Detecting Implicit Hate Speech on Social Media through Neural Models

Qiyue Gu, Binghui Ni

1. Introduction

Social media platforms have become central spaces for public discussion. They host a growing volume of user generated content, and with this growth comes an increasing amount of harmful language. Many moderation systems focus on identifying explicit hate speech. Explicit hate is often direct and easy to flag because it contains recognizable insults, slurs, or direct attacks toward a target group. However, recent studies show that harmful expressions are becoming more subtle. Users often avoid direct abusive terms and instead rely on implications, stereotypes, coded language, or indirect phrasing to express hostility. This shift toward implicit hate makes automatic detection difficult for traditional classifiers that rely mainly on surface patterns or explicit keywords.

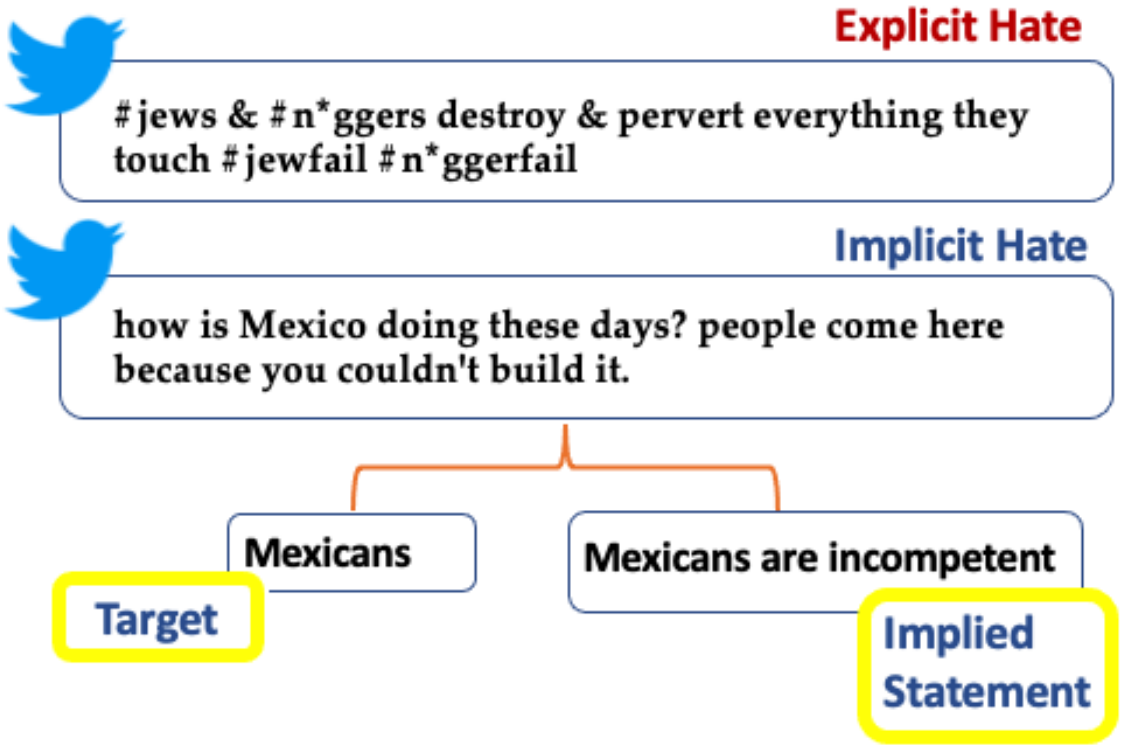


Figure 1: Source: *Latent Hatred: A Benchmark for Understanding Implicit Hate Speech*[1]

To study this challenge, our work focuses on distinguishing implicit hate from non hate content. We use the *Latent Hatred dataset* created by ElSherief et al in 2021[1]. This dataset was released at EMNLP 2021 and is one of the first benchmarks designed specifically for understanding implicit hate. It contains three main categories: *not hate*, *implicit hate*, and *explicit hate*. The *not hate* class has 13,270 samples, the *implicit hate* class has 7,094 samples, and the *explicit hate* has 1,086 samples. Since the *explicit hate* category is relatively small and existing methods can already identify it effectively, this project focuses only on the *not hate* and *implicit hate* data. The dataset includes many examples where hate is conveyed through implication rather than direct insults. This allows us to examine how well different models handle subtle and context dependent expressions.

The goal of this project is to evaluate how both traditional and modern deep learning models perform on this challenging task. We begin with Naive Bayes as our baseline model, which relies on simple bag-of-words features. We then move to two neural models with increasing representational power: a bidirectional LSTM, which captures sequential patterns in text, and BERT, a transformer-based contextual model that learns deep semantic relationships across the entire sentence. Comparing these three models allows us to understand how much additional

linguistic capability is needed to effectively detect implicit hate.

Another important aspect of our study is the strong class imbalance in the dataset. Implicit hate is less frequent than non hate. Models trained on the raw data tend to favor the majority class and often fail to identify implicit hate. To address this issue, we use class weighting and downsampling. We test these methods with both LSTM and BERT to understand how each model responds to imbalance correction.

Overall, this project aims to answer three questions:

- How much improvement can be obtained through simple methods for handling class imbalance?
- Do transformer-based models like BERT offer an advantage over sequence models like LSTM?
- Can neural models detect implicit hate better than traditional baselines?

The results help highlight both the progress in hate speech detection and the remaining challenges in identifying subtle implicit forms of abuse.

2. Literature Review

Research on abusive and hateful language has grown rapidly in recent years, especially as online communication becomes more common. Much of the early work focused on explicit hate speech, where harmful intent can be detected through clear and direct wording. More recent studies highlight the importance of implicit hate, which is often indirect and harder to detect with traditional systems. This section reviews five papers that provide the foundation for our study. They cover implicit hate benchmarks, abuse classification frameworks, deep learning methods, transformer models, and class imbalance techniques. Each paper contributes insight that shapes the design and analysis of our experiments.

2.1 Implicit Hate Benchmarking

ElSherief et al (2021)[1] introduce the Latent Hatred dataset, which is the dataset used in our project. The authors argue that implicit hate is under studied and much harder than explicit hate because it relies on indirect cues, social knowledge, and implied group targeting. They create a benchmark through expert annotation and define several categories of implicit hate, such as stereotypes and insinuations. The main contribution is the construction of a high quality dataset focused specifically on implicit hate rather than general abusive content.

In our study, this paper is central because it provides the dataset for our classification task. It also guides our motivation. The authors show that implicit hate cannot be solved by relying on keywords or simple patterns. This insight supports our use of deep learning models that capture context, such as LSTM and BERT.

2.2 Implicit and Explicit Offensive Language

Caselli et al (2020)[2] examine the difference between implicit and explicit offensive language. They point out that offensive meaning often depends on context rather than direct word choice. Their main contribution is a dataset that separates explicit, implicit, and non offensive statements, along with an analysis of the linguistic features that characterize each category. The authors also note that many models trained on explicit content fail to capture implicit messages.

This work supports our classification setting. It reinforces the idea that implicit abuse requires models capable of understanding subtle cues rather than relying only on surface tokens.

2.3 Early Deep Learning Approaches

Badjatiya et al (2017)[3] apply deep neural networks, including LSTM and gradient boosted embeddings, to hate speech detection. The authors show that deep learning methods significantly outperform traditional bag of words and SVM baselines. They also demonstrate that word embeddings help capture contextual similarity that simple frequency based models miss.

This paper informs our decision to include an LSTM model as a representative deep sequence model. It highlights the benefit of learning representations rather than using manually selected features. The results in this paper also provide a comparison point for our LSTM performance on implicit hate.

2.4 Transformer Models for Hate Detection

Mozafari et al (2019)[4] show how BERT can improve hate speech detection by learning contextualized word representations. The authors fine tune BERT on several hate speech datasets and report strong improvements over LSTM, CNN, and traditional machine learning models. They argue that transformer based models are able to capture nuanced meaning and long range dependencies that earlier models struggle with.

This has direct relevance to our work. The paper suggests that BERT should perform better than LSTM. In our results, BERT does outperform LSTM, but not by a large margin. This gap between expected and observed performance provides an important point of discussion later in the report.

2.5 Class Imbalance in Text Classification

Japkowicz (2000)[5] examines how class imbalance affects classifier behavior. The author compares several balancing methods, including oversampling, undersampling, and instance weighting. The key finding is that many classifiers default to the majority class when data distribution

is skewed. Simple balancing approaches often lead to clear performance gains, especially for recall of the minority class.

This idea directly informs our handling of the dataset. The implicit hate class is smaller, so we test both class weighting and downsampling. The paper helps explain why these techniques produce meaningful improvements in both the LSTM and BERT models and why the original unbalanced training yields biased predictions.

3. Naive Bayes

3.1 Model Architecture

The Naive Bayes model serves as the baseline for this study. It uses a bag of words representation of each post, constructed through the TF IDF transformation. In this representation, each post is converted into a weighted vector that reflects the importance of every unigram and bigram in the text. Words that occur frequently within a post but not across the entire dataset receive higher weight, while common words are downweighted. This representation allows the model to emphasize distinctive lexical patterns.

