# COMPARATIVE ANALYSIS OF VARIOUS TRANSFORMER ARCHITECTURES IN ROMAN URDU HATE SPEECH DATASET

MOAAZ SIDDIQUI1, ARBAZ SHAH2, ANAS ABID3, SYED OWAIS ALI4

*1Department of Software Engineering, NED University of Engineering and Technology*

*2Department of Software Engineering, NED University of Engineering and Technology*

*3Department of Software Engineering, NED University of Engineering and Technology*

*4Department of Software Engineering, NED University of Engineering and Technology*

[*siddiqui4301876@cloud.neduet.edu.pk*](mailto:siddiqui4301876@cloud.neduet.edu.pk%0dSHAH4300013@cloud.neduet.edu.pk%0dali4301500@cloud.neduet.edu.pk%0dhassan4305885@cloud.edu.pk)

[*SHAH4300013@cloud.neduet.edu.pk*](mailto:siddiqui4301876@cloud.neduet.edu.pk%0dSHAH4300013@cloud.neduet.edu.pk%0dali4301500@cloud.neduet.edu.pk%0dhassan4305885@cloud.edu.pk)

[*ali4301500@cloud.neduet.edu.pk*](mailto:siddiqui4301876@cloud.neduet.edu.pk%0dSHAH4300013@cloud.neduet.edu.pk%0dali4301500@cloud.neduet.edu.pk%0dhassan4305885@cloud.edu.pk)

[*abid4302404@cloud.neduet.edu.pk*](mailto:abid4302404@cloud.neduet.edu.pk)

# *ABSTRACT*

*The rise in usage of social media and digital technology has led to an increased usage of hate speech among many social media platforms like Facebook, Twitter, Instagram, and many others. As such, hate speech has also found its way into the regions of South Asia. Our research paper aims to use state-of-the-art transformer architectures and leverage the power of transfer learning to fine-tune these models on hate speech detection in the Roman Urdu language. We present a detailed review of the popular transformer architrctures like BERT, RoBERTa, and DistilBERT. We then provide our training and testing methodology and the results of the model in a tabulated. We see that out of all the three models mentioned, BERT performed well compared to its counterparts.*

**Keywords: Hate-speech detection, Deep Learning, Transformers, BERT, RoBERTa, DistilBERT**

# INTRODUCTION

Recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks have proven to be good in tasks related to natural language processing such as language modeling [1], sentiment analysis [2], named entity recognition [3], and many others. However, they have proven to be inefficient for longer input sequences. As such, different architectures were proposed such as encoder-decoder architecture [4]. In 2017, a ground-breaking transformer architecture was introduced which revolutionized the field of natural language processing [5]. Our paper uses transformer-based architectures for detecting hate speech in the Roman Urdu language.

The background study involves why transformers were introduced in the first place [5] and the drawbacks of RNNs and LSTMs. The literature review provides an in-depth study of these architectures with diagrammatic representations. The methodology contains the fine-tuning process we implemented, the dataset we used, and the evaluation metrics applied. Finally, we present our model evaluation metrics in the results sections in tabulated format. Lastly, there is always room for improvement and further research, which has been described in the last section of our paper.

# BACKGROUND

Recurrent neural networks (RNNs) were first primarily employed for applications involving natural language processing. Because they utilise feedback connections, where a neuron's output at one time step is given back as input to the same neuron at a subsequent time step, they are known as "recurrent" networks. As a result, RNNs may keep a hidden state that contains data about previous inputs in the sequence and can be utilised to inform predictions made by the network at subsequent time steps [6].

However, the "vanishing gradient problem" is a fundamental flaw in RNNs. [7] The gradients of the error function with respect to the parameters are propagated back through time during the training of RNNs, and as a result, they can get very tiny. The network may find it challenging to learn from and utilize information from lengthy sequences as a result. This is because, at each time step, the gradients are multiplied by the weight matrix, and if the weight matrix has eigenvalues below 1, the gradients will rapidly disappear [7].

The vanishing gradient issue with the conventional RNN was addressed with the introduction of the Long Short-Term Memory (LSTM) network. Input, forget, and output gates are used in LSTMs to regulate the flow of information into and out of the memory cell, which functions as a kind of information storage container. The LSTM can keep information over lengthy periods and circumvent the vanishing gradient problem because to the gates' ability to selectively store or forget information [8].

