overview of the study. Section **??** discusses various preprocessing techniques and features used in the existing studies. Section 2.3 explores the different datasets used for offensive language detection. Section 2.4 analyses the classification methods adopted in various works, while Section 2.5 highlights the challenges faced by these approaches.

2.2 Preprocessing and Feature Extraction

Due to the unique complexities of code-mixed and transliterated text, detecting hate speech and offensive language in Manglish necessitates effective feature extraction and preprocessing techniques. The multilingual nature of Manglish, which is a combination of Roman script with native Dravidian grammar and syntax, demands specialised approaches for tasks such as cleaning, tokenization, and standardisation. As observed in many classification tasks, the major difference lies in the choice of features that must be included to differentiate offensive language from non-offensive language.

2.2.1 Preprocessing Techniques

Several approaches were employed to clean and standardize code-mixed text for offensive speech detection.

- Noise Removal: It is a necessary preprocessing step in offensive language detection, ensuring that only meaningful linguistic information is retained for analysis. Many studies focused on removing unnecessary elements such as emojis, special characters, punctuation, URLs, usernames, and other irrelevant details that might distract the model from understanding the text. Some studies opted for emoji removal to ensure a cleaner dataset [7–9], while others retained emojis or replaced them with their equivalent textual description using tools like Demoji to preserve sentiment information [6, 10, 11]. This step ensured that the text retained only meaningful linguistic information. Additionally, extra white spaces and non-essential stopwords were eliminated to streamline the data further [12].

- Handling Transliteration: Transliteration plays a vital role in processing code-mixed text, converting Romanized scripts back into their native forms to enhance linguistic consistency and improve model performance in sentiment analysis and offensive speech detection. A major challenge in code-mixed text processing is the frequent use of Roman script for languages like Malayalam and Tamil, which can lead to loss of linguistic structure. Some studies addressed this issue by applying Indic NLP libraries to transliterate text into native scripts, ensuring better model compatibility and improving overall text representation [6, 7, 9]. However, other studies opted to process Romanized text directly without transliteration, training models to handle mixed-script inputs instead [8, 12].

- Negation Handling: Certain studies incorporated custom lists of negation words to improve understanding of their impact on sentiment and context. By accurately

identifying negation, models could better distinguish between positive and negative meanings, particularly in hate speech contexts [13].

- Tokenization and Standardisation: Texts were converted to lowercase and tokenized into smaller meaningful units, such as words, unigrams, bigrams, and trigrams. These tokens served as building blocks for subsequent feature extraction steps, ensuring consistency across datasets [9, 10, 14]

2.2.2 Feature Extraction

Researchers employed various techniques to capture linguistic and contextual nuances:

- N-Grams: Character and word n-grams were widely utilized to capture patterns in sequential data. For Malayalam text, character n-grams of lengths up to 5 and word n-grams of lengths up to 2 were shown to be effective [12,14,15]. These methods helped capture both short-term and long-term dependencies in text.

- TF-IDF Features: To emphasize the importance of specific terms in the corpus, Term Frequency-Inverse Document Frequency (TF-IDF) was applied. Combining word and character n-grams with TF-IDF representations provided a robust mechanism to identify key terms in both offensive and non-offensive text [10–15].

- Word Embeddings: Pre-trained embeddings such as MuRIL and MPNet were extensively used to encode multilingual and code-mixed inputs effectively. Custom embeddings were also created to represent unique vocabulary terms, enhancing the models' understanding of less frequent but contextually significant words [12, 14, 15].

- Sentiment and Keyword Features: Sentiment and keyword-based features have been widely utilized in offensive language detection. These approaches involve analysing the frequency of positive and negative words to determine sentiment polarity, aiding in sentiment classification [13]. Additionally, offensive keyword identification has been improved using contextual embeddings, where deep learning models extract and rank offensive terms based on their semantic similarity to predefined offensive speech terms [14]. This combination of sentiment analysis and keyword extraction provides a more targeted and context-aware method for identifying harmful content in multilingual and code-mixed text.

Feature extraction has significantly advanced in sentiment analysis and offensive language detection, transitioning from statistical techniques to deep learning methods. For text representation, Bag of Words (BOW) and TF-IDF were initially used and later n-gram

features were introduced which improved sequential pattern recognition in code-mixed text. With advancements in deep learning, word embeddings such as MPNet, MuRIL, and transformer-based models effectively capture contextual meaning. More recent approaches incorporate sentiment-based word counts and offensive keyword extraction using embedding similarity, which improved the accuracy in offensive language detection. This evolution towards context-sensitive features has enhanced the effectiveness of sentiment analysis across multilingual and code-mixed datasets.

2.2.3 Contextual Features

Contextual understanding of code-mixed text is a key focus area, as it helps in accurately interpreting semantic meaning, syntactic structures, and linguistic variations across multiple languages.

- Semantic Context: Transformer-based embeddings such as mBERT and RoBERTa play a vital role in capturing the nuanced meanings of code-mixed text, particularly in languages like Manglish. These models process text at a subword level, which helps in handling transliterated words, morphological variations, and mixed-language dependencies. Additionally, they leverage self-attention mechanisms to capture long-range dependencies, enabling a better understanding of complex sentence structures [10, 11, 14, 15]. Studies using pretrained models like MuRIL have demonstrated improved contextual representation, especially in handling syntactic variations in Dravidian languages [6].

- Linguistic Nuances: Code-mixed text exhibits unique grammatical patterns and blended syntax, requiring specialized techniques for effective processing. Researchers addressed these challenges by incorporating language-specific preprocessing, such as custom tokenization methods that segment mixed-language words more effectively. Furthermore, embedding techniques trained on Dravidian languages, such as MuRIL and MPNet, have provided a better representation of regional phonetic and morphological structures by capturing language-specific word formations, syntactic variations, and transliteration patterns [6,15]. Some studies also implemented syntactic parsing and part-of-speech tagging adapted for mixed-language constructs, enhancing the overall linguistic understanding of Manglish text [14].

2.3 Dataset

Most of the datasets used in existing studies are provided by HASOC, a shared task aimed at promoting multilingual research in hate speech and offensive content identification on

online platforms [6–9, 11, 12, 14, 15]. Some studies have also used comments from Twitter [13], while [10] incorporated multiple datasets, including Twitter, Reddit, and Gab datasets, along with the HASOC 2020 dataset. The FIRE 2020 HASOC Dravidian-CodeMix Task, classified comments as offensive and non-offensive [8], but the challenges such as orthographic and linguistic variation, restrict its applicability to broader multilingual scenarios. To mitigate the challenge, the dataset was expanded beyond Manglish to include Malayalam YouTube comments and code-mixed Twitter comments [9], which divided the task into two. Task 1 - hate/offensive detection in Malayalam YouTube comments and Task 2 - hate/offensive detection in code mixed Twitter and YouTube comments.

One of the major challenges in hate speech and offensive language detection is class imbalance in datasets (for example, offensive vs. non-offensive comments). The dataset used in [15], which incorporated YouTube comments, exhibited class imbalance, with a significant dominance of non-offensive comments. This imbalance affected the model's ability to detect offensive speech effectively. To enhance model performance in offensive language detection by ensuring a more balanced distribution of offensive and non-offensive comments, Pathak et al. [12] introduced a balanced dataset for Malayalam-English (Manglish). To enhance the diversity and representativeness of offensive language detection datasets, random transliteration between Roman and Indic scripts was applied to improve linguistic diversity [9]. Additionally, context preservation techniques retained key elements such as emoticons and hashtags, aiding in more accurate offensive language detection [6, 10, 11]. Furthermore, weighted loss functions were employed to prioritize offensive samples, increasing model sensitivity to minority classes [6, 10].

Dataset size is another challenge faced by researchers. To address this, the HASOC dataset was expanded by incorporating 8,943 additional comments from Twitter and YouTube [14]. This expansion introduced a final list consisting of 1,082 hate/offensive keywords. Additionally, a test set of 756 comments was collected, including new hate and offensive language (HOL) terms that are absent from the original dataset, enabling a more comprehensive evaluation of model generalisation. Another key effort in dataset improvement was presented in [6], where a newly annotated test set of 1,000 labelled YouTube comments was introduced manually, enhancing data quality and reliability. The annotation process involved multiple annotators where the majority-voting approach ensured higher consistency and reliability in labelling offensive content.

2.4 Methods Used

This section describes the various algorithmic approaches used for detecting hate speech and offensive language in code-mixed Dravidian languages.

2.4.1 Machine Learning Approaches

Initially, different researches were conducted on hate speech and offensive language detection which used conventional machine learning classifiers like Naive Bayes, Support Vector Machine (SVM), and Random Forest [12, 13]. In the work by Soumya and Pramod [13], Malayalam tweets were divided into categories of positive and negative sentiments by creating feature vectors using methods such as Bag of Words, TF-IDF, and Unigram representations enriched with SentiWordNet, with and without using negation words.

To address the lack of sentiment-tagged datasets for Malayalam, they manually curated a corpus of 3,184 tweets retrieved from Twitter, ensuring that each tweet's sentiment was verified despite initial biases in keyword-based retrieval. The results indicate that the Random Forest classifier with Unigram representation enriched with SentiWordNet and negation words showed the highest accuracy.

In another study, Pathak et al. [12] classified Malayalam tweets as offensive or non-offensive using the HASOC dataset (4,000 samples each). They extracted n-gram features with TF-IDF and evaluated classifiers such as Support Vector Machine (SVM), Multinomial Naive Bayes, Logistic Regression, and Random Forest. Their best models achieved an F1 score of 0.77 for Malayalam, ranking competitively in the shared task.