The classifier is based on the multinomial form of Naive Bayes. It models the probability that a post belongs to a class by assuming that the tokens in the post are generated independently once the class is known. For each class, the model estimates how likely each word is to appear in that class. During prediction, the model combines these likelihoods with the prior probability of the class and assigns the post to the class with the larger resulting probability.

Although the independence assumption is not realistic for natural language, the model remains effective when classification depends on explicit keywords or direct patterns of word usage. It tends to work well for explicit hate, since explicit hate relies on clear surface level cues such as slurs or direct insults. However, this same assumption limits the model's ability to recognize implicit hate. Implicit hate often depends on context, implication, or indirect phrasing, and these patterns cannot be captured through isolated word counts.

Thus, the Naive Bayes architecture provides a simple and interpretable baseline. It highlights what can be learned from lexical signals alone and establishes a reference point for evaluating the benefits of deeper models that can capture semantics beyond word frequency.

3.2 Workflow

The baseline model follows a straightforward workflow that converts each post into lexical features and trains a probabilistic classifier to distinguish implicit hate from non hate content.

1. Data Processing

The text undergoes light cleaning before feature extraction. This includes removing repeated spaces, stripping HTML fragments, and removing the "RT" marker from retweets. The goal is to reduce simple noise while keeping the original surface wording, since the baseline model relies directly on lexical cues.

2. Feature Extraction

Each post is transformed into a weighted representation using TF-IDF. The feature space includes unigrams and bigrams. Words and short phrases that appear frequently within a post but less often across the dataset receive higher weight and are therefore more informative. This representation captures local lexical patterns without using word order beyond two word sequences.

3. Model Fitting

A multinomial Naive Bayes classifier is trained on the TF IDF vectors. For each class, the model estimates how likely each token pattern is to appear. These likelihoods, along with the class prior, form the basis of the prediction rule. This approach is efficient and interpretable, though limited in its ability to capture deeper semantic context.

4. Evaluation

After training, the model predicts labels on the test set. We compute precision, recall, F1 score, and the confusion matrix. These results form the baseline for comparing the performance of the LSTM and BERT models.

3.3 Experiments and Results

We evaluate the Naive Bayes baseline under three settings. The first uses the original imbalanced dataset. The second applies class weights based on inverse class frequency. The third trains on a downsampled dataset where the two classes are equally represented. These results show how a simple lexical model behaves on a task that requires subtle semantic understanding.

3.3.1 Original Dataset

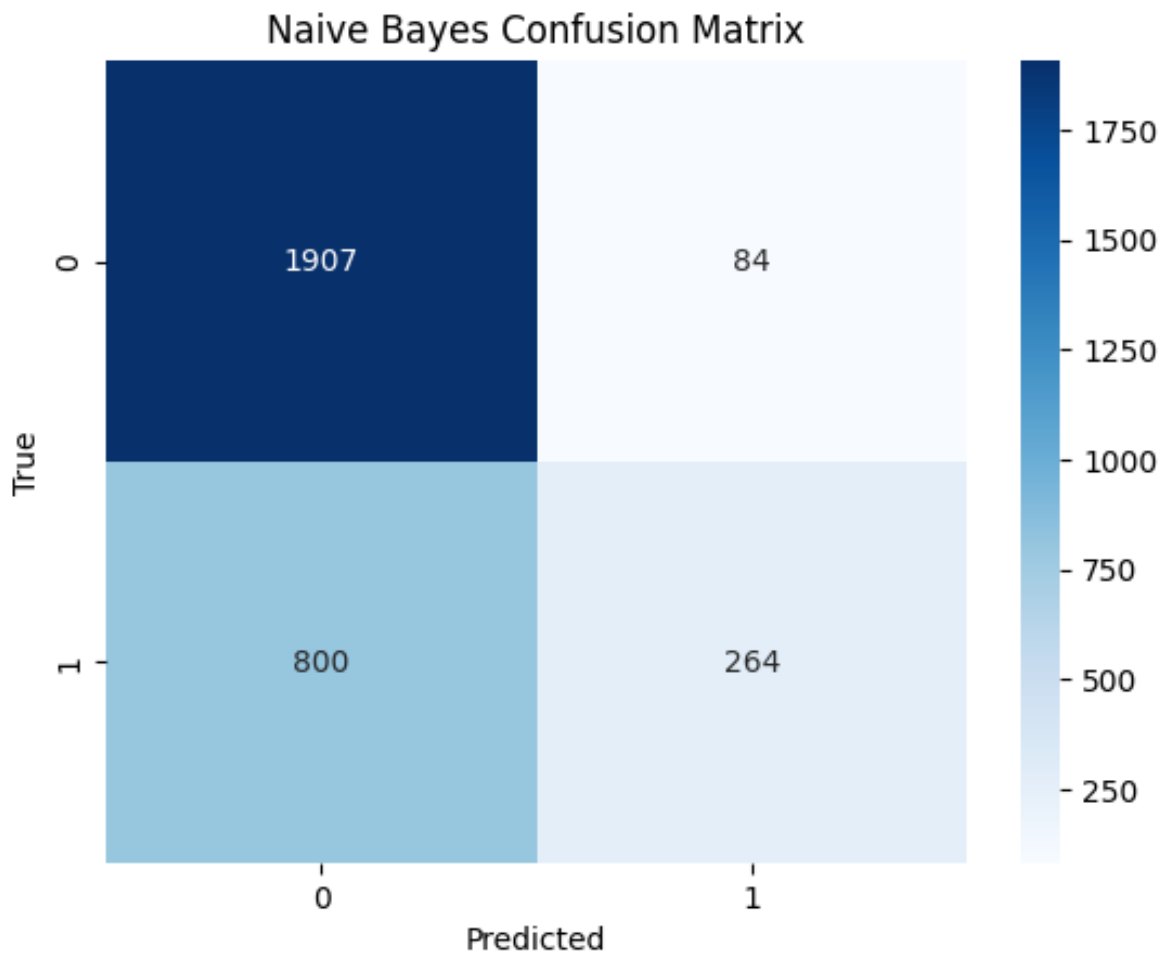
When trained on the original dataset, the model strongly favors the majority class.

```
Naive Bayes (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.7045    0.9578    0.8118      1991
           1     0.7586    0.2481    0.3739      1064

    accuracy                         0.7106      3055
   macro avg     0.7315    0.6030    0.5929      3055
weighted avg     0.7233    0.7106    0.6593      3055
```

It reaches high recall for non hate posts but performs poorly on the implicit hate class. The recall for implicit hate is low, meaning that many subtle or indirect examples are classified as non hate.



This behavior is expected because Naive Bayes depends heavily on distinctive lexical cues. Implicit hate often does not contain explicit markers, so the model struggles to identify it. The results highlight the difficulty of the task and provide a lower bound for comparison with deeper models.

3.3.2 With Class Weights

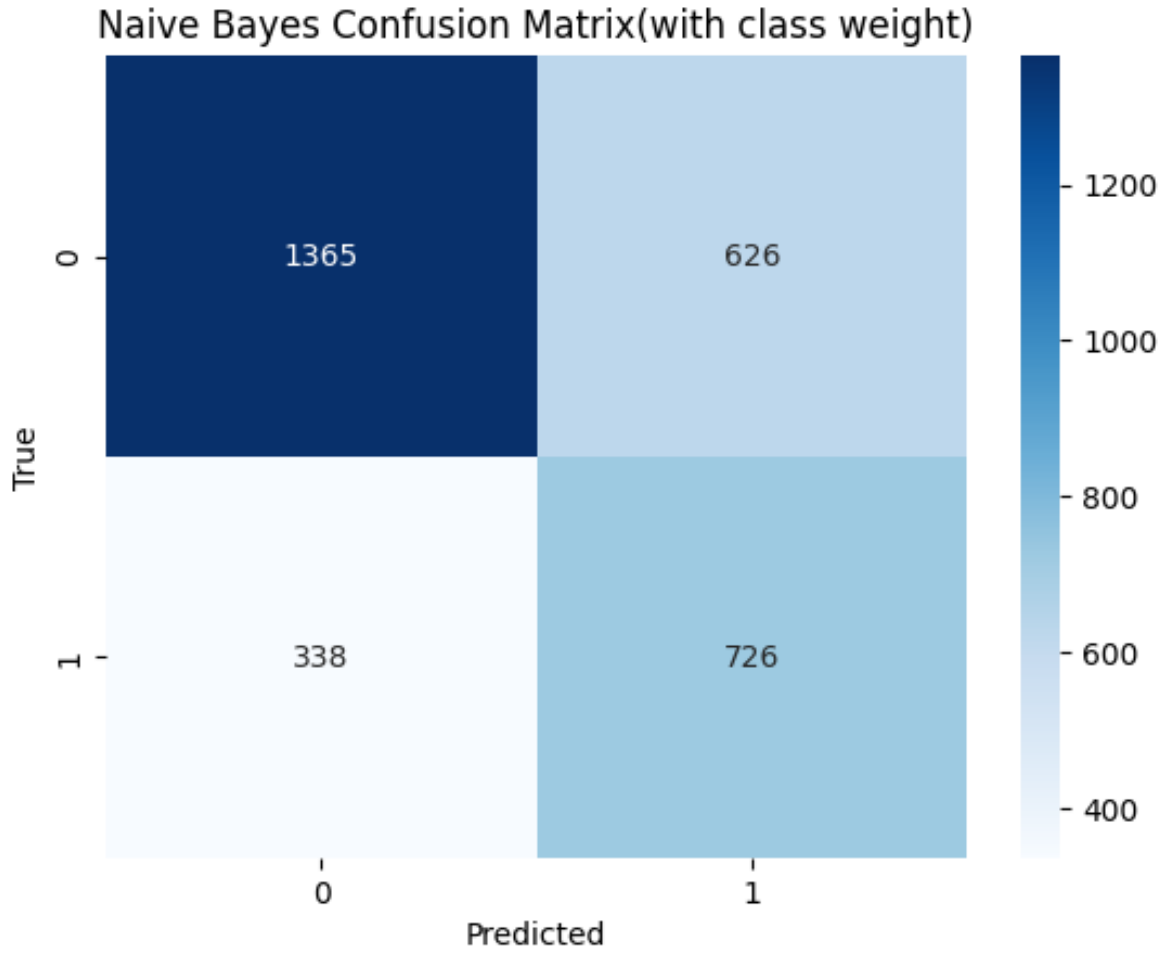
Applying class weights encourages the model to pay more attention to the minority class.

```
Naive Bayes (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.8015    0.6856    0.7390      1991
           1     0.5370    0.6823    0.6010      1064
```

```
    accuracy                           0.6845      3055
   macro avg     0.6693    0.6840    0.6700      3055
weighted avg     0.7094    0.6845    0.6910      3055
```