LSTMs do, however, have significant shortcomings. One is that because of the numerous characteristics in the gates and memory cells, they can be computationally expensive to train, especially when dealing with extremely lengthy sequences. The ineffectiveness of LSTMs in modelling long-range relationships is another significant flaw [5]. In other words, LSTMs might not be able to adequately capture dependencies that are very far apart in time between input and output. The length of the memory cell is set in LSTMs since they are built on a fixed architecture, which restricts their ability to describe long-distance relationships.

Due to all of these discrepancies, transformers were introduced in 2017 which revolutionized the field of natural language processing.

# LITERATURE REVIEW

## Transformers architecture

The transformer is a neural network architecture that was introduced in 2017 in the paper "Attention is all you need" [5]. It uses a set of encoder-decoder networks [9] that can take input sequences and generate output sequences. It has been widely accepted as de-facto architecture for many tasks related to natural language processing. On a high level, a transformer consists of simply an encoder; which is used to take input and generate word embeddings, and a decoder; which is used to output sequences or predictions. The figure below shows the transformer architecture on a high level:

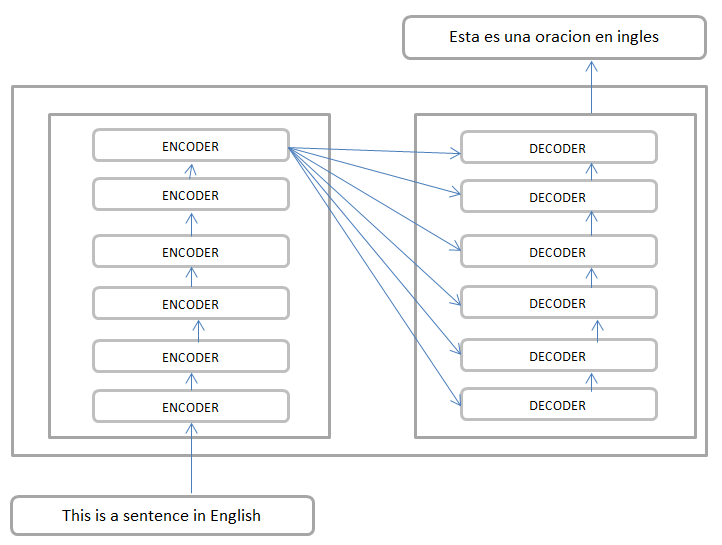


Figure 1: An illustration of simple transformer architecture in the context of language translation

### Word embeddings and positional embeddings

The words or sentences are firstly converted into vectors or embeddings using Word2Vec, and after that, it uses positional embeddings to determine the position of each word in an input sentence [5]. We use positional embeddings because the transformer architecture does not have any other way to determine the position of the words in the input sentence [5].

Let's say we have a sentence "I love to play football" and we want to provide positional information to the transformer model. We can assign a unique vector to each word in the sentence and also another vector to each word representing its position in the sentence. For example, the word "I" can be represented by the vector [0.1, 0.2, 0.3], and its position vector could be [0.4, 0.5, 0.6]. Then, we add these position vectors to the word vectors, so the transformer model takes into account both the meaning of the word and its position in the sentence when making predictions. In this example, the word vector for "I" is [0.1, 0.2, 0.3] and its position vector is [0.4, 0.5, 0.6], the final vector for "I" will be [0.1+0.4, 0.2+0.5, 0.3+0.6] = [0.5, 0.7, 0.9]. After this, the final vector is sent to the first encoder in the transformer model.

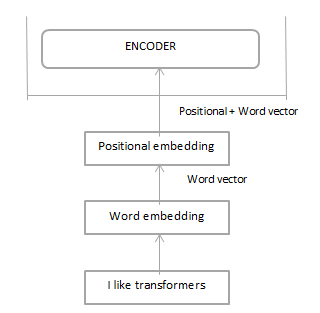


Figure 2: A figure showing how the input sequence is transformed before sending to encoder

### Encoder components

A self-attention layer plus a fully connected feed-forward neural network make up the encoder. A residual and a layer normalisation function come after each sub-layer (a self-attention and a feed-forward neural network) [5].