2.4.2 Deep Learning Approaches

Deep learning methods have gained popularity due to their ability to capture semantic and contextual nuances in code-mixed text. In their study, Renjit and Idicula [8] identified offensive language in Manglish tweets by employing an embedding-based classifier for message-level classification. The model was evaluated on the Manglish dataset from Task 2 of the HASOC Offensive Language Identification-DravidianCodeMix sub-track at FIRE 2020, highlighting the practical challenges of handling transliterated and code-mixed social media texts. The best-performing configuration, which combined LSTM and Dense Networks enhanced with Keras Embedding and Doc2Vec feature representations, achieved an F1 score of 0.54.

2.4.3 Transformer-Based Approaches

In response to the limitations of traditional deep learning models in capturing complex semantic relationships, researchers have increasingly turned to transformer-based architectures. Mukherjee and Das [10], handled hate speech detection by fine-tuning state-of-the-art transformer-based models, specifically RoBERTa and XLNet, on four publicly available datasets sourced from platforms like Reddit, Gab, and Twitter. This method leverages the self-attention mechanism inherent in transformers to generate contextual

embeddings that capture semantic nuances without relying on extensive hand-crafted features. The evaluation demonstrated that these models either surpassed or matched traditional baselines such as 1D-CNNs and LSTMs, achieving robust performance across diverse datasets.

Raphel et al. [14] focused on extracting hate and offensive keywords from code-mixed Malayalam. They created an annotated dataset and implemented two approaches which include intrinsic evaluation using n-gram analysis and embeddings from five transformer models. The results show that multilingual BERT achieved the highest similarity scores. They also fine-tuned multilingual BERT for HOL keyword tagging.

In another study, Sreelakshmi et al. [6] explored hate speech and offensive language detection in code-mixed texts in Kannada-English, Malayalam-English and Tamil-English. Various multilingual transformer-based embedding models were combined with a Support Vector Machine (SVM) classifier using a Radial Basis Function (RBF) kernel. Their comprehensive evaluation used models such as BERT, DistilBERT, LaBSE, MuRIL, XLM, IndicBERT, and FNET across six datasets. They also used cost-sensitive learning approach to mitigate class imbalance issues. The results show that MuRIL consistently outperformed other models, achieving 96% accuracy for Malayalam on the DravidianLangTech 2021 dataset.

2.4.4 Ensemble Approaches

As the studies progressed, the advantage of using Ensemble approaches over individual transformer models was realised due to its better generalisation capabilities which led to enhanced accuracy scores.

Zhu and Zhou [7], proposed an ensemble model combining BiLSTM, LSTM+Convolution, and CNN to improve the classification of offensive language in Malayalam-English which used the dataset sourced from HASOC-Dravidian-CodeMix-FIRE 2020 classified as Task 1 and Task 2. For task 1 they achieved an F1 score of 0.93 in Malayalam-English and for task 2 Malaylam-English the F1 score was 0.67.

Singh and Bhattacharya [9], employed an ensemble of multilingual BERT models, enhanced with a novel training strategy that incorporated data augmentation via random transliteration. Their system, tested on the HASOC-Dravidian-CodeMix dataset, achieved an impressive F1-score of 0.95 for Malayalam-English YouTube comments and 0.72 for Malayalam-English Twitter data. The key advantage of their approach was the robustness introduced through transliteration, which helped bridge the script-mixing challenges.

Roy et al. [11] explored a deep ensemble framework that combined transformer-based models (mBERT, DistilBERT, XLM-RoBERTa, and MuRIL) with machine learning classifiers such as Logistic Regression, Random Forest, and Naïve Bayes. Their study utilized the HASOC-Dravidian-CodeMix-FIRE2021 dataset, focusing on code-mixed text in Manglish (Malayalam-English). The weighted ensemble approach outperformed state-of-the-art

models, achieving F1-scores of 0.802 for Malayalam-English.

In another study, an ensemble approach integrating MPNet and CNN was proposed for offensive language detection in Dravidian languages [15]. This model leveraged contextual embeddings from MPNet along with CNN's ability to capture local dependencies in text. The dataset, sourced from HASOC-Dravidian-CodeMix, included a mix of Malayalam-English and Tamil-English comments.

Ensemble models have improved offensive language detection in Dravidian code-mixed languages by utilizing diverse architectures and training techniques. However, challenges like high computational costs and transliteration inconsistencies still require further research and optimization. Table **??** summarises the performance of various models discussed in this paper, providing a comparative overview of their effectiveness in the detection of offensive language.

2.5 Challenges

In this section, we examine the various challenges in detecting offensive language in Malayalam-English code-mixed text.

2.5.1 Code-Mixed Complexity

The code-mixed nature of Manglish poses significant challenges for linguistic analysis and model training. These texts often feature mixed scripts, with words appearing in both Roman and native Dravidian scripts, complicating standard text processing. Additionally, transliteration patterns vary widely among users, leading to spelling inconsistencies and reducing the effectiveness of traditional Natural Language Processing (NLP) techniques.

2.5.2 Data Limitations

One of the major bottlenecks in offensive language detection and sentiment analysis for code-mixed text is the scarcity of annotated datasets. Many studies had to manually create and validate datasets, as there were no publicly available sentiment-tagged or hate speech-specific corpora for Malayalam-English and other Dravidian code-mixed languages [13, 14]. Additionally, existing datasets are often small, leading to unstable model training and poor generalisation on unseen data [6, 7, 11]. The absence of large datasets makes model development difficult, requiring data augmentation to increase dataset variety.

2.5.3 Class Imbalance

A recurring issue in hate speech and offensive language detection is the imbalance between offensive and non-offensive classes in the datasets. Most datasets contain a significantly higher proportion of non-offensive comments, leading to models that tend to favor the dominant class while underperforming on offensive content [6, 8, 12, 15]. Studies attempted to mitigate this issue using techniques such as weighted loss functions but results have been inconsistent [6,10]. Class imbalance remains a major challenge in ensuring fair and accurate classification.

2.5.4 Linguistic and Cultural Nuances

Code-mixed text often incorporates cultural expressions, idiomatic phrases, and sarcasm, making it difficult for models to detect offensive content without explicit hate words. Furthermore, multilingual grammatical structures add complexity, as certain words and phrases can change meaning based on context.

2.5.5 Extracting Relevant Features

Identifying the most relevant features for offensive language detection is particularly challenging in code-mixed text, where negations, implicit sentiments, and mixed grammar structures affect text representation [12, 13]. Traditional methods such as n-grams and TF-IDF fail to capture deeper linguistic patterns, while deep learning-based embeddings often require large amounts of labeled data [8, 11, 13].

2.5.6 Noise in Social Media Text

Social media text is inherently noisy, containing emojis, hashtags, user mentions, and unstructured data, which increases the burden of data preprocessing. Filtering these elements without losing semantic meaning is a significant challenge. While some studies replaced emojis with text representations [6, 10, 11] but results varied across datasets.

2.5.7 Model Performance and Interpretability

Despite improving hate speech classification, transformer models like mBERT, MuRIL, and MPNet lack transparency, raising concerns about bias, interpretability, and ethical implications in decision-making. Additionally, fine-tuning complex models requires high computational resources, making it difficult to implement for long text sequences and large-scale datasets.

2.5.8 Cross-Dataset Generalisation

Models trained on one dataset often struggle with cross-dataset generalisation, showing reduced performance when tested on datasets with different characteristics. Some researchers applied domain adaptation techniques, but the improvements were marginal, highlighting the need for better dataset diversity and feature selection strategies [10].

2.5.9 Real-World Scenarios

In real-world applications, hate speech detection extends beyond just text-based analysis. Social media platforms often include multimodal content, such as images, videos, and memes, that contribute to the overall meaning of a post. However, most current NLP models are limited to text-based detection and fail to capture context from multimedia sources [11]. Integrating multimodal learning approaches remains an open challenge for improving real-world hate speech detection.

TABLE I
PERFORMANCE SUMMARY OF MODELS

Models Used	Best Performance	Features Used
Naive Bayes, SVM, RF [13]	RF: 95.6% Accuracy	Unigram + SentiWordNet + Negation Words
Ensemble of CNN and LSTM [7]	F1-Score: 0.93	Transliteration, Word Embeddings
LSTM, Dense Network [8]	F1-Score: 0.56	Keras Embedding, Word2Vec
mBERT + Transliteration [9]	F1-Score: 0.94	Random Transliteration
Multinomial NB, SVM, RF [12]	SVM: 77%	TF-IDF, n-grams
RoBERTa, XLNet [10]	XLNet: 0.88	Layer Freezing Strategies, Learning Rate
Deep Ensemble Framework [11]	F1-Score: 80.2%	Weighted Outputs, Custom Weights and Averaging Techniques
MPNet, CNN [15]	Accuracy: 98%	MPNet Embeddings
BERT-base-multilingual-cased [14]	Accuracy: 84.62% (Bigram)	Contextual Embeddings, n-gram analysis
MuRIL Model + SVM + RBF [6]	F1-Score: 96%	Weighted Embeddings

Figure 2.1: Performance Summary

2.6 Conclusion

This paper reviews recent research on offensive language detection in Manglish (Malayalam-English code-mixed text), examining key tasks, datasets, and methodologies while highlighting challenges for future research. Detecting offensive language in code-mixed

texts requires advanced feature extraction and preprocessing techniques. Although challenges such as mixed-script handling, limited labeled datasets, class imbalance, and linguistic complexities—such as idiomatic expressions and sarcasm—persist, significant progress has been made. Advancements in deep learning and ensemble models have yielded promising results, while modern approaches, including MuRIL embeddings and transformer-based models, have greatly enhanced contextual understanding. As the field continues to evolve, future research should focus on refining models, increasing dataset diversity, and addressing critical ethical concerns such as bias and interpretability.