Recall for implicit hate increases noticeably, and the predictions become more balanced. However, this comes with a reduction in precision for the minority class because the model predicts hate more often.



The shift in behavior shows that weighting can partially correct the bias introduced by the skewed training distribution. Even so, the overall performance remains limited because the model still relies only on lexical frequency and cannot capture deeper semantic patterns.

3.3.3 Downsampling the Majority Class

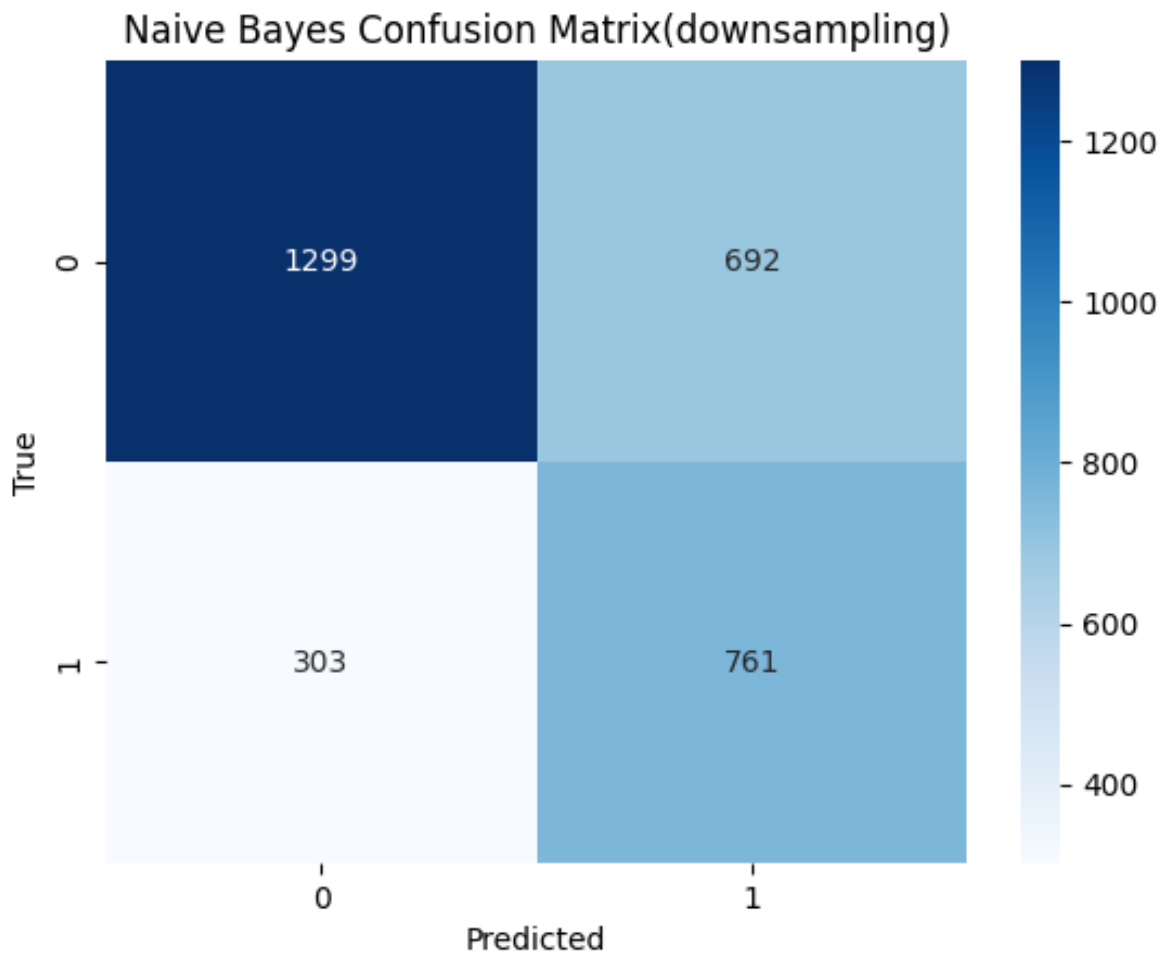
Downsampling creates a balanced training set by reducing the number of non hate samples.

```
Naive Bayes (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.8109    0.6524    0.7231      1991
           1     0.5237    0.7152    0.6047      1064

    accuracy                         0.6743      3055
   macro avg     0.6673    0.6838    0.6639      3055
weighted avg     0.7109    0.6743    0.6818      3055
```

With this adjustment, the classifier produces results similar to the class weighted setting. Recall for implicit hate increases, and the model becomes less biased toward the majority class. The accuracy decreases slightly, which reflects the loss of training information from the removed majority samples.



Despite this, downsampling improves the model's ability to detect implicit hate. The similarity between the two methods shows that both weighting and downsampling modify the effective class ratio in a way that benefits the baseline.

3.4 Summary

The Naive Bayes baseline shows that simple lexical features are not sufficient for detecting implicit hate. Although the model performs well on direct and explicit patterns, it struggles with subtle or indirect expressions and remains strongly influenced by class imbalance. These limitations demonstrate the need for models that can capture deeper semantic and contextual information, which motivates the use of LSTM and BERT in the following sections.

4. LSTM

4.1 Model Architecture

Recurrent neural networks process text as a sequence and update a hidden state as they read each word. LSTM units improve on standard RNNs by using gating mechanisms. These gates control how much of the previous information is kept, how much new information is added, and how much is passed to the next time step. This helps the model keep important information across longer spans of text and reduces the vanishing gradient problem.

At each time step t, the model receives an input vector x_t and the previous hidden state h_{t-1} and cell state c_{t-1}. The LSTM first computes three gates and a candidate cell state. The formulas are

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c} * t = \tanh(W_c x_t + U_c h * t - 1 + b_c)$$

Here f_t is the forget gate, i_t is the input gate, o_t is the output gate, and $\tilde{c}_t$ is the candidate update. The function σ is the sigmoid function and tanh is the hyperbolic tangent. The model then updates the cell state and the hidden state as

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \tanh(c_t)$$

The symbol $\odot$ denotes elementwise multiplication. In words, the forget gate decides how much of the old memory c_{t-1} to keep. The input gate decides how much of the new candidate $\tilde{c}_t$ to write into the cell. The output gate decides how much of the internal cell state to expose as the hidden state h_t. This structure allows the LSTM to keep or discard information in a controlled way.