* ***Self-attention layer***

A transformer has a self-attention mechanism enables the model to concentrate on particular input data while producing the output. A series of matrices called "Key," "Query," and "Value" are created, multiplied, and passed to a Softmax function, which outputs the answer. To obtain the ultimate self-attention value, it is multiplied by the "value" parameter [5]. Following are the steps:

1. *Generate K, Q, and W by applying linear transformation as shown in the equations shown below:*

*K = W\_k \* X + b\_k*

*Q = W\_q \* X + b\_q*

*V = W\_v \* X + b\_v*

*where:*

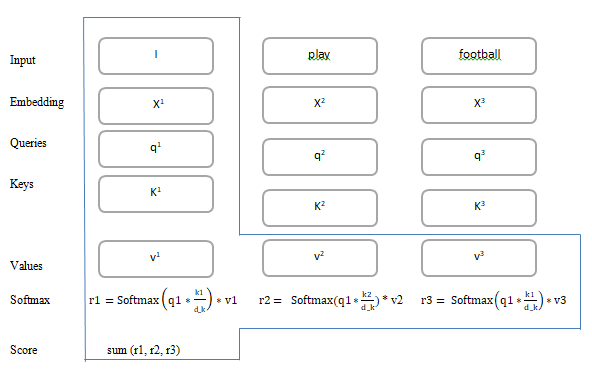
*X is a matrix where each row corresponds to an encoded-word representation in the input.*

*W\_k, W\_q, and W\_v are the weights matrices.*

*b\_k, b\_q, and b\_v are the biases during training*

1. *Calculate the similarity score between K and Q matrix by taking the dot product between Q and KT and dividing by the square of the dimension of the K matrix.*
2. *Apply a Softmax function to get a probability distribution.*
3. *Multiply the result by V to get the final result.*

For example, for the sentence, “*I play football”,* below is a pictorial representation for calculating self-attention for the word *“I”*:



* ***Multi-headed attention***

Multi-headed attention is a technique used in transformer models to increase the capacity of the self-attention mechanism [5]. In multi-headed attention, multiple self-attention mechanisms are applied to the input representation, each with their own key, query, and value matrices. Each self-attention head give resultant values which are then concatenated and

The values from each self-attention mechanism are concatenated and passed through a final linear transformation to produce a final value representing all of the individual attention heads[5]. In other words, Multi-headed attention allows the model to attend to different positions of the input with different learned linear projections, and thus increases the model's ability to capture complex relationships in the data. By applying multiple self-attention mechanisms in parallel, the model can attend to different parts of the input at the same time, which can be useful for modeling certain types of data or for fine-tuning the model's performance.

The number of heads (parallel attention mechanisms) used in multi-headed attention is a hyper-parameter that can be adjusted during the training process to optimize the model's performance on the task at hand.

1. *attention\_heads=[]*
2. *for i in range(length(attention\_heads)):*
   1. *Generate K\_i, Q\_i, and W\_i by applying linear transformation as shown in the equations shown below:*
   2. *Calculate the similarity score between K and Q matrix by taking the dot product between Q\_i and KT\_i and dividing by the square of the dimension of K\_i matrix.*
   3. *Apply a Softmax function to get a probability distribution.*
   4. *Multiply the result with V to get the final result.*
   5. *Attention\_heads[i] = score*
3. *Concatenate attention heads into a single dimensional vector*

* ***Feed-forward neural network***

The feed-forward neural network (FFNN) in transformers is a fully connected neural network that is applied to the output of the self-attention mechanism. The FFNN is used to further process the self-attention representation and produce the final output of the transformer layer. The FFNN typically consists of two linear transformations, which are implemented as fully connected layers, also known as dense layers, with learnable parameters such as weights and biases [10]. The first linear transformation is applied to the self-attention representation, and the second linear transformation is applied to the output of the first linear transformation. The output of the second linear transformation is then added to the self-attention representation as a residual connection.