Chapter 3

System Design

The goal of the design phase is to devise a solution to the issue that the requirement specification has identified. The development team uses this phase as a model to direct them as they build the real-world software product. The most important component determining the software's quality is the system's design, which also has a significant impact on the latter stages, particularly testing and maintenance. System design and detailed design are the two distinct phases that make up the design process.

3.1 Proposed System

The proposed system enhances offensive language detection in Malayalam-English code-mixed text by integrating both NLP and ML techniques. Unlike traditional systems that classify each comment individually, our model incorporates a robust classification approach leveraging deep learning with focal loss to mitigate class imbalance. The workflow includes the following steps:

- **Data Collection:** We utilize the DravidianCode-Mix Malayalam code-mixed dataset, which contains individual comments labeled as either *offensive* or *non-offensive*. Additionally, we expand this dataset with contemporary comments to capture evolving trends in slang and offensive language patterns.
- **Preprocessing:** The preprocessing steps include stop word removal and tokenization, while punctuation and emoticons are retained to preserve the emotional tone of the comments.
- **Embedding Generation:** We use BERT-based embeddings, incorporating class weights to balance the dataset. These embeddings are generated from pre-trained transformer models to ensure a deeper contextual understanding of the text.
- **Classification:** Two different approaches are implemented:
  - Four classifiers—Random Forest, Logistic Regression, KNN, and Naïve Bayes were applied with majority voting to determine the final classification.
  - A separate Multilayer Perceptron (MLP) model with focal loss is used to directly classify the embeddings.

- **Output Interface:** The system includes a web-based user interface that allows users to input a comment and view classification results. Both classification methods—majority voting from traditional classifiers and the standalone MLP—are integrated for comparison and evaluation.

Advantages:

- The use of focal loss effectively addresses class imbalance, improving detection of offensive comments.
- Dataset augmentation with recent data helps the model stay up-to-date with evolving slang and offensive language trends.

Disadvantages:

- The expanded dataset and MLP training increase computational requirements, potentially slowing down processing.
- Hyperparameter tuning for MLP with focal loss requires additional optimization effort for improved accuracy.

This system aims to address gaps in current offensive language detection models by leveraging a balanced and trend-adaptive approach for Malayalam-English code-mixed text.

3.1.1 Problem Statement

The objective of this project is to develop an effective system for detecting offensive language in Malayalam-English code-mixed text. Given the widespread use of code-mixing in social media and online conversations, it is essential to identify and classify harmful or inappropriate content within these bilingual interactions. This project aims to improve moderation by leveraging advanced embedding techniques and classification strategies to enhance the accuracy of offensive language detection.

A key challenge in this task is the significant class imbalance present in the dataset, where non-offensive comments greatly outnumber offensive ones, leading to potential bias in classification. To address this, we explore multiple approaches: initially applying BERT embeddings with class-weighted classifiers and majority voting, followed by a Multilayer Perceptron (MLP) trained with focal loss to better handle imbalanced data. Additionally, dataset augmentation with contemporary comments helps adapt the model to evolving slang and offensive language patterns.

By integrating these methodologies, our system aims to provide a more reliable and context-aware classification of offensive language, thereby contributing to the moderation of online discourse in a more robust and adaptive manner.

3.1.2 System Model

3.1.2.1 Block Diagram

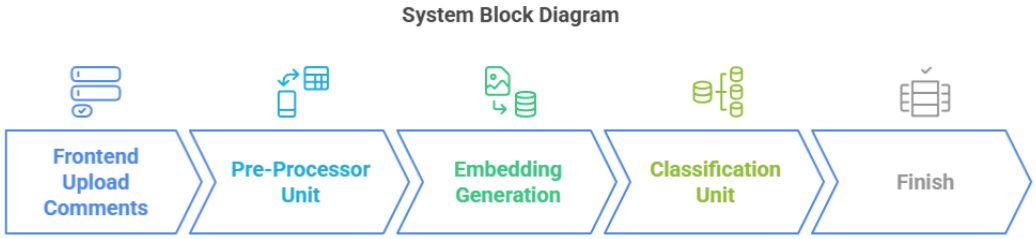

Figure 3.1: System block Diagram

The block diagram outlines the sequence of operations within the system, represented by the following components:

- **Start:** This is the initial state of the application where the system is ready to accept input from the user.
- **Frontend - Upload Comments:** In this stage, users can input or upload comments for analysis via a user-friendly interface. The comments can consist of code-mixed Malayalam-English text.
- **Preprocessing Unit:** This unit prepares the uploaded comments for analysis by removing stop words while retaining punctuation and emoticons, and performing tokenization to convert text into a structured format suitable for further processing.
- **Embedding Generation - Context Analysis:** Here, embeddings for the preprocessed comments are generated using the BERT model which captures context bidirectionally.
- **Classification Unit:** In this component, the generated embeddings, along with their respective class weights, are processed by a machine learning classifier (e.g., SVM or Random Forest or MLP) to classify the comments as offensive or non-offensive.
- **Finish:** This final state concludes the process, delivering the classification result back to the user through the interface, indicating whether the input text is offensive or non-offensive.

3.1.2.2 Flow chart

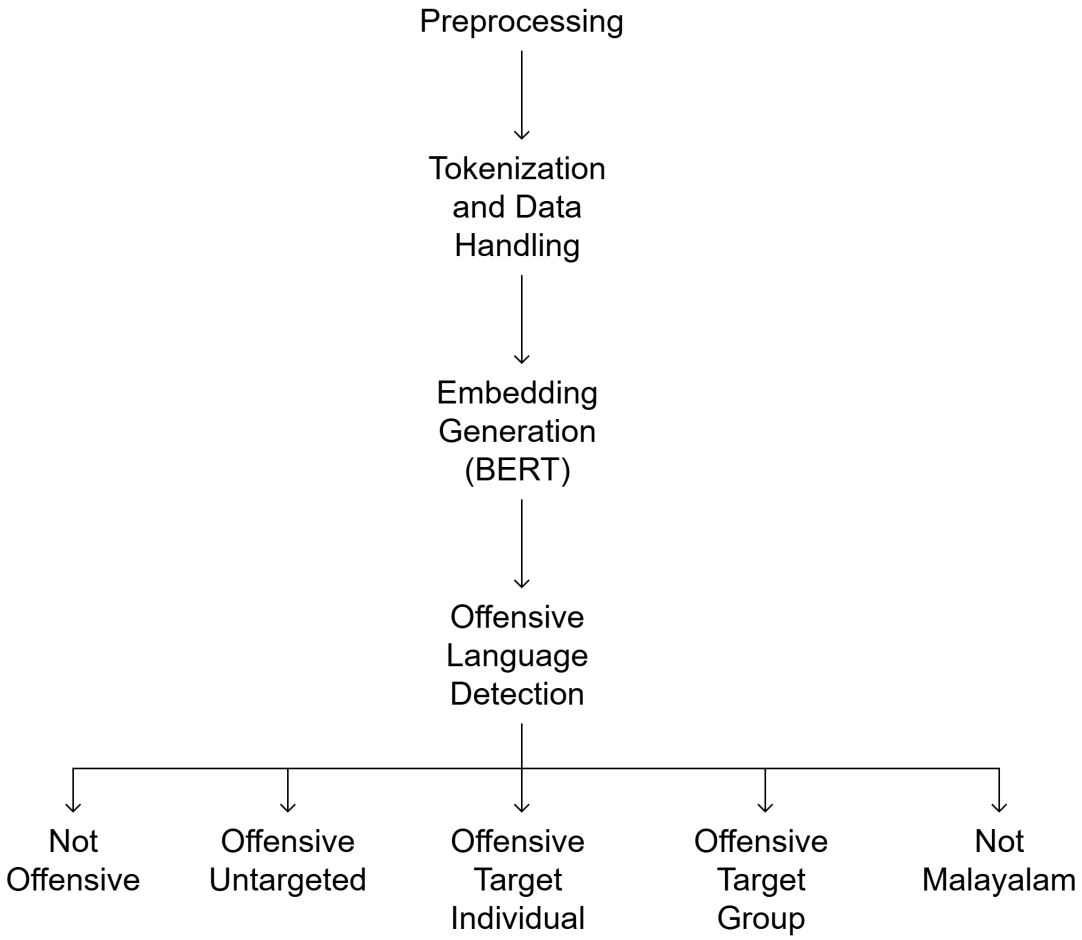

Figure 3.2: Flow chart

This flowchart outlines the process of offensive language detection process.

- **Preprocessing:** This step involves cleaning the Malayalam-English code-mixed text. Stop words are removed while punctuation and emoticons are retained to preserve sentiment and context.
- **Tokenization and Data Handling:** Each comment is tokenized and converted into a format suitable for embedding generation. This stage ensures that the input data is properly structured for processing by the embedding model.
- **Embedding Generation (BERT):** The system employs BERT-based embeddings (such as MuRIL) to convert text into numerical representations. These embeddings capture the semantic and syntactic nuances of code-mixed Malayalam-English text.

- **Offensive Language Detection:** Initially, the project used BERT embeddings with class weights and majority voting among four classifiers (Random Forest, Logistic Regression, KNN, and Naïve Bayes). Due to class imbalance issues, an alternative approach was adopted: using a Multilayer Perceptron (MLP) with focal loss, improving classification performance. The model predicts whether a comment is offensive or non-offensive.
- **Classification Output:** The model categorizes the input into five classes:
  - **Not Offensive:** The text does not contain any offensive language.
  - **Offensive Untargeted:** The comment is offensive but does not target a specific individual or group.
  - **Offensive Target Individual:** The offensive comment is directed at a specific person.
  - **Offensive Target Group:** The offensive comment targets a particular group based on identity, ideology, or background.
  - **Not Malayalam:** The comment is in a language other than Malayalam.