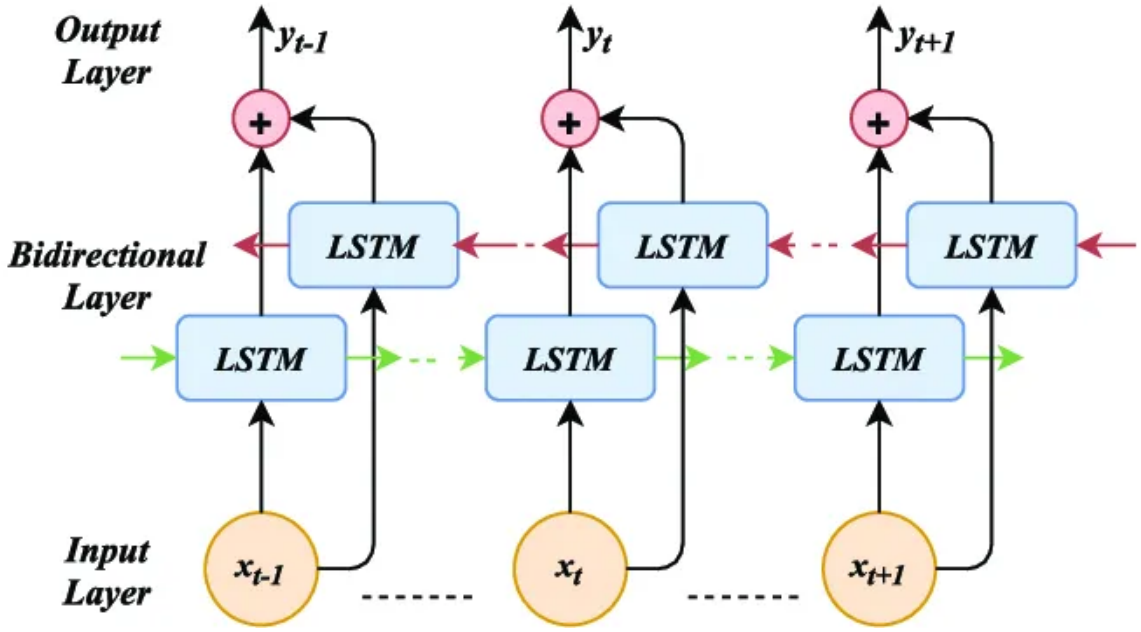


Figure 2: Source: *A Deep Learning Approach for Human Activities Recognition From Multimodal Sensing Devices*[6]

A bidirectional LSTM reads the sentence from both directions. One LSTM processes the input from left to right and produces a sequence of forward hidden states $(\overrightarrow{h_1}, \ldots, \overrightarrow{h_T})$. Another LSTM processes the same input from right to left and produces backward hidden states $(\overleftarrow{h_1}, \ldots, \overleftarrow{h_T})$. For each position t, the final representation is formed by combining these two states, for example by concatenation

$$h_t^{\text{bi}} = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

In our model, we use the final bidirectional hidden state as a summary of the whole sentence and pass it to a dense layer for binary classification. While this architecture captures sequential patterns and some context, it still depends entirely on the supervised training data and does not use external knowledge or very long range reasoning. This makes it a strong but limited baseline for implicit hate detection.

4.2 Workflow

The LSTM model follows a clear workflow that includes data cleaning, tokenization, sequence preparation, model construction, and training. The main steps are described below.

1. Data Processing

We begin by cleaning the raw text. This includes removing repeated spaces, HTML tags, and Twitter style markers such as the *rt* symbol for retweets. We also convert the text to lowercase and remove characters that do not carry meaning, such as stray punctuation or URL fragments. These steps ensure that the model focuses on the actual content of the message instead of formatting noise.

2. Tokenize

After cleaning, each post is converted into a sequence of tokens. We use a word level tokenizer with a maximum vocabulary size of 20,000. Words outside this vocabulary are mapped to a special unknown token. Tokenization converts each sentence into a list of integer ids that the embedding layer can process.

3. Sequence Padding

Posts have different lengths, so we pad or truncate each token list to a fixed length of 50. Padding uses zeros at the beginning of the sequence. This allows us to create batches of uniform size and ensures that the LSTM receives consistent input shapes.

4. Embedding Layer

Each token id is mapped to a vector of dimension 100. These embeddings are learned during training. The embedding layer helps the model capture similarity between words and smooth the input space.

5. Bidirectional LSTM Layer

The padded embedding sequences are passed into a bidirectional LSTM with 64 units in each direction. The forward pass processes the sequence from left to right and the backward pass processes it from right to left. The outputs from both directions are combined to form a single representation of the sentence.

6. Dropout and Classification Layer

A dropout layer with rate 0.5 is applied to reduce overfitting. The final hidden state from the bidirectional LSTM is then fed into a dense layer with a sigmoid activation to produce a probability that the post contains implicit hate.

7. Training

We train the model using the Adam optimizer and binary cross entropy loss. The training loop includes early stopping based on validation loss. This prevents the model from overfitting because the validation curves diverge early in the training process. We also monitor accuracy, precision, recall, and AUC to evaluate the model.

8. Evaluation

After training stops, the version of the model with the lowest validation loss is used for testing. The model processes each post and produces a probability for the implicit hate class. These predictions are then compared with the ground truth labels to compute accuracy, precision, recall, F1 score, and the confusion matrix.

4.3 Experiments and Results

We evaluate the LSTM model under three settings. The first uses the raw training data. The second uses class weights computed from the training distribution. The third uses a downsampled dataset in which the majority class is reduced to match the minority class. Although each training run is set to ten epochs, early stopping is triggered around the third epoch in all cases. This shows that the model begins to overfit very early. The training loss continues to fall while the validation loss increases.

4.3.1 Original Dataset

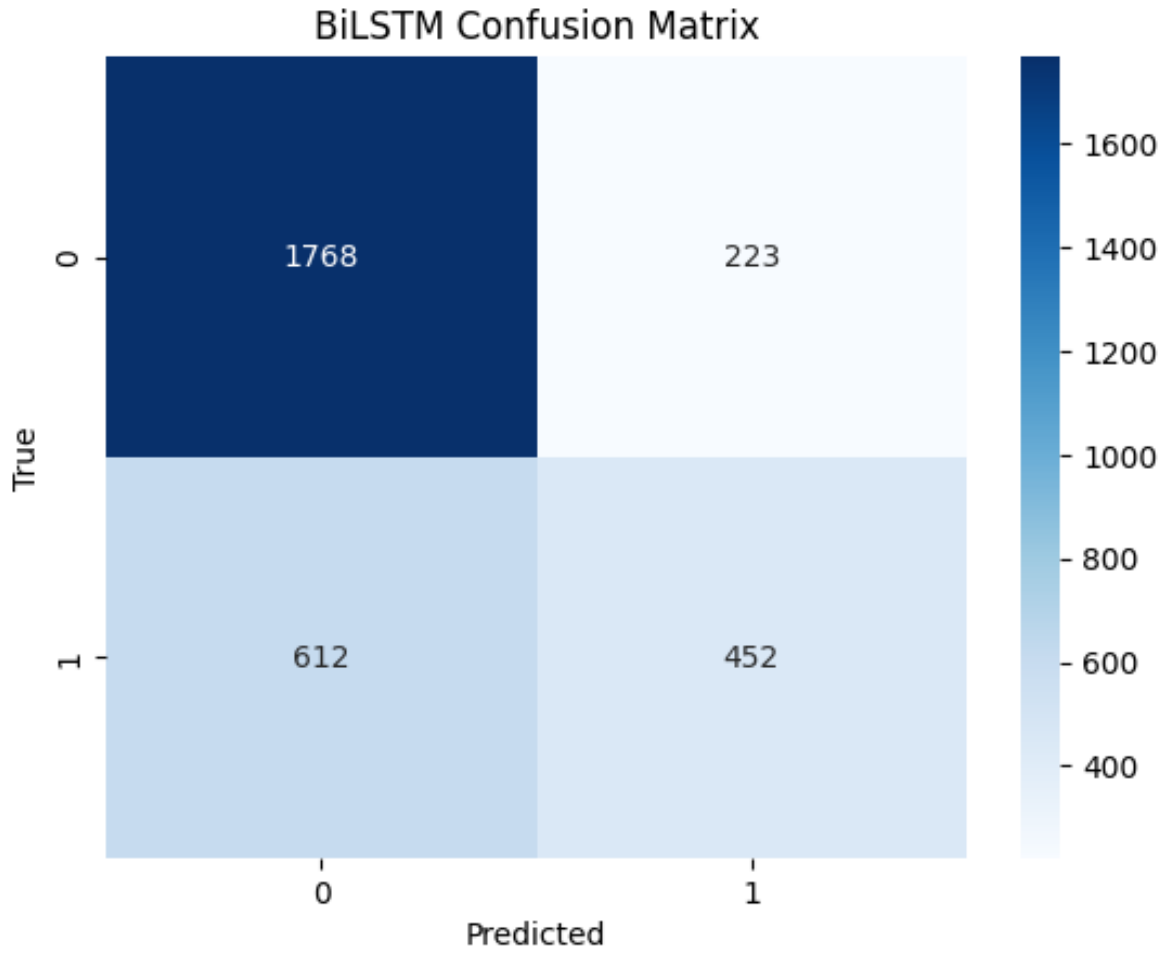
When trained on the original imbalanced dataset, the model learns to favor the majority class.

```
BiLSTM (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.7429    0.8880    0.8090      1991
           1     0.6696    0.4248    0.5198      1064

    accuracy                         0.7267      3055
   macro avg     0.7062    0.6564    0.6644      3055
weighted avg     0.7174    0.7267    0.7083      3055
```

It achieves reasonable accuracy but performs poorly on the implicit hate class. The recall for implicit hate is low, which means the model misses many harmful posts. This confirms that class imbalance is a major difficulty in the task.



The model sees far more non hate samples and therefore is biased toward predicting the safer majority class.

