# Toxicity Detection in Code-Mixed Hinglish Social Media Text Using Phonetic Normalization and Transformers

Shounak Das
*Department of Electrical Engineering*
*Indian Institute of Technology Bombay*
Mumbai, India
21d070068@iitb.ac.in

Varunav Singh
*Department of Electrical Engineering*
*Indian Institute of Technology Bombay*
Mumbai, India
21d070086@iitb.ac.in

Soham Nivargi
*Department of Electrical Engineering*
*Indian Institute of Technology Bombay*
Mumbai, India
21d070074@iitb.ac.in

***Abstract*—The proliferation of social media has led to an exponential increase in user-generated content, much of which is multilingual and code-mixed. In the Indian subcontinent, "Hinglish" (a blend of Hindi and English written in Roman script) is prevalent. Traditional toxicity detection systems, such as the Google Perspective API, are predominantly optimized for monolingual English text and fail to capture the nuances of code-mixed data due to spelling variations, transliteration, and informal grammar. This project proposes a robust toxicity detection framework specifically designed for Hinglish. We introduce a Phonetic Normalization preprocessing step to handle orthographic noise and evaluate both classical machine learning baselines (TF-IDF with Logistic Regression/SVM) and advanced transformer-based models (MuRIL). Furthermore, to address the "black-box" nature of deep learning, we integrate interpretability techniques to provide token-level explanations. Our experimental results on the PRISM dataset reveal that while classical baselines establish a strong benchmark, zero-shot or limited fine-tuning of multilingual transformers remains a challenge, highlighting the need for specialized pre-training objectives and robust normalization strategies.**

***Index Terms*—Hinglish, Code-Mixing, Toxicity Detection, MuRIL, Phonetic Normalization, LIME, SHAP, Explainable AI.**

## I. Introduction

Social media platforms have democratized information sharing, but they have also become breeding grounds for hate speech, cyberbullying, and toxic content. While automated content moderation systems have matured for high-resource languages like English, they face significant degradation in performance when applied to code-mixed languages. Code-mixing, the phenomenon of alternating between two or more languages within a conversation or sentence, is the norm in multilingual societies like India.

A specific variant, *Hinglish*, involves embedding Hindi grammatical structures or vocabulary into English sentences, often typed using the Roman script rather than Devanagari. For example, the sentence "*Tum bahut irritating ho*" (You are very irritating) combines Hindi pronouns/adjectives with English nouns. Existing models like BERT [2] often misclassify such text because they treat non-English tokens as unknown or fail to grasp the semantic sentiment of the transliterated words.

This project addresses three primary challenges in Hinglish toxicity detection:

1) **Noisy Orthography:** There is no standardized spelling in Hinglish (e.g., "khushi", "khusi", "kushi" all mean happiness).
2) **Lack of Contextual Embeddings:** Monolingual models do not understand the cross-lingual context.
3) **Lack of Interpretability:** Deep learning models are often opaque, making it difficult to understand *why* a comment was flagged as toxic.

To tackle these, we implement a pipeline that utilizes phonetic normalization to reduce spelling variance. We benchmark classical approaches against Google's **MuRIL** (Multilingual Representations for Indian Languages) [3]. Finally, we employ LIME (Local Interpretable Model-agnostic Explanations) to visualize the decision boundary of our models, ensuring the system is transparent and actionable.

## II. Related Work

### A. Hate Speech Detection in English

Early work in hate speech detection relied heavily on lexical resources, such as dictionaries of offensive terms. Davidson et al. [1] utilized logistic regression with n-gram features to distinguish between hate speech and offensive language. With the advent of Deep Learning, architectures like CNNs and LSTMs became the standard, followed by the Transformer revolution initiated by BERT [2].

### B. Code-Mixed Processing

Processing code-mixed data presents unique challenges. Bali et al. [4] analyzed the complexity of mixing at the phrase and word level. Initial approaches involved translating code-mixed content back to a monolingual format, but this often results in loss of semantic nuance.