The FFNN can be represented by the following equation. The W\_1, W\_2, b\_1, and b\_2 are the learnable parameters of the linear transformation.

output = self-attention + W\_2 \* relu (W\_1 \* self-attention + b\_1) + b\_2

* ***Residual connection and layer normalization***

The residual connection is used to bypass one or more layers if those particular layers do not improve the output and the latter is used to normalize the input to the next layer before it is passed through the layer's weights. This helps to stabilize the training process and reduce the amount of internal covariate shift, which can cause the model to converge more slowly or not at all.

For example, given a neural network with three layers, X, Y, and Z; the final output without a residual connection would be the output of the last layer C, but when we add a residual connection, the final output would be a combination of the last layer C and the original input. This helps to prevent the vanishing gradient problem.

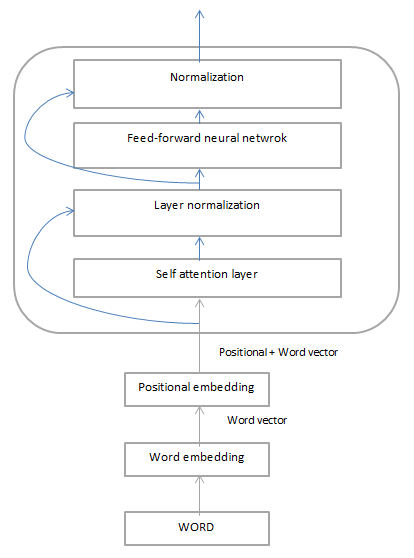
******

Figure 3: The diagram below shows the full process of how a single word is encoded and sent to the first encoder. The output of the first encoder is then passed to the second encoder in the architecture.

* + 1. **Decoder components**

The architecture of the encoder and the decoder are comparable. It comprises of a number of stacks of decoders, each of which has a feed-forward neural network and a self-attention layer. The residual connection, layer normalisation technique, and positional embedding are all used by the decoder just like they are by the encoder. Additionally, it makes use of a multi-headed self-attention mechanism to identify intricate connections between the data. The self-attention mechanism in the decoder incorporates both the output and the input sequence from the preceding layer, enabling the model to pay attention to both. In order to prevent the model from attending to subsequent tokens in the input sequence, it additionally employs a method known as "masked self-attention" to make sure that the output is created in the correct order.

* ***Encoder-Decoder attention layer***Top of Form

A set of attention vectors K and V are created from the top encoder's output. Each decoder in the "encoder-decoder attention" layer should use these to compute attention scores with its own query vector (Q), which enables the decoder to concentrate on the necessary portions of the input sequence and produce the output. Each encoder's output is passed to the bottom-most decoder first, which then begins processing the inputs just like the encoder did.

In other words, the decoder's self-attention layer enables the model to focus on its own output, while the encoder-decoder attention feature enables the decoder to focus on the encoded version of the input produced by the encoder.

A vector of floating-point numbers produced by the final decoder is then given to a fully integrated neural network. This is done in order to transform the output into a huge vector known as "logits." Each cell shows a word's score individually. The final Softmax layer receives the logits and produces a vector of probabilities ranging from 0 to 1. The final result is the cell with the highest probability. There are a total of 6 encoders and 6 decoders in the architecture, according to the paper [5]. However, this is a hyper-parameter that can be changed.

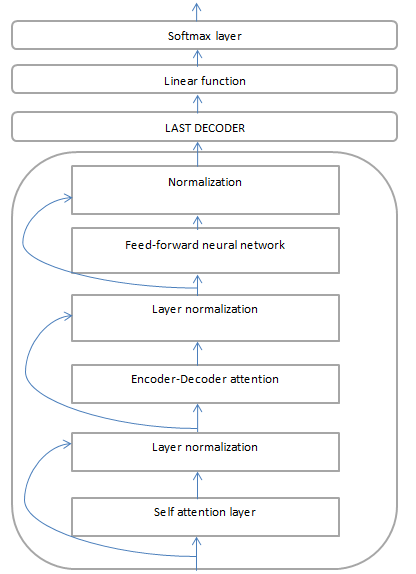


Figure 4: Decomposition of a single decoder in the decoder component of transformers

## BERT architecture

It is a cutting-edge pre-trained transformer model, called BERT (Bidirectional Encoder Representations from Transformers), and trained on a sizable corpus of text. In contrast to vanilla transformers, it is bi-directional, which enables it to anticipate a specific word by studying its context in both the left and right directions [11]. This makes it more precise than standard transformers.