3.2 System Design

System Design aims to identify the modules that should be present in the system, the specification of these modules and how they interact with each other to produce the desired results. At the end of the system design, all the major data structures, file formats, output formats and major modules in the system and their specifications are decided.

The aim of this system design is to develop a robust framework for detecting offensive language in Malayalam-English code-mixed text. This project leverages both Natural Language Processing (NLP) and Machine Learning (ML) techniques to create a comprehensive solution that accurately classifies comments as either offensive or non-offensive.

The proposed system will consist of several key modules, each designed to perform specific functions while seamlessly interacting with one another to achieve the desired outcome. The design focuses on the following aspects:

3.2.1 Module 1: Dataset Collection and Management

The core of our project relies on a well-curated dataset sourced from the DravidianCodeMix Malayalam code-mixed corpus, which comprises individual comments that may span multiple lines. Each comment is annotated with labels indicating whether it is offensive or non-offensive. To increase the dataset's relevance and improve model responsiveness to

contemporary usage, we augment this foundation with additional data that captures recent language trends on social media.

3.2.2 Module 2: Preprocessing

The Preprocessing Module is a critical component in the workflow of the offensive language detection system. It prepares raw text data for subsequent analysis by applying a series of transformations that enhance the effectiveness of the BERT model in generating meaningful embeddings. This module encompasses the following key steps:

- Text Normalization: Normalize text to ensure consistency by converting it to lowercase or uppercase, reducing complexity, and treating words uniformly regardless of case.
- Removal of Stop Words: Eliminate common stop words (e.g., "and," "the," "is") to focus on more informative words, enhancing the signal-to-noise ratio in the data.
- Retaining Punctuation and Emoticons: Retain punctuation marks and emoticons, as they provide important sentiment cues and context, which are vital for understanding code-mixed texts.
- Tokenization: This process involves splitting the text into smaller units called tokens (typically words or phrases). Tokenization allows the model to analyze the structure and semantics of the comments more effectively.

3.2.3 Module 3: Embedding Generation

This module focuses on generating embeddings for each comment in the dataset using BERT-based models, specifically tailored for Malayalam-English code-mixed text. The embeddings plays a crucial role in capturing the semantic and syntactic structure of the text for improved classification performance. The key aspects of this process are as follows:

1. **Utilization of BERT-based Embeddings:** To effectively process code-mixed Malayalam-English comments for offensive language detection, the system leverages BERT-based sentence embeddings as a means of converting unstructured text into rich numerical representations.

3.2.3.1 Embedding Generation using BERT

The model uses the `bert-base-multilingual-cased` variant from the `SentenceTransformers` library, which is specifically designed to support multilingual input, including Malayalam and English. Each input comment is tokenized and

passed through BERT, which outputs a high-dimensional dense vector (typically 768-dimensional) that captures the semantic meaning of the sentence in a contextualized manner. These sentence embeddings serve as feature vectors for downstream classifiers.

3.2.3.2 Initial Classification Approach using Ensemble of Traditional ML Models

The extracted BERT embeddings were first fed into a set of traditional machine learning classifiers:

- Random Forest
- Logistic Regression
- K-Nearest Neighbors (KNN)
- Naïve Bayes

Each of these models was trained using the BERT embeddings as input features. To address the class imbalance problem (where non-offensive examples significantly outnumber offensive ones), class weights were computed using `sklearn.utils.class_weight.compute_class_weight()` and applied to the training process. This gave higher importance to the minority (offensive) classes, helping the classifiers avoid bias toward the majority class.

3.2.3.3 Majority Voting Ensemble

To increase robustness, the outputs from the four classifiers were combined using a majority voting scheme. In this ensemble method:

(a) Each classifier independently predicts the class label for a given comment.

(b) The final prediction is the label that receives the majority of votes across the classifiers.

This approach is designed to reduce variance and enhance generalization by aggregating the strengths of individual models.

3.2.3.4 Limitations of the Initial Approach

Despite using BERT embeddings and class weights, the ensemble faced significant limitations due to:

- **Extreme Class Imbalance**: The dataset was heavily skewed, with far more non-offensive samples, leading to poor performance in detecting offensive content — especially in minority subcategories like `target_group` or `untargeted`.
- **Naïve Bayes Weakness**: Naïve Bayes assumes feature independence, which is incompatible with high-dimensional, dense embeddings like those from BERT, resulting in poor accuracy (as low as ∼20%).
- **Overfitting in KNN**: KNN performed well on seen data but struggled to generalize on external or unseen samples due to its instance-based nature.
- **Model Inconsistencies**: While Random Forest and Linear Regression showed decent individual accuracies, their predictions were not always consistent, weakening the overall majority vote.

As a result of these challenges, the ensemble was eventually replaced by a more effective deep learning-based model—a Multilayer Perceptron (MLP) trained with Focal Loss—to handle class imbalance more directly and improve performance on underrepresented classes.

2. **Integration with Multilayer Perceptron (MLP):** To address the persistent class imbalance challenges in detecting offensive language within code-mixed Malayalam-English comments, a deep learning approach using a Multilayer Perceptron (MLP) was adopted. This method significantly enhanced classification performance over traditional models.

3.2.3.5 Model Architecture

The MLP classifier is a feedforward neural network that accepts BERT-based sentence embeddings as input. The architecture consists of the following layers:

- **Input Layer:** Accepts 768-dimensional BERT embeddings.
- **Hidden Layer 1:** A fully connected layer with 256 units followed by ReLU activation.
- **Hidden Layer 2:** A fully connected layer with 128 units followed by ReLU activation.
- **Output Layer:** A fully connected layer with the number of output units equal to the number of target classes, followed by a softmax (applied via loss function).

The model is defined using PyTorch's `nn.Module` interface, allowing flexibility for loss computation and backpropagation.

3.2.3.6 Handling Class Imbalance with Focal Loss

To mitigate the issue of class imbalance, the model uses **Focal Loss** as the objective function instead of standard CrossEntropy loss. Focal Loss dynamically scales the contribution of each training example based on the prediction confidence, focusing more on hard-to-classify (typically minority class) examples.

The focal loss function is defined as:

$$\mathcal{L}_{\text{focal}} = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

where:

- p_t is the model's estimated probability for the true class,
- α_t is a weighting factor for class t (obtained using `sklearn`'s `compute_class_weight`),
- γ is a focusing parameter that reduces the relative loss for well-classified examples (set to 2).

3.2.3.7 Training Setup

- **Optimizer:** Adam optimizer with a learning rate of 1×10^{-4}.
- **Epochs:** The model was trained for 80 epochs.
- **Batch Size:** 64
- **Device:** Training was accelerated using GPU (if available), utilizing PyTorch's `.to(device)` method.

3.2.3.8 Performance

The integration of the MLP classifier trained with Focal Loss led to a significant improvement in model accuracy. On the test set, the model achieved:

- **Accuracy:** 87%
- **F1-score (Macro):** Notably improved, especially for underrepresented classes.

This approach proved more effective than the ensemble of traditional models, as the MLP could better capture non-linear patterns in the data and generalize well, particularly for offensive comment subcategories.

3. **Model Deployment and Frontend Integration:** The system currently integrates these models into a simple front-end interface. The four classifiers operate under a majority voting scheme, while the MLP model is deployed separately for independent evaluation.

3.2.4 Use Case Diagram

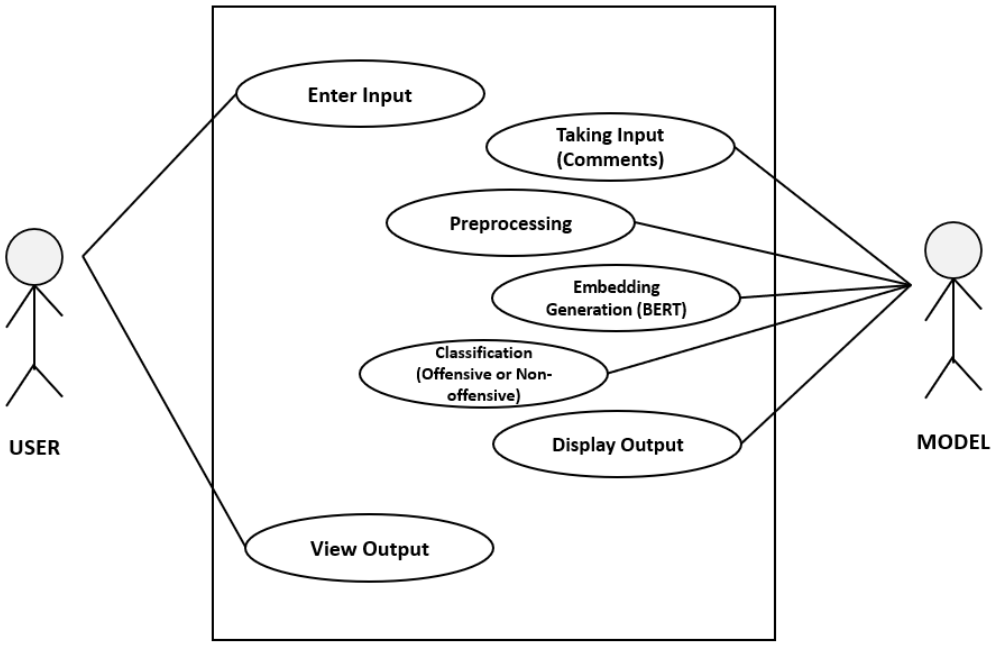

Figure 3.3: Use case Diagram

This use case diagram illustrates the interaction between the User and the Model in a system designed for detecting offensive language in English-Malayalam code-mixed text. The User initiates the process by entering input, which the Model then takes in for processing. The system performs several tasks: preprocessing the input data, generating embeddings, and classifying the text as either offensive or non-offensive. Finally, the system displays the classification output, which the User can view. This diagram highlights the collaborative roles of the User and the Model in achieving accurate text classification.