4.3.2 With Class Weights

Applying class weights changes the loss function so that errors on minority class examples contribute more to the gradient. Specifically, the weighted binary cross-entropy loss is defined as:

$$\mathcal{L} * \text{BCE} = -\frac{1}{N} \sum *i = 1^{N} \left[w_1, y_i \log(\hat{y}_i) + w_0, (1 - y_i) \log(1 - \hat{y}_i) \right],$$

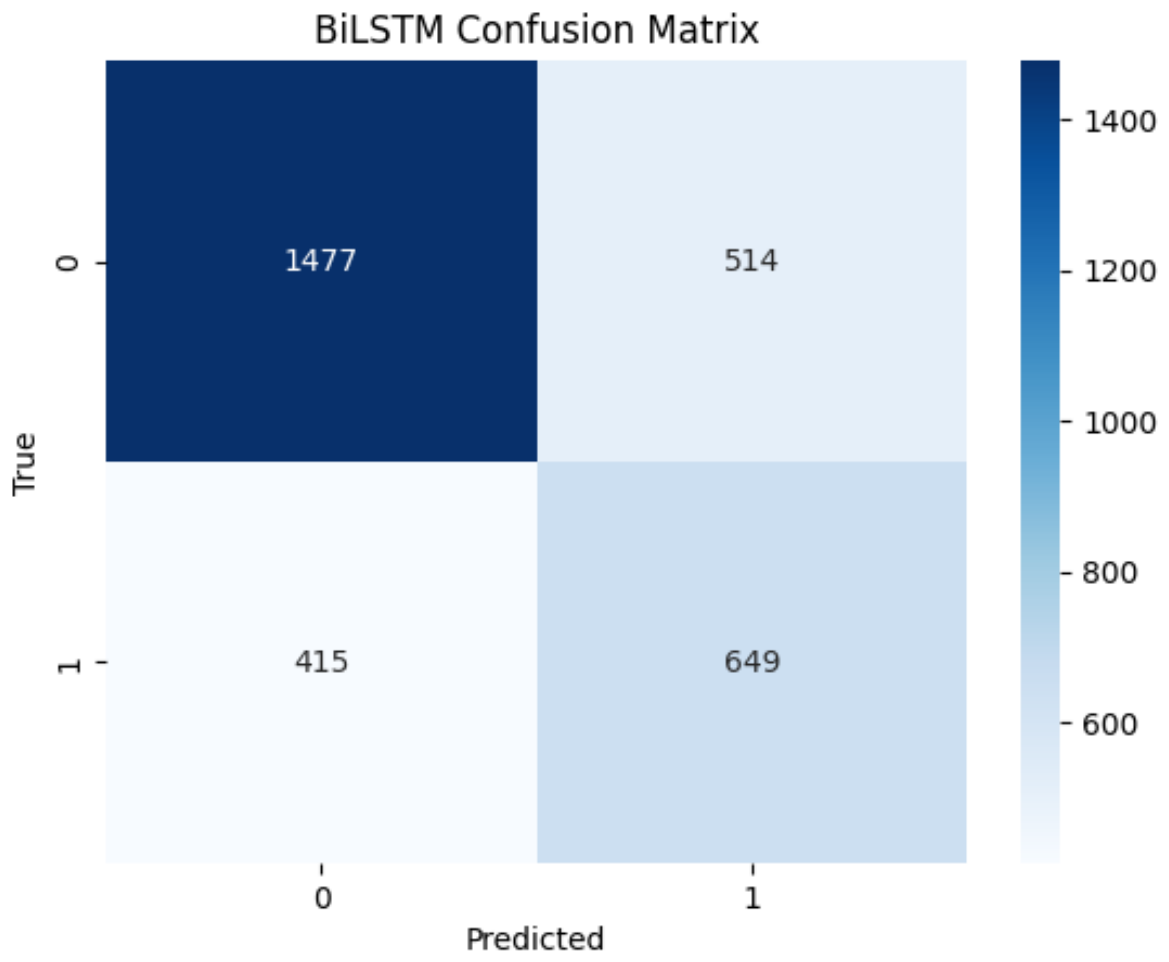
where w_1 increases the penalty for misclassifying implicit hate and w_0 controls the contribution of the majority class. This adjustment helps correct the imbalance without reducing the amount of training data.

```
BiLSTM (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.7807    0.7418    0.7608      1991
           1     0.5580    0.6100    0.5828      1064

    accuracy                         0.6959      3055
   macro avg     0.6693    0.6759    0.6718      3055
weighted avg     0.7031    0.6959    0.6988      3055
```

With this adjustment, recall for the implicit class increases in a clear way. The model becomes more sensitive to minority patterns and detects more subtle cases than before. Precision for implicit hate decreases slightly because the model predicts hate more often, but the overall F1 score becomes more balanced.



4.3.3 Downsampling the Majority Class

Downsampling creates a balanced dataset by removing a portion of the majority class (not-hate). It results in an equal number of not-hate and implicit hate examples. In this setting, the model sees a symmetric distribution during training.

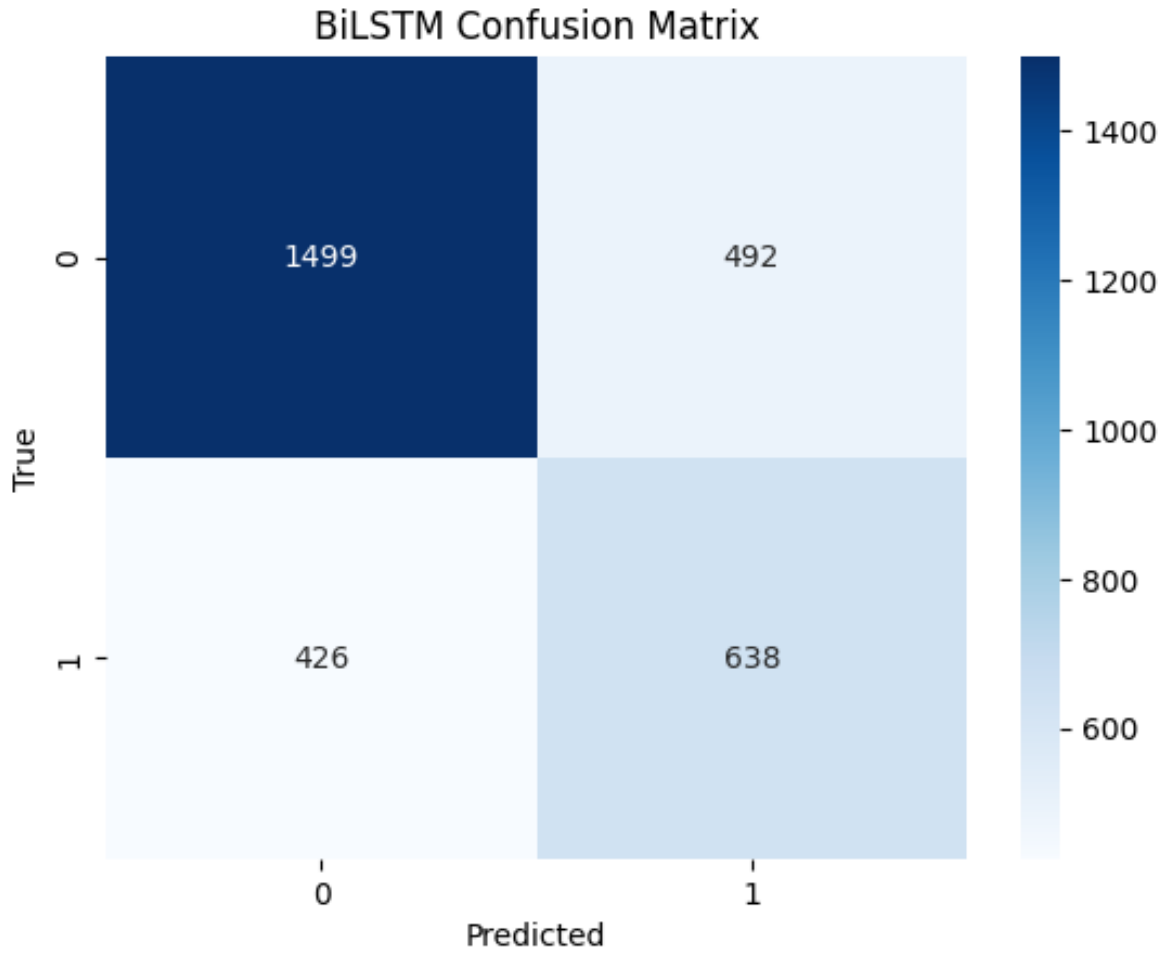
```
BiLSTM (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.7787    0.7529    0.7656      1991
           1     0.5646    0.5996    0.5816      1064

    accuracy                         0.6995      3055
```

macro avg	0.6717	0.6763	0.6736	3055
weighted avg	0.7041	0.6995	0.7015	3055

The performance pattern is very similar to the class weight setting. Recall for implicit hate improves, and the model is less biased toward the majority class. Accuracy decreases slightly because removing many non hate samples reduces the variety of examples the model sees.



Even so, recall remains strong and the model produces a more balanced prediction pattern. Since both methods shift the effective class ratio, their behavior is almost identical.

4.4 Summary

The most important observation is that both class weighting and downsampling produce similar improvements. Both methods change the relative importance of the implicit hate class, and

both lead the model to consider minority examples more carefully. In class weighting, this is achieved by increasing the loss contribution of these samples. In downsampling, this is achieved by reducing the number of majority class updates. In practice, both techniques increase the share of gradient updates that come from implicit hate examples. For a model like LSTM, this leads to similar optimization behavior and similar performance outcomes.