The original work describes two steps of BERT: pre-training (during which the model picks up the language and context from a large corpus) and fine-tuning (where the model is trained on a labeled dataset for a specific task) [11]

### Pre-training BERT

In pre-training, we use two special techniques, masked language modeling, and next-sentence prediction.

* ***Masked Language Modelling:***

Using this method, about 15% of the input texts' words are changed (masked) to a [MASKED] token. After then, the model must correctly predict the first token by learning about the context of previous tokens (unmasked ones). For tasks like text categorization and named entity recognition, BERT may use this technique to determine the meaning of words in a phrase and their context. The loss function only considers predictions for those phrases, so the model can better understand the context and is additionally punished if it predicts the [MASKED] words incorrectly [11].

* ***Next-Sentence Prediction:***

In this approach, a pair of sentences is provided to the model as input. The second stage is to determine if the second statement of the pair logically follows the first. For training purposes, 50% of the input pairs have the second sentence being the following one in the original text, and 50% have the second sentence being a randomly chosen sentence from the corpus. This enables BERT to learn more about the context of a sentence, which is important for tasks like answering questions [11].

* ***Input and output representation***

BERT accepts input in a particular format. It employs three different forms of embeddings: a positional embedding, a sentence embedding (which helps us determine which sentence pair the input text belongs to), and a token embedding (the paper employed WordPiece embeddings of vocabulary size 30,000) which helps to preserve the order of the tokens [11]. Additionally, unique tokens are added to each pair of sentences. The [CLS] token is used to denote the beginning of a sentence, and the [SEP] token is used to denote the conclusion of a sentence.

After pre-training, the BERT model outputs a fixed-length vector representation of the input text. The meaning of the input text and the connections between words in the context of the full input sequence are captured by this vector, also referred to as the "encoded representation" or "embedding."

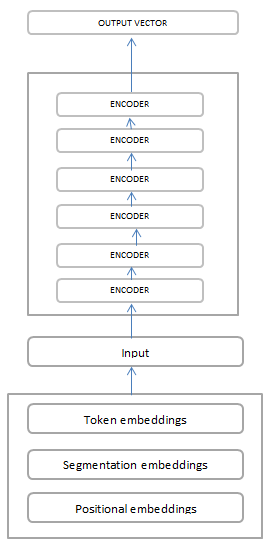


Figure 5: BERT model architecture

### Finetuning BERT

A task-specific layer (a Softmax layer with as many neurons as the vocabulary size) is added on top of the pre-trained model when BERT is fine-tuned for a particular job. With a few minor adjustments, the pre-trained model parameters remain unchanged. After being fine-tuned on a dataset that has been appropriately labelled, the model then generates the desired output for that task. The output of BERT, for instance, would be the probability distribution over the number of classes in the case of a text classification problem.

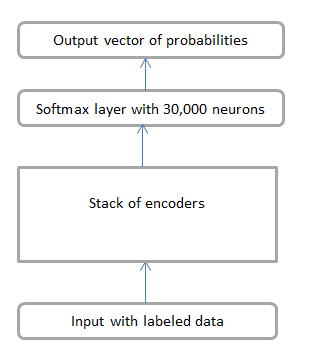
**­­**

Figure 6: Output from BERT model

## RoBERTa architecture

RoBERTa (a robustly optimized training approach) is an enhanced variation of the original BERT architecture. It has the same encoder and decoder componenets as that of BERT, with a few changes in hyperparameters and some training tasks [12]. Following are the changes in the RoBERTa model:

### Dyanmic Masking

BERT used to perform masking on the training dataset only once before training, and then that masked data was duplicated muliple times and masking was done at different positions in every training instance. This was done to avoid training the model to predict the same mask at every epoch. RoBERTa uses a technique called Dynamic masking in which it masks the input seqeunces as it is fed to the model as input during training. Static masking gave an F1-score of 78.3on the SQuAD 2.0 dataset while dynamic masking gave an F1-score of 78.7 [12].