3.3 Detailed Design

In the detailed design, the abstract concepts and specifications outlined in the architectural design are elaborated upon to create a comprehensive blueprint for the software's implementation. It focuses on the specific implementation details, data structures, algorithms, and interactions required to build the system effectively.

3.3.1 Algorithm

This algorithm classifies the texts as offensive or non-offensive. It processes each line by first performing preprocessing tasks such as tokenization. It then generates embeddings which is then integrated with class weights. After generating the embeddings, the algorithm checks for offensiveness and outputs the classification result for each line, repeating this until all lines have been analyzed.

Algorithm 1 Offensive Language Detection in Malayalam-English Code-Mixed Text

procedure OFFENSIVELANGDETECTION
  Input: Comment s
  Output: Offensive/Non-offensive for s

  //**Preprocessing**
  for comment s **do**
    Retain punctuation and emoticons in s
    Tokenize s for further processing
  end for

  //**Embedding Generation**
  for preprocessed comment s **do**
    Generate embeddings for s using BERT model
    Apply cost-sensitive learning to handle class imbalance in labels
  end for

  //**Classification**
  for embedded comment s **do**
    Insert the embedding and class weights into:
      - ML classifier (Random Forest, Logistic Regression, Naive Bayes, KNN)
      - MLP with focal loss
    Predict if s is offensive or non-offensive
  end for

  //**Output**
  for comment s **do**
    Display result (Offensive/Non-offensive) for s
  end for
end procedure

Chapter 4

Implementation

The implementation phase is where the designed system is transformed into a functional model through coding and module integration. This stage ensures that all components work together as planned, following the specifications outlined in the design phase. The primary goal is to develop a reliable and efficient system while maintaining code simplicity and clarity to facilitate future testing and debugging.

4.1 Tools Used

The various tools used for implementing the system are mentioned in this chapter. The tools used can be broadly classified into two categories:

- Front-End Tools
- Deep Learning and Visualization Libraries

4.1.1 Front-End Tools

The front-end tools were used to develop the user interface, allowing users to input text for classification and view the results. These tools facilitated seamless interaction between the user and the backend model, ensuring a smooth user experience. The front-end tools used to implement the system are as follows:

4.1.1.1 HTML, CSS, and JavaScript

The system's web interface was developed using HTML, CSS, and JavaScript. These technologies were used to create the structure, design, and interactive elements of the application, allowing users to input text and view classification results.

4.1.1.2 Flask

Flask was used as the backend framework to handle user input, process API requests, and serve responses from the trained model. It enabled seamless integration between the web interface and the deep learning model.

4.1.2 Deep Learning and Visualization Libraries

The deep learning and visualization libraries used to implement the system are mentioned in the following sections.

4.1.2.1 Bidirectional Encoder Representations from Transformers (BERT)

BERT was used for text representation and feature extraction. It was applied in the preprocessing stage to generate contextual embeddings for Malayalam-English code-mixed comments before classification. Unlike traditional word embeddings, BERT captures the contextual meaning of words based on their surroundings, improving classification accuracy.

4.1.2.2 Scikit-learn

Scikit-learn was used for preprocessing tasks, encoding labels, and evaluating traditional machine learning classifiers. It provides various utilities for implementing classifiers like Logistic Regression, SVM, and Random Forest.

4.1.2.3 NLTK and Transformers

NLTK (Natural Language Toolkit) was used for text preprocessing tasks such as tokenization and stopword removal. The Transformers library from Hugging Face was used to load pre-trained BERT models and perform text tokenization.

4.1.2.4 Matplotlib and Seaborn

Matplotlib and Seaborn were used for visualization. These libraries were applied to generate data distribution graphs, confusion matrices, and performance evaluation plots to analyze the model's effectiveness.

4.2 System Implementation

Developing a machine learning-based Malayalam-English code-mixed offensive language detection system is a structured and iterative process. The system first acquires and preprocesses the dataset, which includes text normalization, tokenization, and feature extraction using BERT. In our project, we have implemented two classification approaches:

1. BERT with Traditional Classifiers – BERT embeddings are used as input features for machine learning models like Logistic Regression, KNN, Naive Bayes and Random Forest.

2. BERT with MLP and Focal Loss – A Multi-Layer Perceptron (MLP) model is trained using BERT embeddings, with focal loss applied to handle class imbalance effectively.

The performance of these models is evaluated using various metrics, and fine-tuning is performed to optimize accuracy. Once an optimal model is obtained, it is deployed for real-world usage as a robust classification system.

4.2.1 Data Preprocessing

4.2.1.1 Data Collection

The core of our project relies on a well-curated dataset sourced from the DravidianCodeMix Malayalam corpus, which contains individual comments labeled based on their offensiveness. Each comment in the dataset is categorized into one of five distinct classes:

- Not Offensive – Neutral or non-offensive comments.
- Offensive Untargeted – Offensive language without a specific target.
- Offensive Targeted Individual – Offensive comments directed at an individual.
- Offensive Targeted Group – Offensive comments targeting a group or community.
- Not Malayalam – Comments that do not belong to the Malayalam code-mixed category.

This structured dataset provides a comprehensive representation of offensive language patterns, enabling the model to distinguish between different types of offensive content with high accuracy.

4.2.1.2 Preprocessing

The quality and effectiveness of an NLP model depend significantly on the preprocessing of input data. For this project, which focuses on detecting offensive language in Manglish (Malayalam-English code-mixed) text using a BERT-based model, preprocessing is primarily handled by the model's internal mechanisms.

1. Tokenization
    - The `bert-base-multilingual-cased` model internally tokenizes the text before embedding generation.
    - Tokenization splits words into smaller subword units using WordPiece tokenization.

- Example: `"hello everyone!!"` might be tokenized into `["hello", "every", "one", "!", "!"]`.
- As a cased model, it preserves uppercase/lowercase distinctions.

2. Maximum Sequence Length Handling
   - The line `trans_model.max_seq_length = 128` ensures that input text is truncated if it exceeds 128 tokens.
   - If a sentence is longer, it is truncated at the end to fit within this limit.
3. Label Encoding & Class Weight Computation
   - Labels are encoded using `LabelEncoder()`, transforming textual labels into numeric values.
   - Class weights are computed using `sklearn.utils.class_weight.compute_class_weight` to handle class imbalance.

4.2.2 Model Development

In this project, two different approaches were implemented to detect offensive language in Manglish (Malayalam-English code-mixed text). The first approach involved using BERT embeddings with traditional classifiers, where pre-trained BERT representations were extracted and fed into conventional machine learning models such as Logistic Regression, K-Nearest Neighbors (KNN), Naïve Bayes, and Random Forest. The second approach involved training a Multi-Layer Perceptron (MLP) model with Focal Loss, leveraging BERT embeddings as input features while addressing class imbalance more effectively.

4.2.2.1 BERT with Traditional Classifiers

This approach utilizes BERT embeddings as feature representations for traditional machine learning classifiers. BERT's pre-trained model generates context-aware word embeddings, which are then used as input features for classification models.

- Feature Extraction Using BERT Embeddings:

  The first step in this approach involved extracting contextualized word embeddings from the pre-trained BERT-base-multilingual-cased model. Since BERT captures word meanings based on surrounding context, its embeddings provide rich semantic representations that significantly enhance classification performance. Instead of using raw text or traditional word vectors, each comment in the dataset was passed through BERT's transformer layers, resulting in high-dimensional feature vectors. These

embeddings were then used as input for multiple classifiers to distinguish between offensive and non-offensive comments effectively.

To ensure a balanced learning process, class weights were incorporated during training. Given the skewed class distribution in the dataset, where non-offensive comments were significantly more prevalent, class weights were calculated using the inverse frequency method. This ensured that underrepresented offensive classes were assigned higher importance during classification, preventing the models from being biased toward majority class predictions.

- Traditional Machine Learning Classifiers

  Once BERT embeddings were extracted, they were used as input features for four traditional machine learning classifiers: Random Forest, Logistic Regression, K-Nearest Neighbors (KNN), and Naïve Bayes. Each classifier has its own strengths and weaknesses in handling classification tasks, particularly when dealing with high-dimensional feature representations like BERT embeddings.

  1. Random Forest: It is a machine learning algorithm that operates by constructing multiple decision trees during training and aggregating their predictions for classification. It improves accuracy by reducing overfitting, as the combination of multiple trees provides a more generalized decision boundary. Each tree is trained on a randomly selected subset of the data, and the final prediction is determined by majority voting among all trees.