The LSTM results show that implicit hate detection cannot rely on raw data alone. Without addressing class imbalance, the model fails to identify many harmful posts. Simple balancing methods give clear gains and help shift the model's attention to important minority samples. However, the improvements are limited by the model's sequential architecture and by the complexity of implicit hate. These findings provide a starting point for evaluating more advanced contextual models such as BERT, which we examine in the next section.

5. Bert

5.1 Model Architecture

The BERT model used in this project is the base version of the transformer architecture, with 12 encoder layers, 12 attention heads in each layer, and a hidden size of 768. The model is designed to capture contextual meaning by allowing every word in a sentence to attend to every other word. This makes it more suitable for detecting implicit hate, which often depends on subtle relations between words rather than explicit keywords.

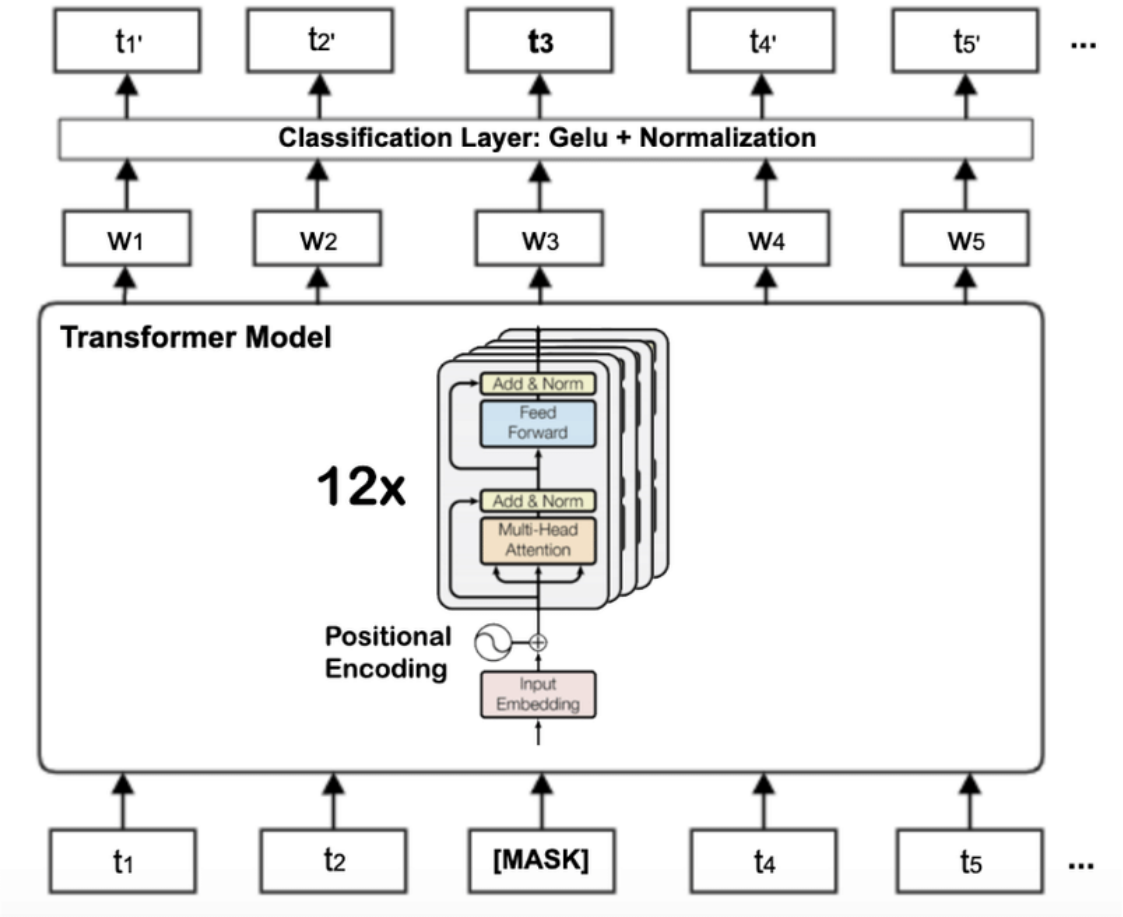


Figure 3: Source: *Rubert: A bilingual roman urdu bert using cross lingual transfer learning*[7]

At its core, BERT uses multi head self attention. For an input sequence that has been tokenized and embedded, attention is computed through three learned projections. Each token produces a query vector Q, a key vector K, and a value vector V. The attention score between two tokens is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where d_k is the dimension of the key vectors. The softmax normalizes the similarity scores so that each token distributes its attention across the entire sequence in a stable way.

BERT stacks multiple layers of these attention operations. Each layer uses residual connections and layer normalization to improve stability during training. After the attention block, a feed

forward network is applied to each token. This network expands the hidden size and applies a non linear activation before projecting back to the original dimension.

BERT is pre trained on two large language modeling tasks. The first is masked language modeling, where random tokens are masked and the model predicts them based on context. The second is next sentence prediction, which teaches BERT to understand relationships between sentences. These objectives allow BERT to learn general representations that are useful for downstream tasks.

For classification, the model uses the hidden representation of the special *[CLS]* token. BERT produces a vector for this token after the final encoder layer. A linear layer with two output units is added on top, and a softmax function converts these outputs into class probabilities. The model is fine tuned end to end on the implicit hate detection task, which means all layers update their weights besides the classifier.

5.2 Workflow

The workflow for training BERT on the implicit hate detection task includes data preparation, converting the data into model inputs, fine-tuning the pretrained network, and performing inference. The main steps are described below.

1. Data Processing

2. Tokenization

Each cleaned post is processed with the tokenizer that comes with the pretrained model. The tokenizer splits the text into smaller subword units and maps them to indices in its vocabulary. This step handles rare or unfamiliar words by breaking them into smaller pieces. The tokenizer also inserts special tokens. The *[CLS]* token appears at the start of the sequence to mark the classification target, and the *[SEP]* token marks the end.

3. Sequence Padding

The tokenized sequences are then adjusted to a fixed length of 128 tokens. Shorter sequences are padded and longer ones are truncated. This ensures that all samples have the same shape when they are sent into the model. Padding positions are ignored by the model through a mask, so they do not affect learning.

4. Dataset Splitting

The processed sequences are organized into three groups. The training set is used to update the model weights, the validation set is used to monitor overfitting, and the test set is used for final evaluation. Each set preserves the label distribution from the original data split to ensure a fair comparison across experiments.

5. Fine-tuning

We fine tune all layers of the model. Each batch is passed through BERT, and the classifier layer produces logits for the two classes. The loss is computed with cross entropy. For the class weighted experiment, we apply a weighted loss function to account for class imbalance. The optimizer is AdamW with a learning rate of 2e-5, which is a common choice for transformer fine tuning. The small learning rate to prevent large updates from damaging the pretrained knowledge. Early stopping is used to stop training when validation loss stops improving.

6. Early Stopping

During training, the model is evaluated on the validation set after each pass through the training data. The validation loss stops improving after a small number of passes, and early stopping is triggered around the third epoch. This prevents further overfitting, which is important because BERT can memorize the training set quickly when fine-tuned on small datasets.

7. Evaluation

After training stops, the version of the model with the lowest validation loss is used for testing. The model processes each post and produces a probability for the implicit hate class. These predictions are then compared with the ground truth labels to compute accuracy, precision, recall, F1 score, and the confusion matrix.

5.3 Experiments and Results

We evaluate BERT under three settings. The first uses the original imbalanced training set. The second applies class weights to adjust the loss during fine tuning. The third uses a downsampled training set in which the two classes appear in equal proportions. As in the LSTM experiments, early stopping activates around the third epoch for all three settings, which shows that the model reaches its best validation performance after only a small amount of training.