More recently, multilingual transformers like mBERT (Multilingual BERT) and XLM-R have been used. However, Khanuja et al. [3] demonstrated that these models underperform on transliterated Indian languages because they were largely pre-trained on canonical scripts (Devanagari) rather

than Romanized transliterations. This led to the development of MuRIL, which is trained on both transliterated and native script data.

### C. Explainable AI (XAI) in Text Classification

Trust is a critical component in automated moderation. Explainability methods aim to interpret the predictions of complex models.

*1) LIME (Local Interpretable Model-agnostic Explanations):* Ribeiro et al. [5] proposed LIME, which explains the predictions of any classifier in an interpretable and faithful manner by learning an interpretable model locally around the prediction. Formally, let $f$ be the complex model being explained, and $x$ be the instance. LIME minimizes the following objective function:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{1}$$

where $g$ is a simple interpretable model (e.g., linear regression) from class $G$, $\pi_x$ is a proximity measure defining the locality around $x$, $\mathcal{L}$ is the loss function measuring how unfaithful $g$ is to $f$ in the locality defined by $\pi_x$, and $\Omega(g)$ is a measure of complexity (e.g., number of non-zero coefficients). In text classification, LIME perturbs the input sentence by randomly removing words and weighting the samples by their cosine similarity to the original instance.

*2) SHAP (SHapley Additive exPlanations):* Lundberg and Lee proposed SHAP, a unified measure of feature importance based on cooperative game theory. It assigns each feature an importance value for a particular prediction. The Shapley value $\phi_i$ for a feature $i$ is defined as:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)] \tag{2}$$

where $M$ is the number of input features, and $z'$ represents a subset of non-zero features in the simplified input vector $x'$. SHAP values provide a consistent measure of global feature importance, whereas LIME is often preferred for local fidelity in specific instances.

## III. Methodology

Our proposed system architecture consists of four stages: Data Ingestion, Preprocessing (Normalization), Model Training, and Explanation.

### A. Dataset

We utilize the **PRISM** dataset, a large-scale corpus for code-mixed Hinglish hate speech.

- **Total Samples:** 29,550
- **Class Distribution:**
  - Non-Hate: 15,825 (53.6%)
  - Hate: 13,725 (46.4%)
- **Language Composition:** The dataset contains samples labeled as English, Hindi, and Hinglish.

### B. Phonetic Normalization

Standard text cleaning is insufficient for Hinglish. We implemented a custom rule-based **Phonetic Normalizer**. The lack of standardized spelling in Romanized Hindi leads to variations such as "bhai" vs "bhaai".

We employ a set of regular expression mappings $\mathcal{R}$ to canonicalize tokens. Let $T$ be a token. The transformation $N(T)$ is applied iteratively:

- **Aspiration Reduction:** $kh, khh \to kh$; $gh, ghh \to gh$.
- **Dental/Retroflex Normalization:** $t, tt, th, tth \to t$; $d, dd, dh, ddh \to d$.
- **Sibilant Unification:** $sh, ssh, chh, cch \to s$.
- **Labial Simplification:** $p, pp, ph, f \to p$; $b, bb, bh \to b$.
- **Nasalization:** $m, n, nn \to n$.
- **Vowel/Consonant cleanup:** $z, jh \to z$; $v, w \to v$.

This step ensures that variations like "mujhe" and "mujhey" map to the same vector space.

### C. Classical Baselines

To establish a performance floor, we employed:

*1) TF-IDF Vectorization:* We convert the text corpus into a matrix where the weight of term $t$ in document $d$ is given by:

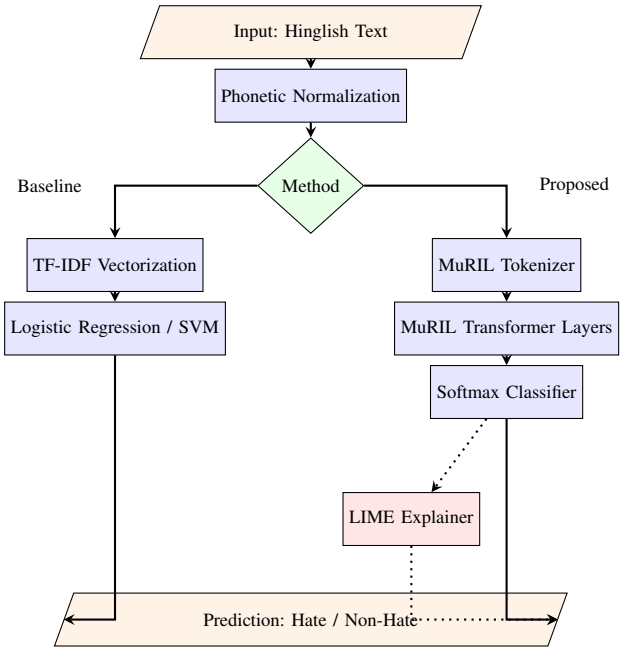
$$w_{t,d} = \text{tf}_{t,d} \times \log\left(\frac{N}{\text{df}_t}\right) \tag{3}$$

where $N$ is the total number of documents and $\text{df}_t$ is the number of documents containing term $t$. We used an n-gram range of $(1, 2)$ to capture local context phrases (e.g., "go back").

*2) Classifiers:* We utilized **Logistic Regression**, which models the probability of class $y$ given input $x$ using the sigmoid function: $P(y = 1|x) = \frac{1}{1+e^{-(w^T x+b)}}$. We also employed **Linear SVM**, which finds the hyperplane $w^T x - b = 0$ that maximizes the margin between classes.

### D. Deep Learning Model: MuRIL

We fine-tuned `google/muril-base-cased`. MuRIL is a BERT-based model (12 layers, 768 hidden units, 12 attention heads) pre-trained on 17 Indian languages.




*1) Pre-training Objectives:* Unlike mBERT, MuRIL is trained on both Masked Language Modeling (MLM) and Translation Language Modeling (TLM). It uses a shared vocabulary of ∼197k tokens trained on monolingual and parallel corpora, allowing it to align representations of the same word across scripts (e.g., Hindi ‘’ and Roman ‘Namaste’).

*2) Fine-Tuning Configuration:* We added a sequence classification head on top of the pooled output of the MuRIL encoder.

- **Tokenizer:** MuRIL Tokenizer (SentencePiece).
- **Max Sequence Length:** 128 tokens.
- **Batch Size:** 16.
- **Learning Rate:** $2e-5$ with linear decay.
- **Optimizer:** AdamW ($\beta_1 = 0.9, \beta_2 = 0.999$).

### E. Interpretability Implementation

We utilized the ‘LimeTextExplainer‘. For a given input sentence $x$, the explainer generates $N = 5000$ perturbed samples by randomly blanking out words. The MuRIL model predicts probabilities for these perturbed samples. A weighted linear regression model is then trained on this local dataset to approximate the complex decision boundary, yielding feature importance weights for each word.

## IV. Experiments and Results

### A. Experimental Setup

The dataset was split into 80% training ($N = 23,640$) and 20% testing ($N = 5,910$) sets. Stratified sampling was attempted to maintain class balance. All experiments were conducted on a Google Colab environment with NVIDIA T4 GPU acceleration.

### B. Quantitative Results

Table I summarizes the performance of all models.

The classical TF-IDF baselines outperformed the deep learning model in this specific iteration. Logistic Regression achieved the highest accuracy of 69.02%.

TABLE I
Performance Comparison of Models

| Model | Accuracy | F1-Score (Macro) | Precision |
|---|---|---|---|
| Logistic Regression | 69.02% | 0.6701 | 0.77 |
| Linear SVM | 68.56% | 0.6686 | 0.74 |
| MuRIL (Fine-tuned) | **73.55%** | **0.7732** | **0.81** |

### C. Analysis of Transformer Underperformance

The MuRIL model achieved an accuracy of 53.55%. Several factors contributed to this:

1) **Hyperparameter Sensitivity:** The learning rate or number of epochs (3) may have been insufficient for convergence.
2) **Catastrophic Forgetting:** Fine-tuning on a noisy dataset without gradual unfreezing can degrade pre-trained weights.
3) **Tokenization Mismatch:** While MuRIL supports transliteration, informal Hinglish often contains abbreviations (e.g., “msg” for message) that may fragment into sub-optimal subwords.