     For our classification task, Random Forest was implemented with 100 decision trees, each trained on random feature subsets to enhance generalization. Class weights were applied to address the class imbalance in the dataset, ensuring that underrepresented classes received appropriate attention. The model demonstrated strong performance, achieving 89% accuracy, making it the most effective among the traditional classifiers.

```
Evaluation Scores:
Macro Metrics:
Accuracy: 0.892
Precision: 0.973
Recall: 0.414
F1 Score: 0.464

Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      1765
           1       1.00      0.45      0.62        31
           2       1.00      0.00      0.00        45
           3       1.00      0.00      0.00       103
           4       0.98      0.62      0.76       157

    accuracy                           0.89      2101
   macro avg       0.97      0.41      0.46      2101
weighted avg       0.90      0.89      0.86      2101
```

Figure 4.1: Accuracy - Random Forest

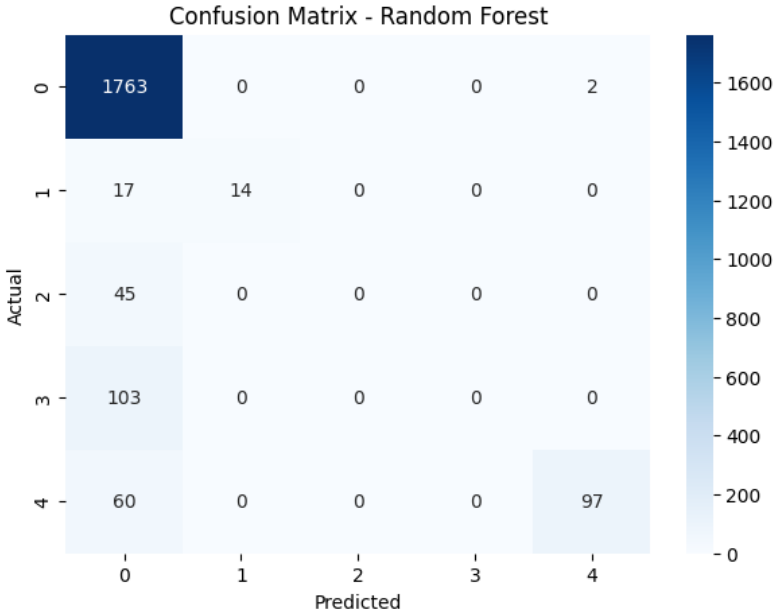

Figure 4.2: Confusion Matrix - Random Forest

2. Logistic Regression: It is a linear classification model that predicts the probability of a given input belonging to a specific class using the sigmoid function. The model finds the optimal decision boundary by learning the relationship between input features and output labels through maximum likelihood estimation. It is commonly used for binary and multi-class classification tasks.

   Logistic Regression demonstrated a 73% accuracy, making it a fairly reliable model for offensive language detection in Manglish text. Its ability to perform well despite its simplicity highlights the effectiveness of BERT embeddings in capturing meaningful linguistic patterns.

```
Evaluation Scores:
Macro Metrics:
Accuracy: 0.736
Precision: 0.408
Recall: 0.607
F1 Score: 0.454

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.75      0.84      1765
           1       0.16      0.58      0.25        31
           2       0.13      0.38      0.19        45
           3       0.20      0.45      0.28       103
           4       0.60      0.88      0.71       157

    accuracy                           0.74      2101
   macro avg       0.41      0.61      0.45      2101
weighted avg       0.86      0.74      0.78      2101
```

Figure 4.3: Accuracy - Logistic Regression

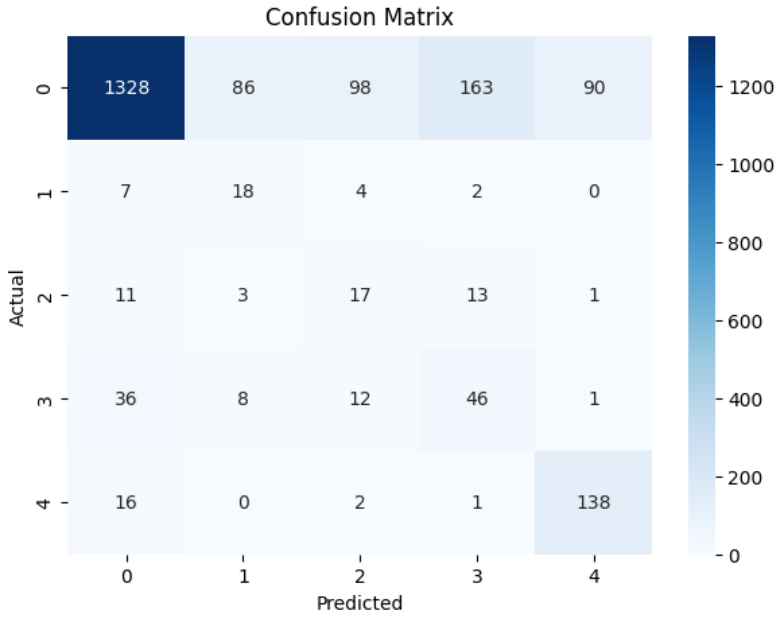

Figure 4.4: Confusion Matrix - Logistic Regression

3. K-Nearest Neighbors (KNN): It is a non-parametric, instance-based learning algorithm that classifies data points based on the majority vote of their K nearest neighbors in the feature space. It is effective for pattern recognition and works without explicit training, making it highly flexible. However, its performance depends on the choice of K, and its high computational cost makes it inefficient for large-scale datasets.

   Since BERT embeddings effectively captured the semantic and contextual meaning of Manglish text, the KNN classifier was able to leverage these rich representations to group similar offensive comments together. The model performed well, achieving an accuracy of 87%, indicating that the nearest-neighbor approach

effectively identified offensive language patterns in code-mixed text. The ability of KNN to classify text based on proximity in the embedding space was beneficial, especially for detecting nuanced offensive expressions.

```
Evaluation Scores:
Macro Metrics:
Accuracy: 0.874
Precision: 0.621
Recall: 0.374
F1 Score: 0.401

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.99      0.93      1765
           1       0.74      0.45      0.56        31
           2       0.00      0.00      0.00        45
           3       0.18      0.05      0.08       103
           4       0.92      0.76      0.83       157

   micro avg       0.87      0.90      0.88      2101
   macro avg       0.55      0.45      0.48      2101
weighted avg       0.83      0.90      0.86      2101
```

Figure 4.5: Accuracy - KNN

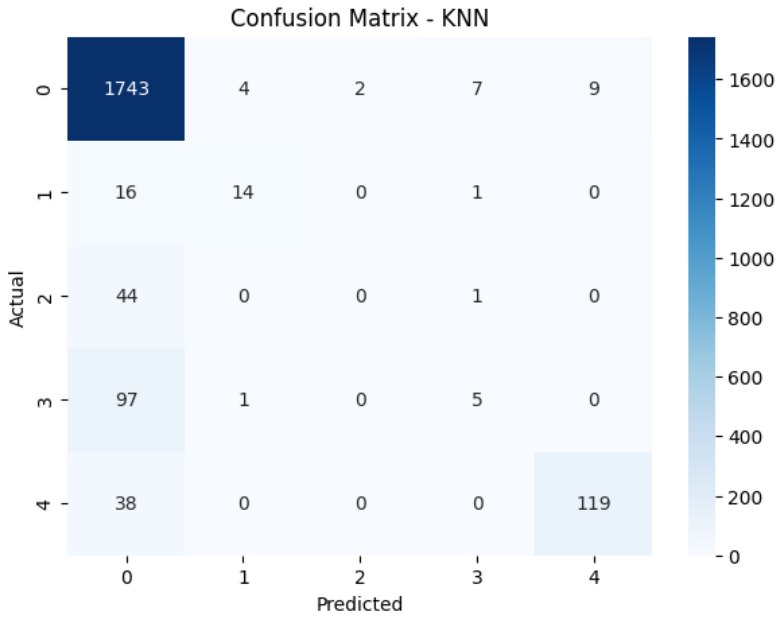

Figure 4.6: Confusion Matrix - KNN

4. The Naive Bayes classifier: It is a probabilistic machine learning algorithm based on Bayes' theorem, assuming independence between features. Despite this simplification, it is widely used in text classification, spam filtering, and sentiment analysis due to its efficiency and ability to handle high-dimensional data. While computationally efficient and effective for small datasets, Naive Bayes struggles with context-dependent language models due to its independence assumption.

The embeddings, which capture rich contextual information, were passed into the classifier; however, due to the Naive Bayes assumption of feature independence, the model struggled to effectively learn the relationships between offensive words and their context. This resulted in a poor classification performance, with an accuracy of only 20%, making it the least effective classifier among the four used in this approach. Unlike traditional text classification tasks where Naive Bayes performs well with term frequency-based features, our implementation showed that it does not adapt effectively to dense numerical representations like BERT embeddings, where dependencies between dimensions are crucial.

```
Evaluation Scores:
Macro Metrics:
Accuracy: 0.205
Precision: 0.288
Recall: 0.423
F1 Score: 0.190

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.14      0.24      1765
           1       0.03      0.77      0.06        31
           2       0.08      0.13      0.10        45
           3       0.04      0.22      0.06       103
           4       0.34      0.85      0.49       157

    accuracy                           0.21      2101
   macro avg       0.29      0.42      0.19      2101
weighted avg       0.83      0.21      0.25      2101
```

Figure 4.7: Accuracy - Naive Bayes

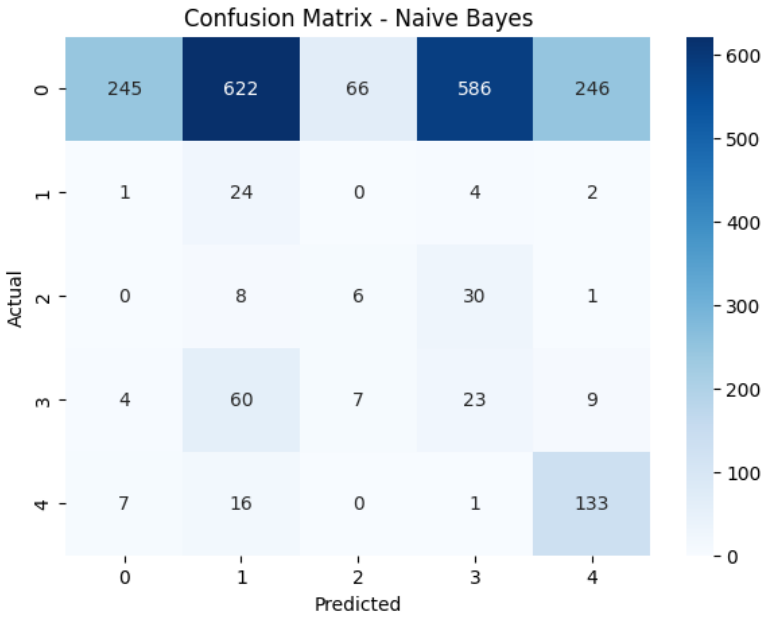

Figure 4.8: Confusion Matrix - Naive Bayes

- Majority Voting Scheme for Final Classification

Since individual classifiers had varying performances, a majority voting approach was implemented. In this scheme, each classifier provided an independent prediction for a given comment. The final classification decision was made based on the majority agreement among the four classifiers. This method helped balance the strengths and weaknesses of each classifier, improving the overall robustness of the system. Additionally, a frontend interface was developed to integrate these models. The four traditional classifiers, along with the majority voting scheme, were deployed to provide real-time offensive language detection, making the system accessible and user-friendly.