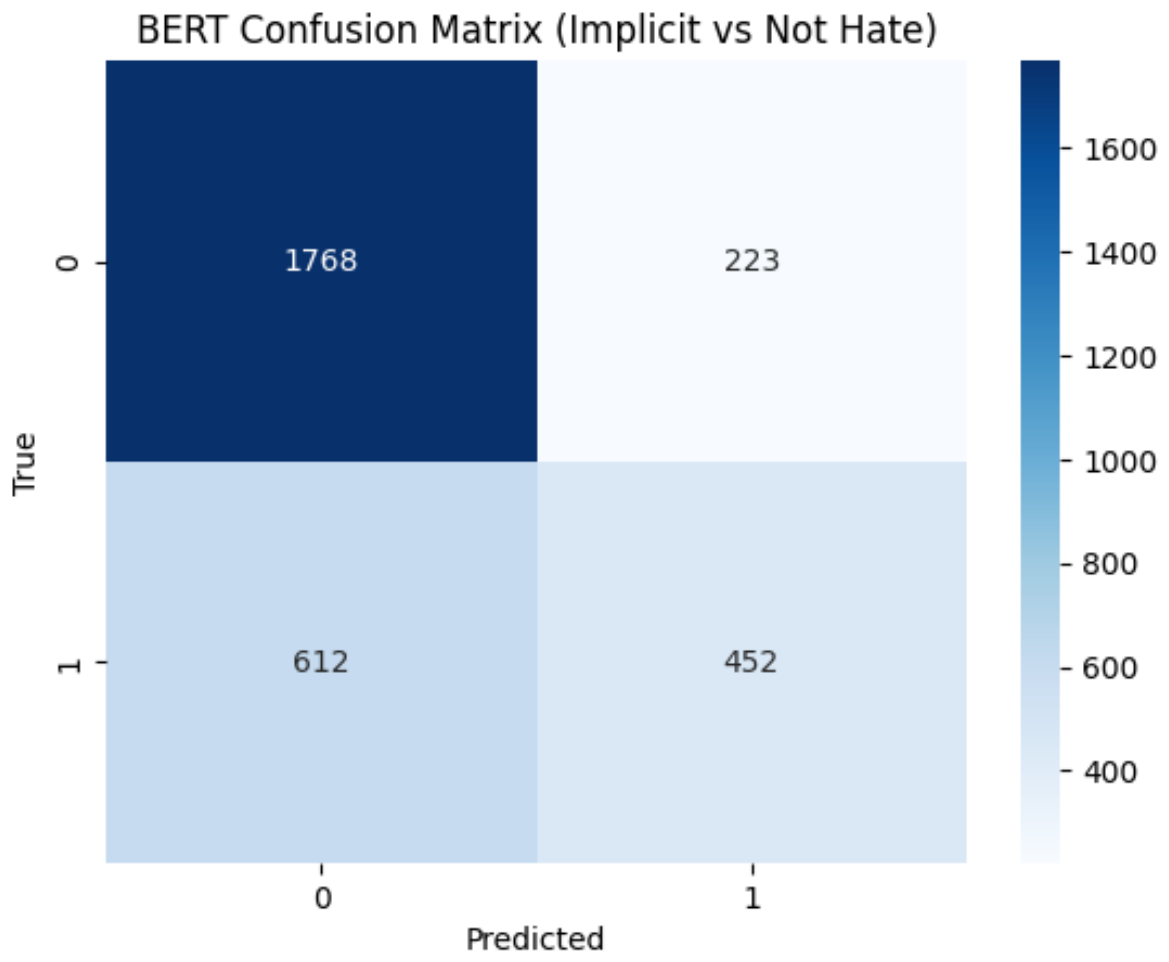
5.3.1 Original Dataset

```
BERT (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.7429    0.8880    0.8090      1991
           1     0.6696    0.4248    0.5198      1064

    accuracy                         0.7267      3055
   macro avg     0.7062    0.6564    0.6644      3055
weighted avg     0.7174    0.7267    0.7083      3055
```

When trained on the original dataset, BERT performs better than the LSTM baseline but still shows clear difficulty with implicit hate. The model fits the majority class well and reaches strong recall for non hate posts. However, recall for implicit hate remains limited. Many subtle examples are not identified, which suggests that class imbalance continues to affect the model during fine tuning.



Although BERT learns contextual relations more effectively than the LSTM, the imbalance still encourages the classifier to predict the majority label. The confusion matrix shows that a large number of implicit hate posts are classified as non hate. This confirms that even a strong contextual model requires additional adjustments when the training data is skewed.

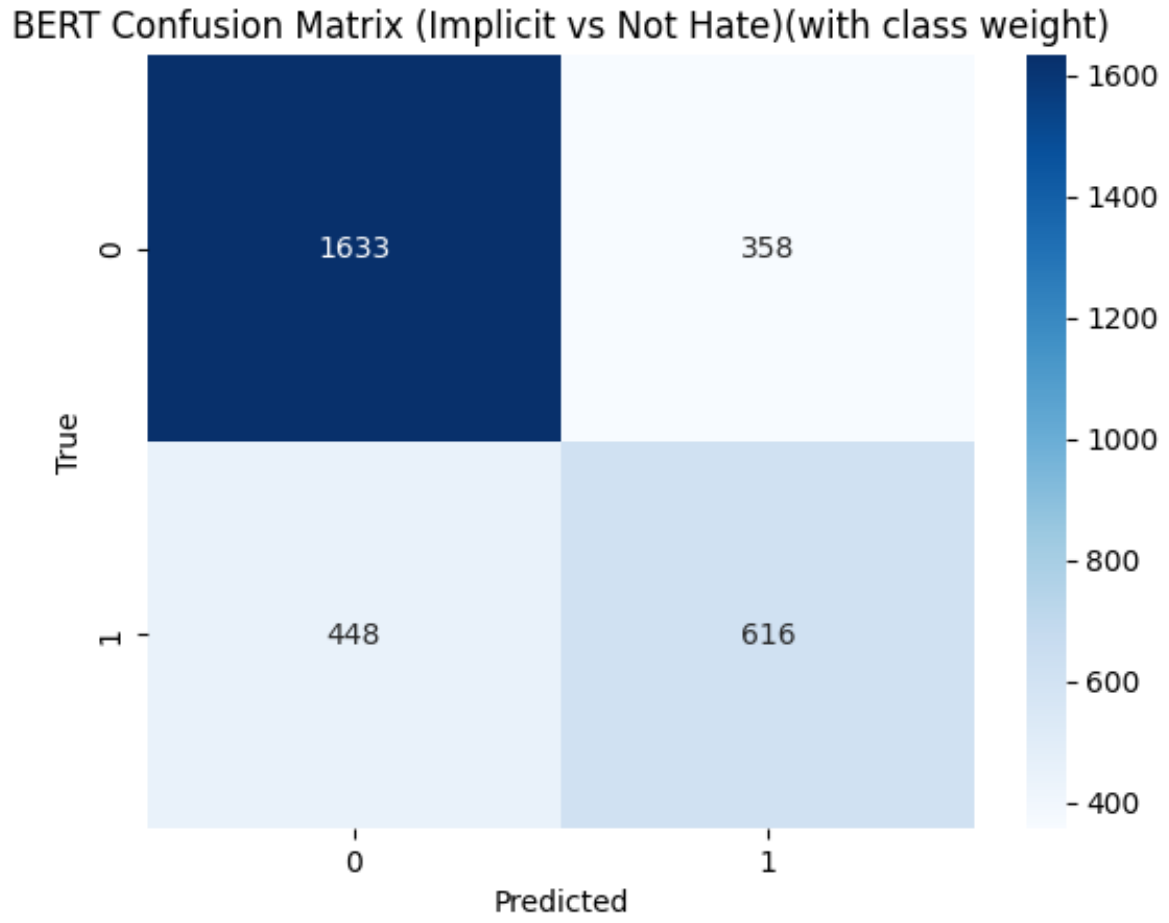
5.3.2 With Class Weights

Applying class weights shifts the model's attention toward the minority class. Errors on implicit hate examples have more influence during training, and this leads to a clear improvement in the recall for this class. The model becomes more willing to assign the hate label when the content contains indirect or coded expressions.

```
BERT (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support
```

0	0.7847	0.8202	0.8021	1991
1	0.6324	0.5789	0.6045	1064
accuracy			0.7362	3055
macro avg	0.7086	0.6996	0.7033	3055
weighted avg	0.7317	0.7362	0.7333	3055

There is a modest decrease in precision because the model predicts the hate label more frequently, but the overall balance between precision and recall improves. The increase in the F1 score for the implicit class shows that the model is capturing more subtle signals than before. The weighted average metrics also increase. This indicates that class weighting helps BERT correct the bias introduced by the skewed distribution.



5.3.3 Downsampling the Majority Class

Downsampling produces a balanced training set, which removes the bias toward the majority class. The results are close to those obtained with class weighting.