### Next Sentence Prediction

BERT originally used to do masked language modelling. In recent studies, it was found that it might or might not affect the model performance. Therefore, this hypothesis was thoroguhly tested in RoBERTa and it was found that removing NSP resulted in better training accuracy. The original BERT architecture gave a performance of 88.5/76.3 on the SQuAD 2.0 dataset while RoBERTa gave a score of 90.6/79.7 [12].

### Byte-Pair Encoding

Byte Pair Encoding (BPE) is a technique that is used to learn a set of subword units (often called "tokens") that can be used to represent the text. In the case of RoBERTa, BPE is used to segment the text into a set of subwords, which are then used as the input to the model [12].

The intuition behind BPE is to begin with a set of characters and iteratively merge the most frequent pair of characters or subwords until a certain vocabulary size is reached. The resulting subword units are chosen based on the frequency of their occurrence in the text. This allows BPE to learn a set of subwords that are more likely to be useful for the task at hand, as they are based on the actual distribution of characters and subwords in the text.

* 1. **DistilBERT**

DistilBERT is a light weight version of the original BERT architecture. The reason it was introduced was to make a ligh-weight, scalable version of the BERT architecture, that when deployed, can easily run on mobile applications and website with low latency and fast response time [13]. However, this reduction in size does not affect the learning and efficiency of the model. DistilBERT is 60% more faster than BERT, has 40% less parameters than BERT, and also retains 95% of the original BERT knowledge.

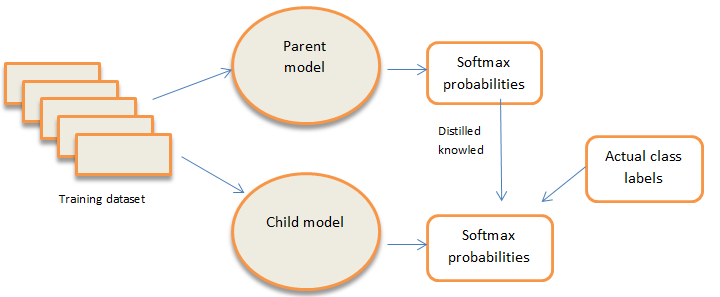
­­

Figure 7: RoBERTa architecture

* + 1. **Knowledge Distillation**

It is a technque in which a smaller, light-weight model is trained to produce the outputs like that of a larger model. The concept is that information from the parent model (specifically, output from the softmax layer) is fed into the child model and then it is trained with a weighted cost function to minimze the error [14]. The parameters of the parent model are frozen since it is already trained, we are simply transferring ite results to the child model, and trianing the child model on a combined loss function of both the parent and child model.

* + 1. **Softmax with Temperature (T)**

In both the teacher and child model, we use a softmax function with a temperature value T, which is a hyper paramaeter that can be scaled. It is represented by the following formulae, for the parent and child model, respectively [13]:

For the loss function in DistilBERT, two loss functions are used:

* A cross-entropy loss is used between the logits from the teacher model with softmax-temperature and the logits from the child model with softmax-temperature.
* a cross-entropy loss between the child logits and the actual target values

These two loss functions are combined to give a new loss called “Distillation loss”, shown in the formula below:

A masked language modelling loss is used (similar to BERT) and a cosine embedding loss is also added. All these three losses are combined and a weighted average is taken to create a final loss fucntion, on which the model is trained.

# METHODOLOGY AND IMPLEMENTATION

## Dataset used

The Roman Urdu Hate-Speech and Offensive Language Detection (RUHSOLD) dataset [15] was used in this paper. It is a dataset consisting of two parts relating to hate speech detection. First part contains a coarse grained dataset of roman Urdu tweets taken from Twitter with only 2 classes. The second part contains roman Urdu tweets taken from Twitter but with 5 classes. As such, for a detailed analysis, we used the second one. The 5 classes were,  
Normal (1), Abusive/Offensive (0), Religious Hate(2), Sexism(3), and Profane/Untargeted(4). The dataset had three parts, “train” containing about 7000 examples, “test” containing about 2000 examples, and “validation” set containing about 7000 examples.