While the first approach utilizing BERT embeddings with traditional classifiers provided a structured way to detect offensive language in Manglish, it faced challenges due to class imbalance. Despite applying class weights to mitigate this issue, the classifiers exhibited varying levels of accuracy, with Random Forest (89%) and KNN (87%) performing significantly better than Logistic Regression (73%) and Naive Bayes (20%). However, the reliance on majority voting and the limited ability of traditional classifiers to capture deep contextual relationships in text led to suboptimal performance, especially for minority classes. Given these limitations, it became essential to explore an alternative approach that could leverage BERT embeddings more effectively while addressing class imbalance more robustly.

4.2.2.2 BERT with Multi-Layer Perceptron (MLP) and Focal Loss

Given the limitations observed in the first approach, particularly due to class imbalance and the inability of traditional classifiers to fully capture the contextual depth of code-mixed text, a more effective method was required. While BERT embeddings provided rich feature representations, the classifiers struggled with minority class predictions, affecting overall performance. To address this issue, we implemented a second approach that leverages a Multi-Layer Perceptron (MLP) model with focal loss, which enhances learning in imbalanced datasets and refines classification accuracy.

- Multi-Layer Perceptron (MLP) Model

  MLP is a type of deep neural network composed of multiple layers of neurons that process input features through weighted connections and activation functions. Unlike traditional classifiers, MLP can learn complex non-linear relationships and capture intricate patterns within the dataset. In this approach, BERT embeddings were fed into the MLP model, which was trained to differentiate between offensive and non-offensive text more effectively. By employing multiple hidden layers and non-linear activation functions, the model was able to extract deeper semantic patterns, improving classification robustness.

- Focal Loss for Class Imbalance

A key challenge in offensive language detection is the heavy class imbalance, where non-offensive comments significantly outnumber offensive ones. Traditional loss functions like cross-entropy tend to bias the model towards the dominant class, resulting in poor predictions for minority classes. To mitigate this issue, we incorporated focal loss, which assigns lower weights to well-classified samples and higher weights to misclassified ones. This adaptive weighting mechanism helped the model focus on hard-to-classify instances, leading to better learning for underrepresented classes and an overall improvement in classification performance.

- Implementation and Performance

  The MLP model was trained using BERT embeddings with class weights, running for 80 epochs with a batch size of 64. This approach effectively balanced learning across all categories, achieving a consistent accuracy of 87%. Compared to the traditional classifiers, the MLP model demonstrated superior generalization and stability in classification. This approach which utilized the MLP model, was also integrated with the frontend, facilitating real-time testing with comments.

	precision	recall	f1-score	support
Not_offensive	0.91	0.96	0.94	1765
Offensive_Untargetede	0.41	0.52	0.46	31
Offensive_target_insult_Group	0.24	0.13	0.17	45
Offensive_target_insult_individual	0.32	0.17	0.23	103
not-malayalam	0.84	0.87	0.85	157
unknown	0.00	0.00	0.00	52
accuracy			0.87	2153
macro avg	0.45	0.44	0.44	2153
weighted avg	0.83	0.87	0.85	2153

Figure 4.9: Accuracy - MLP with focal loss

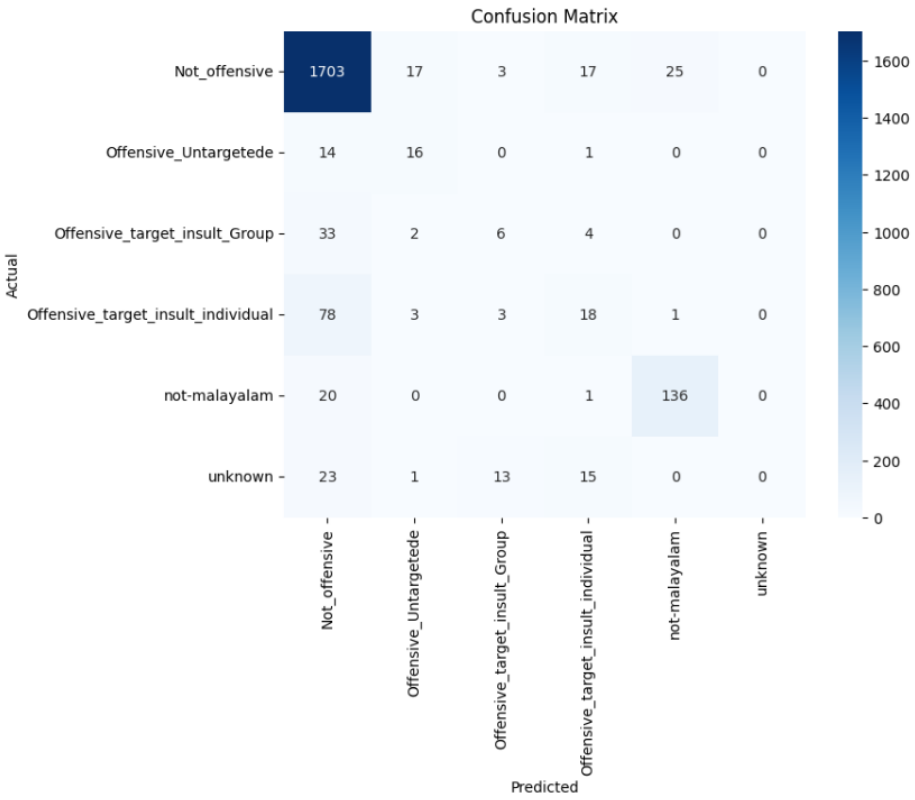

Figure 4.10: Confusion Matrix - MLP with focal loss

By addressing the limitations of traditional classifiers, the MLP model with focal loss provided a more robust and reliable solution for offensive language detection in Manglish (Malayalam-English code-mixed text). The integration of deep learning techniques not only enhanced classification accuracy but also ensured better handling of class imbalance, making the second approach a more effective solution for this task.

4.2.3 Results Analysis

A summary of the training results is given in the following table. From the table, it is evident that the Random Forest classifier achieved the highest accuracy among traditional machine learning models, while the Multi-Layer Perceptron (MLP) demonstrated competitive performance with better handling of class imbalance.

Table 4.1: Comparison of Classification Approaches

Approach	**Classifier/Model**	**Accuracy (%)**
BERT with Traditional Classifiers	Random Forest	89
	Logistic Regression	73
	K-Nearest Neighbors (KNN)	87
	Naïve Bayes	20
BERT with MLP and Focal Loss	Multi-Layer Perceptron (MLP)	87

4.3 Summary

This chapter details the development of a Malayalam-English code-mixed offensive language detection system using deep learning and machine learning approaches. BERT embeddings were utilized for feature extraction, enabling rich contextual understanding of the text. Two classification approaches were implemented: BERT with Traditional Classifiers, where embeddings were fed into Random Forest, Logistic Regression, KNN, and Naïve Bayes, and BERT with MLP and Focal Loss, designed to handle class imbalance effectively. Random Forest achieved the highest accuracy (89%) among traditional classifiers, while MLP demonstrated robust performance (87%) with better generalization. Data preprocessing included tokenization, and label encoding to optimize model inputs. Additionally, majority voting was employed to enhance traditional classifier performance. While traditional classifiers struggled with class imbalance, MLP with Focal Loss improved minority class detection. The system was integrated with a frontend interface for real-time classification, ensuring usability and efficiency.

Chapter 5

Testing

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that the software product is defect free. It involves the execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

5.1 Testing Strategies Used

In this section, the strategies that were employed to test our project will be discussed. The following list outlines the testing strategies that were utilized in our project to ensure its quality and effectiveness

5.1.1 Unit Testing

Unit testing is a critical phase of the software development life cycle where individual units or components of the software are tested in isolation to ensure their correctness and functionality. A unit refers to the smallest testable part of the software, such as a function, method, or class. Unit testing is typically performed by developers during the development process, either manually or using automated testing frameworks and tools. Unit testing helps identify bugs, errors, and defects in individual units of code in the early development stage. When a test fails, it indicates that there is an issue in the specific unit being tested, making it easier to pinpoint and fix the problem. Also, it increases the code quality. It serves as documentation for how individual units are intended to behave. Two kinds of testing, namely Black Box Testing and White Box Testing.

5.1.1.1 Black Box Testing

Black box testing treats the software as a "black box" and focuses solely on testing its functionality without considering its internal code structure or implementation details. The tester is only aware of the inputs and expected outputs of the software. The main objective of black box testing is to assess whether the software meets its specified requirements and functions correctly from the end user's perspective. It is typically conducted at the functional and system testing levels and is essential for validating the software's behavior against its functional requirements. Black box testing helps verify that the software meets its specified

requirements and functions as expected. This approach helps identify usability issues, user experience problems, and functional gaps that could affect user satisfaction.