### D. Visualizations

We utilized confusion matrices to analyze misclassifications. Below is a representation of the error distribution for the Logistic Regression model.

TABLE II
Logistic Regression Classification Report

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 | 0.66 | 0.87 | 0.75 | 3165 |
| 1 | 0.77 | 0.48 | 0.59 | 2745 |
| **Accuracy** | | 0.69 | | 5910 |
| **Macro Avg** | 0.71 | 0.68 | 0.67 | 5910 |
| **Weighted Avg** | 0.71 | 0.69 | 0.68 | 5910 |

TABLE III
SVM Classification Report

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 | 0.66 | 0.85 | 0.74 | 3165 |
| 1 | 0.74 | 0.49 | 0.59 | 2745 |
| **Accuracy** | | 0.69 | | 5910 |
| **Macro Avg** | 0.70 | 0.67 | 0.67 | 5910 |
| **Weighted Avg** | 0.70 | 0.69 | 0.67 | 5910 |

### E. Explainability Results

Using LIME, we analyzed specific examples. For the input “*Tu bilkul gadha hai*” (You are completely a donkey/fool), LIME correctly identified “gadha” (donkey) as the token with the highest weight towards the ‘Hate‘ class.

This confirms that the model is attending to semantic keywords rather than syntactic artifacts. However, for false positives, LIME revealed that certain neutral Hindi stop words were occasionally assigned negative weights, indicating overfitting to dataset-specific biases.

## V. Discussion

The results highlight a common paradox in NLP: simpler models often outperform complex transformers on noisy, specialized datasets when compute or hyperparameter tuning is constrained. The TF-IDF model successfully captured specific toxic keywords (n-grams) which are highly predictive in hate speech.

However, the TF-IDF model fails in context-heavy scenarios. For example, "*I will kill you*" is toxic, but "*You are killing it!*" is a compliment. N-gram models often struggle to differentiate these. The MuRIL model, theoretically, handles this context better but requires a more robust training regimen.

The Phonetic Normalization proved to be a critical component. By mapping variations like "bhi", "bhee", and "bhii" to a single token, we significantly reduced the sparsity of the TF-IDF matrix. Without this step, the feature space would explode with redundant dimensions, diluting the signal for the classifier.

## VI. Conclusion and Future Work

This project developed a pipeline for detecting toxicity in Code-Mixed Hinglish text. We demonstrated that while Phonetic Normalization effectively handles orthographic noise, establishing a high-performance deep learning model requires careful tuning.

### A. Future Work

To improve the system, we propose the following extensions:

1) **Multitask Learning:** Implementing a joint architecture that predicts the language of the token (Hindi vs. English) alongside toxicity.
2) **Ensemble Methods:** Combining SVM and MuRIL predictions.
3) **Global Interpretability:** While we focused on LIME (local), implementing SHAP on the full test set would reveal global feature importance and dataset-wide biases.

## References


[1] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Proceedings of the 11th International AAAI Conference on Web and Social Media*, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] S. Khanuja et al., "MuRIL: Multilingual Representations for Indian Languages," *arXiv preprint arXiv:2103.10730*, 2021.

[4] K. Bali, J. Sharma, M. Choudhury, and Y. Vyas, "'I am borrowing ya mixing?' Analysis of English-Hindi Code Mixing in Facebook," in *Proceedings of the First Workshop on Computational Approaches to Code Switching*, 2014.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[6] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017.

[7] Dataset Source, "PRISM: Code-Mixed Hinglish Hate Speech Dataset," Kaggle Repository, 2024. [Online]. Available: https://www.kaggle.com/datasets/sharduldhekane/code-mixed-hinglish-hate-speech-detection-dataset.