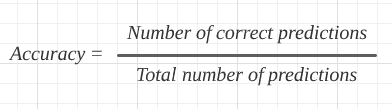
In order to clean the dataset, we used regular expressions techniques to remove any sort of emojis or unwanted spaces. We decided to remove emojis because they might not express the real meaning of a tweet and may add unwanted noise to the dataset.

## Accuracy metrics used

For our usecase, we used accuracy, precision, recall, and F1-score as our accuracy metrics. They are described below:

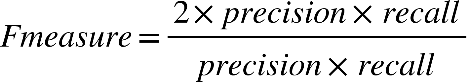
* + 1. **Accuracy**

The most basic model evaluation metric is accuracy, although not a very good one. Its mathematical formula is:

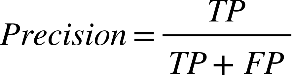


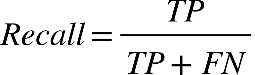
* + 1. **F1-score**

It is the most common and preferred type of model evaluation metric, also called F-score.



where:





# RESULTS

The following table shows the 4 evaluation metrics that we used on the testing datasets.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-score |
| BERT | 97.2 | 98.89 | 97.54 | 98.2 |
| RoBERTa | 96.5 | 98.71 | 92.13 | 95.3 |
| DistilBERT | 96.8 | 96.43 | 95.92 | 96.17 |

# CONCLUSION AND FURTHER RESEARCH

Hate speech detection is a common task in the field of natural language processing, yet it is one of the most difficult tasks. There are cases where people express sarcasm or other emotions that can easily fall under the umbrella of hate speech, but often deep learning models fail to classify them properly. Moreover, the dataset we used was small in size, as we did not have enough computing resources to handle large amounts of data. Moreover, we did not do any sort of hyperparameter optimization in the fine-tuning process either, and used the default parameters of the models only. Further research could include a large dataset with more sentiments. Our work can also be extended to include hyperparameter tuning in the fine-tuning process as well.

The code is available at the following link: https://github.com/moaaz12-web/Hate-speech-detection-transformers

# REFERENCES

[1] Soutner, D., and Müller, L. "Application of LSTM neural networks in language modelling," in International Conference on Text, Speech and Dialogue, pp. 105-112, Springer, Berlin, Heidelberg, September 2013.

[2] Rojas-Barahona, L. M. "Deep learning for sentiment analysis," Language and Linguistics Compass, vol. 10, no. 12, pp. 701-719, 2016

[3] Arkhipov, M. Y., and Burtsev, M. S. "Application of a Hybrid Bi-LSTM-CRF model to the task of Russian Named Entity Recognition," in Conference on Artificial Intelligence and Natural Language, pp. 91-103, Springer, Cham, September 2017.

[4] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. "On the properties of neural machine translation: Encoder-decoder approaches," arXiv preprint arXiv:1409.1259, 2014.

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. "Attention Is All You Need," ArXiv, 2017. [Online]. Available: <https://doi.org/10.48550/arXiv.1706.03762>.

[6] Sherstinsky, A. "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," Physica D: Nonlinear Phenomena, vol. 404, pp. 132306, 2020.

[7] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 6, no. 2, pp. 107-116, 1998

[8] DiPietro, R., and Hager, G. D. "Deep learning: RNNs and LSTM." In Handbook of medical image computing and computer assisted intervention, pp. 503-519. Academic Press, 2020.

[9] Ghojogh, B., and Ghodsi, A. "Attention mechanism, transformers, bert, and gpt: Tutorial and survey," 2020.

[10] Bebis, G., and Georgiopoulos, M. "Feed-forward neural networks," IEEE Potentials, vol. 13, no. 4, pp. 27-31, 1994.

[11] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," arXiv preprint arXiv:1907.11692, 2019.

[13] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," arXiv preprint arXiv:1910.01108, 2019.

[14] J. Gou, B. Yu, S. J. Maybank and D. Tao, "Knowledge distillation: A survey," International Journal of Computer Vision, vol. 129, no. 6, pp. 1789-1819, 2021.

[15] U. Azam, H. Rizwan, and A. Karim, "Exploring Data Augmentation Strategies for Hate Speech Detection in Roman Urdu," in Proceedings of the Thirteenth Language Resources and Evaluation Conference, 2022, pp. 4523-4531.