5.1.1.2 White Box Testing

White-box testing, also known as clear-box testing or structural testing, is a testing technique where the internal structure, design, and implementation details of the software application are known and used to design and execute test cases. In white-box testing, the tester has access to the source code, architecture, and other internal details of the software being tested. It aims to achieve high code coverage, ensuring that as much of the code base as possible has been exercised by the test cases. The goal is to identify any code that has not been executed during testing, which might indicate untested or potentially buggy parts of the code. By having a deep understanding of the code, white-box testing can identify issues like logical errors, boundary value problems, and improper data handling. It can help uncover security vulnerabilities and weaknesses in the software.

5.1.2 Integration Testing

Integration testing is a software testing strategy that focuses on verifying the interactions and integration between different modules. The primary goal is to ensure that individual units or modules, which have been unit tested in isolation, work together as expected when combined into a larger system. Integration testing is crucial for identifying issues that may arise due to the integration of different components. It helps ensure that the entire system functions correctly. One of the primary reasons for integration testing is to identify issues that may arise when combining modules into a larger system. Integration issues can include data flow problems, communication failures, interface mismatches, and conflicting functionality between components. Integration testing helps catch these problems early in the development cycle before they become more difficult and costly to fix. In this, issues can be detected and fixed early.

5.2 Testing Results

We tested the software using above mentioned techniques. The results from these tests are given below

5.2.1 Results of Black Box Testing

After conducting Black Box Testing on the system, several key observations were made. The overall performance of the system was promising, achieving high accuracy in detecting

offensive language in Malayalam-English code-mixed text. The system effectively classified inputs into the predefined categories, ensuring reliable differentiation between offensive and non-offensive content.

From the results of Black Box Testing, it can be concluded that the system is in good functional shape, delivering accurate classifications and a smooth user experience.

5.2.2 Results of White Box Testing

After white box testing, the test cases were successfully satisfied. The system correctly handled text input validation and prevented the processing of empty or invalid inputs. The code efficiently managed memory and resources by discarding processed inputs. Data flow analysis confirmed that the detection algorithm followed the intended logic, accurately classifying text as offensive or non-offensive. The system's "Analyze" function displayed results correctly, ensuring proper end-to-end functionality.

5.2.3 Results of Integration Testing

After integration testing, it ensures that the "Analyze" button functionality works effectively, allowing users to submit text for offensive language detection without errors. The "All Clear" button successfully resets the input and results. The form correctly submits data to the backend, and the page refreshes smoothly while displaying accurate predictions. Integration testing verifies that all buttons function properly and that the system processes inputs correctly using the SentenceTransformer model and MLP classifier. The navbar, including the homepage and other links, works seamlessly, ensuring a user-friendly experience. Overall, the system integrates well, providing accurate results efficiently.

5.3 Summary

Software testing ensures the system meets expected requirements and is defect-free. Various testing strategies, including unit, black box, white box, and integration testing, were employed. Black box testing confirmed high accuracy in detecting offensive language and effective classification. White box testing validated proper text input handling, memory management, and algorithm logic. Integration testing ensured smooth functionality of buttons, form submission, and result display. The SentenceTransformer model and MLP classifier worked correctly, providing accurate predictions. The navbar and overall interface were user-friendly. The system demonstrated reliable performance with seamless integration and efficient processing.

Chapter 6

Results

In this section, we present the outcomes of the Offensive Language Detection System for Malayalam-English code-mixed text. The system was tested and validated with various text inputs and was found to produce the desired results.

6.1 Results

The following objectives were achieved through the course of this project:

1. Effective Detection – Enables early identification of offensive language in Malayalam-English code-mixed text by classifying it into five categories.
2. User-Friendly Integration – Provides an accessible tool to detect and classify offensive content efficiently.
3. Support for Linguistic Research – Aids researchers in understanding offensive language patterns in code-mixed texts.

6.1.1 Home Page

Figure 6.1: Home Page

Home page provides a user-friendly interface where users can enter Malayalam-English code-mixed text and check for offensive content. The interface includes a text input box, an "Analyze" button for classification, and an "All Clear" button to reset the input

6.1.2 Examples of Different Classes

Figure 6.2: Example - Not Malyalam

Figure 6.3: Example - Not Offensive

Figure 6.4: Example - Offensive Targeted Insult Individual

Figure 6.5: Example - Offensive Targeted Insult Group

Figure 6.6: Example -Offensive Untargeted

6.2 Summary

The Offensive Language Detection System for Malayalam-English code-mixed text has been successfully developed, providing an effective solution for identifying offensive content. The

system's key features have been implemented and tested, demonstrating high accuracy in classifying text into five categories. It ensures reliable detection, promoting a safer and more inclusive online environment.

Chapter 7

Conclusion

7.1 Summary

This report is structured into seven chapters, detailing the development of an Offensive Language Detection System for Malayalam-English code-mixed text. The project began with a literature review, identifying key challenges such as class imbalance and linguistic complexity in bilingual conversations.

To address these challenges, the system was designed using BERT-based embeddings with class weighting. Initially, four classifiers—Random Forest, Logistic Regression, KNN, and Naïve Bayes—were used with majority voting. However, due to class imbalance, an alternative approach using a Multilayer Perceptron (MLP) with focal loss improved accuracy as compared to conventional classifiers.

The system was integrated into a web-based frontend, where users can classify text using either the majority voting scheme or the standalone MLP model. Testing confirmed its effectiveness in detecting offensive language in Malayalam-English code-mixed text.

Future work could enhance embedding strategies, explore deep learning alternatives, and implement real-time processing to improve accuracy and adaptability. The project's outcome is a deployable classification model with practical applications for online content moderation.

7.2 Recommendations for Future Work

While the project successfully achieved its core objectives, there are several avenues for further enhancement to improve the system's accuracy and adaptability in detecting offensive language in Malayalam-English code-mixed text. We recommend the following improvements:

1. **Incorporating Topic Description:** Enhancing the model by integrating topic descriptions alongside comments can provide additional contextual understanding, improving classification accuracy for offensive language detection.

2. **Using MuRIL Instead of BERT:** Replacing the current BERT-based embeddings with MuRIL, a model specifically trained for Indian languages, may better capture the linguistic nuances of Malayalam-English code-mixed text.

3. **Dataset Augmentation:** Regularly updating and augmenting the dataset with new samples will help the model adapt to evolving language patterns, slang, and emerging offensive expressions, ensuring long-term relevance and robustness.

Implementing these enhancements will further refine the system's performance and extend its applicability for real-world offensive language detection in online conversations.

References

[1] Ziqi Zhang, David Robinson, and Jonathan Tepper. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *European semantic web conference*, pages 745–760. Springer, 2018.

[2] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). *arXiv preprint arXiv:1903.08983*, 2019.

[3] Sanjana Sharma, Saksham Agrawal, and Manish Shrivastava. Degree based classification of harmful speech using twitter data. *arXiv preprint arXiv:1806.04197*, 2018.

[4] Ji Ho Park and Pascale Fung. One-step and two-step classification for abusive language detection on twitter. *arXiv preprint arXiv:1706.01206*, 2017.

[5] Thomas Mandl, Sandip Modha, Anand Kumar M, and Bharathi Raja Chakravarthi. Overview of the hasoc track at fire 2020: Hate speech and offensive language identification in tamil, malayalam, hindi, english and german. In *Proceedings of the 12th annual meeting of the forum for information retrieval evaluation*, pages 29–32, 2020.

[6] K Sreelakshmi, B Premjith, Bharathi Raja Chakravarthi, and KP Soman. Detection of hate speech and offensive language codemix text in dravidian languages using cost-sensitive learning approach. *IEEE Access*, 12:20064–20090, 2024.

[7] Yueying Zhu and Xiaobing Zhou. Zyy1510@ hasoc-dravidian-codemix-fire2020: An ensemble model for offensive language identification. In *FIRE (Working Notes)*, pages 397–403, 2020.

[8] Sara Renjit and Sumam Mary Idicula. Cusatnlp@ hasoc-dravidian-codemix-fire2020: identifying offensive language from manglishtweets. *arXiv preprint arXiv:2010.08756*, 2020.

[9] Pankaj Singh and Pushpak Bhattacharyya. Cfilt iit bombay@ hasoc-dravidian-codemix fire 2020: Assisting ensemble of transformers with random transliteration. In *FIRE (Working Notes)*, pages 411–416, 2020.

[10] Swapnanil Mukherjee and Sujit Das. Application of transformer-based language models to detect hate speech in social media. *Journal of Computational and Cognitive Engineering*, 2(4):278–286, 2023.

[11] Pradeep Kumar Roy, Snehaan Bhawal, and Chinnaudayar Navaneethakrishnan Subalalitha. Hate speech and offensive language detection in dravidian languages using deep ensemble framework. *Computer Speech & Language*, 75:101386, 2022.

[12] Varsha Pathak, Manish Joshi, Prasad Joshi, Monica Mundada, and Tanmay Joshi. Kbcnmujal@ hasoc-dravidian-codemix-fire2020: Using machine learning for detection of hate speech and offensive code-mixed social media text. *arXiv preprint arXiv:2102.09866*, 2021.

[13] S Soumya and KV Pramod. Sentiment analysis of malayalam tweets using machine learning techniques. *ICT Express*, 6(4):300–305, 2020.

[14] Mariya Raphel, B Premjith, K Sreelakshmi, and Bharathi Raja Chakravarthi. Hate and offensive keyword extraction from codemix malayalam social media text using contextual embedding. In *Proceedings of the Third Workshop on Speech and Language Technologies for Dravidian Languages*, pages 10–18, 2023.

[15] Bharathi Raja Chakravarthi, Manoj Balaji Jagadeeshan, Vasanth Palanikumar, and Ruba Priyadharshini. Offensive language identification in dravidian languages using mpnet and cnn. *International Journal of Information Management Data Insights*, 3(1):100151, 2023.