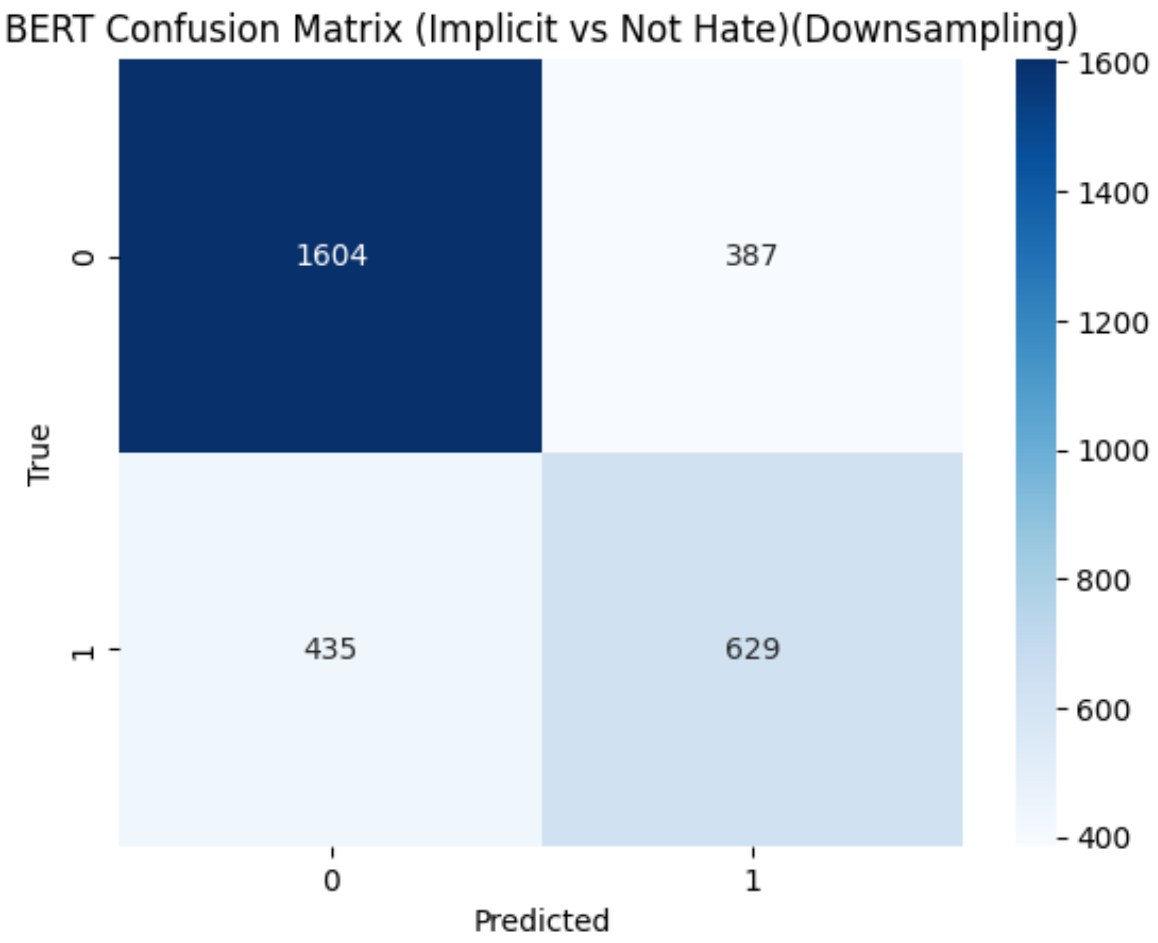
```
BERT (Implicit vs Not Hate) Report:
              precision    recall  f1-score   support

           0     0.7867    0.8056    0.7960      1991
           1     0.6191    0.5912    0.6048      1064

    accuracy                         0.7309      3055
   macro avg     0.7029    0.6984    0.7004      3055
weighted avg     0.7283    0.7309    0.7294      3055
```

The model reaches higher recall for implicit hate and makes more balanced predictions across the two classes. This shows that presenting the model with an equal number of examples for each class is enough to reduce the strong bias seen in the original dataset.

The slight decrease in precision for the minority class is similar to the pattern observed in the class weighted setting. Since both methods change the effective class ratio that the model sees during training, it is natural that they produce nearly identical trends. One difference is that downsampling reduces the amount of non hate data available during fine tuning, which can limit the diversity of examples the model learns from.



5.4 Summary

The experiments show clear differences between the two neural models. The LSTM model captures sequential patterns and learns word order relationships, but its ability to detect implicit hate is limited. Implicit hate often relies on subtle shifts in meaning, vague associations, or coded expressions, and these signals can be spread across the sentence in ways that are difficult for a single directional or even bidirectional recurrent model to summarize. As a result, the LSTM often misses minority class examples unless strong imbalance corrections are applied.

BERT performs better overall. Because it uses self attention, the model can examine relations between all words in a post at the same time, rather than processing the sequence step by step. This allows BERT to recognize indirect phrasing and contextual cues more effectively. In the experiments, BERT consistently achieves higher performance than the LSTM across accuracy,

recall, and F1 score, especially for the implicit hate class. The improvement is visible even when both models use the same imbalance handling strategy.

The impact of class imbalance is strong for both models. Without any correction, both models favor the majority class. However, BERT benefits more from imbalance handling. With class weighting or downsampling, BERT recovers many subtle cases and shows a more balanced prediction pattern. The LSTM also improves under these settings, but its overall performance remains lower because it has weaker contextual modeling ability.

Another difference between the two models is stability. The LSTM begins to overfit almost immediately, while BERT holds stable performance for a slightly longer period before early stopping activates. Although both models require early stopping, BERT preserves more of its generalization ability when the dataset is small and subtle patterns matter.

Overall, the comparison shows that contextual models like BERT provide a clearer advantage in detecting implicit hate. They are better suited to capturing the indirect and nuanced linguistic patterns that define this type of harmful language. LSTM models can offer reasonable performance, especially with proper imbalance handling, but they are limited by their ability to represent long range dependencies. BERT delivers stronger and more reliable improvements across all metrics.

6. Conclusion

This study examines the challenge of detecting implicit hate on social media, a form of hateful expression that avoids explicit slurs and instead relies on suggestion, coded language, or indirect framing. We evaluate three models of increasing complexity and semantic capacity: a TF-IDF Naive Bayes classifier, a BiLSTM network, and a pretrained BERT model. Each model is trained under three conditions, including the original imbalanced dataset, class weighting, and downsampling. Both class weighting and downsampling consistently improve model performance compared with the original setting, confirming that addressing class imbalance is essential for this task. Between the two imbalance handling strategies, class weighting offers a clear practical advantage because it improves minority class recall without discarding any training data. For this reason, it is used as the preferred approach for comparing the three models.

Our goal is to assess each model's ability to detect the semantic subtlety of implicit hate. We trade off between a high recall rate and a high precision rate, aiming for a model that neither overlooks potential implicit hate posts nor incorrectly flags a large number of normal posts as hateful. Therefore, we use the minority-class F1 score as the primary evaluation metric because it reflects the balance between identifying harmful posts and avoiding excessive false positives.

Model	Class 1 Precision	Class 1 Recall	Class 1 F1	Accuracy	Macro F1
Naive Bayes	0.5370	**0.6823**	0.6010	0.6845	0.6700
BiLSTM	0.5580	0.6100	0.5828	0.6959	0.6718
BERT	**0.6324**	0.5789	**0.6045**	**0.7362**	**0.7033**

Under this criterion, BERT achieves the strongest overall performance. It strikes a strong balance between higher precision and recall, which results in the highest F1 score among the three models. This suggests that the contextual representations learned by BERT allow it to capture the indirect or coded patterns that characterize implicit hate, even when explicit keywords are absent.

The Naive Bayes and BiLSTM models show a different pattern. Both models achieve higher recall on the implicit hate class, meaning they label more samples as hate. However, this comes with lower precision and more false positives. Their decision boundaries are more aggressive, and although this raises recall, the overall F1 score remains below BERT. These results highlight the limits of models that rely heavily on surface lexical features or sequential patterns without deeper contextual grounding.

Overall, the comparison shows that implicit hate detection requires models that can go beyond keyword matching and simple context windows. Pretrained language models are better suited for understanding subtle meaning shifts and indirect expressions. At the same time, handling class imbalance is critical for all models. Future work may explore data augmentation for implicit hate, contrastive learning objectives, or multi task setups that incorporate stance or sentiment as auxiliary signals. These extensions may help models capture even finer forms of indirect hostility and improve reliability in real social media environments.

7. Future Works

One interesting observation in our results is that Naive Bayes performed much better than we initially expected. We believe this is because the dataset consists mainly of short tweets—typically under 50 tokens. For short text, much of the signal that distinguishes implicit hate comes directly from specific lexical or phrasal cues. As a result, TF-IDF features become surprisingly effective, and Naive Bayes can capture these patterns with very simple decision boundaries.

In contrast, BERT's contextual advantage is less visible here. Transformers excel when either (1) inputs contain richer semantic structure, or (2) large training datasets allow the model to generalize subtle contextual distinctions. In our case, the dataset is relatively small, and the posts themselves are very short, which limits the amount of contextual information BERT can meaningfully extract. This also increases the risk of overfitting. We observed this directly during training: both the BiLSTM and BERT models converged within the first two to three

epochs, suggesting that the available data does not provide enough complexity or volume for these models to fully utilize their representational capacity.

For future work, a promising direction is to explore setups that allow contextual models to better leverage their strengths—such as using longer posts. It may help models capture the subtle meaning in implicit hate more effectively.

1. ElSherief, M. *et al.* Latent hatred: A benchmark for understanding implicit hate speech. in *Proceedings of the 2021 conference on empirical methods in natural language processing* 345–363 (Association for Computational Linguistics, 2021).
2. Caselli, T., Basile, V., Mitrović, J., Kartoziya, I. & Granitzer, M. I feel offended, don't be abusive! Implicit/explicit messages in offensive and abusive language. in *Proceedings of the twelfth language resources and evaluation conference* 6193–6202 (2020).
3. Badjatiya, P., Gupta, S., Gupta, M. & Varma, V. Deep learning for hate speech detection in tweets. in *Proceedings of the 26th international conference on world wide web companion* 759–760 (2017).
4. Mozafari, M., Farahbakhsh, R. & Crespi, N. A BERT-based transfer learning approach for hate speech detection in online social media. in *International conference on complex networks and their applications* 928–940 (Springer, 2019).
5. Japkowicz, N. & Stephen, S. The class imbalance problem: A systematic study. *Intelligent data analysis* **6**, 429–449 (2002).
6. Ihianle, I. *et al.* A deep learning approach for human activities recognition from multimodal sensing devices. *IEEE Access* **8**, 179028–179038 (2020).
7. Khalid, U., Beg, M. O. & Arshad, M. U. Rubert: A bilingual roman urdu bert using cross lingual transfer learning. *arXiv preprint arXiv:2102.11278* (2021).