Fine-Tuning HateBERT for Hate Speech Classification

Victor-Andrei Căbulea, Marian Flămînzanu-Mateiuc
Dept. of Computer Science and Engineering
Faculty of Automatic Control and Computer Engineering
"Gheorghe Asachi" Technical University of Iasi
Iasi, Romania
{victor-andrei.cabulea, marian.flaminzanu-mateiuc}@student.tuiasi.ro

***Abstract*—This study investigates the benefits of further fine-tuning the HateBERT model for hate speech classification. Our results demonstrate that, despite the additional computational cost, fine-tuning significantly improves classification accuracy. These findings underscore the importance of task-specific adaptation of pre-trained language models to enhance performance on complex and nuanced NLP tasks such as hate speech detection.**

***Index Terms*—hate speech classifying, natural language processing, fine-tuning, large language model used as a judge**

I. Introduction

In today's digitally connected world, online platforms have become the central point for communication, social interaction, and information exchange. However, the proliferation of hate speech on social networks and other digital forums poses a threat to societal harmony and individual well-being. Hate speech, defined as language that discriminates, threatens, or incites violence against individuals or groups based on attributes such as race, religion, gender, or ethnicity, not only fosters hostility, but also contributes to real-world harm and social polarization.

To effectively combat this challenge, automated detection systems are essential. This project aims to harness these capabilities to create a reliable hate speech binary classification tool that can be deployed on social platforms and moderation systems. By improving the precision of hate speech identification, such tools can help reduce the spread of harmful content, protect vulnerable communities, and create safer online environments.

II. State-of-the-art - short review

A. The Evolution of Hate Speech Detection

Hate speech detection has grown as a vital research area due to its societal and psychological impact. Early methods relied on traditional machine learning, such as SVMs and logistic regression with hand-crafted features, which struggled to capture linguistic nuances.

The advent of deep learning and transformers, especially BERT, marked a significant advance by effectively modeling the bidirectional context [1]. Subsequent models such as RoBERTa and ALBERT improved robustness and efficiency [2]. Domain-adapted transformers such as HateBERT, pre-trained on hateful language corpora, further enhanced detection accuracy [3].

Hybrid approaches that combine transformer embeddings with CNNs or RNNs were also proposed to better capture semantics and sequence patterns [4], [5]. Models like ToxicBERT and T5 demonstrated strong results in toxicity and hate speech classification, supporting nuanced and multi-label moderation.

Despite these advances, challenges persist in detecting implicit hate, reducing bias, and generalizing across domains.

B. The architecture of BERT model

BERT (Bidirectional Encoder Representations of Transformers) is a deep learning model developed by Google in 2018 [1]. It is based entirely on the Transformer architecture and is designed to pretrain deep bidirectional representations by jointly conditioning on both left and right contexts in all layers. BERT consists only of the encoder part of the transformer architecture [6]. It is not autoregressive, so it does not generate text and is deeply bidirectional, capturing contextual information more effectively than the unidirectional models. There are two versions of BERT:

- **BERT-base:** 12 layers, 12 attention heads, 768 hidden size
- **BERT-large:** 24 layers, 16 attention heads, 1024 hidden size

Each layer includes a multihead self-attention mechanism, a feed-forward neural network, layer normalization and residual connections [6]. Each token in the input is represented by the sum of token embeddings, segment embeddings and position embeddings. BERT uses self-attention to compute a representation of each word in context.

Given queries Q, keys K, and values V, the attention function is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

Multi-head attention is used to allow the model to jointly attend to information from different representation subspaces.

Each encoder layer also contains a fully connected feed-forward network applied to each position independently:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \quad (2)$$

The final hidden states of the encoder are contextualized embeddings for each input token. The output embedding of the [CLS] token is used for classification tasks. Token-level embeddings are used for tasks such as recognizing named entities.

C. The improvements brought by HateBERT model

HateBERT was initialized from BERT-base and then further pre-trained on more than 1 million posts from Reddit's "r/hatereads" subreddit. This additional domain-specific pre-training helps HateBERT better capture the linguistic characteristics of hateful and offensive language, making it more effective for tasks such as hate speech detection, offensive language classification, and related toxic content analysis [3].

III. Data preprocessing

A. Dataset Collection and Harmonization

We integrated four hate speech and offensive language datasets, all accessed via the HuggingFace library.

1) HateXplain: A multi-annotator dataset where each post is labeled by three annotators and comes with a rational justification. We extracted the primary textual content from the "post_tokens" field and derived binary labels by majority voting over the annotators" hate/offensive label annotations [9].

2) Davidson Dataset: This dataset includes tweets labeled with a continuous hate speech score. We binarized this by thresholding: tweets with a hate speech score higher than 0.5 were considered hateful, and others were considered non-hateful [10].

3) Offensive Language Identification Dataset (OLID): We combined both the training and test splits of OLID, mapping the 'subtask_a" labels into binary values: "OFF" as 1 (for the offensive ones) and "NOT" as 0 (for the non-offensive ones). The textual content was taken from the "tweet" field.

4) Social Bias Inference Corpus (SBIC): We extracted the main post from the "post" field and created a binary label based on the "offensiveYN" value. Posts labeled as "1.0" (offensive) or "0.5" (maybe offensive) were assigned a label of 1 (offensive), while posts labeled as "0.0" (not offensive) were assigned a label of 0 (not offensive) [7], [11].

Each dataset was converted into a unified schema with two columns: text and label. After concatenation, we performed deduplication and dropped entries with textual length below a threshold of 5 characters to maintain content quality.

B. Text Cleaning and Normalization

To mitigate the noise and variation inherent in social media data, we implemented a comprehensive function to clean the text, converting the text to lowercase, Unicode normalization ("NFKD" form), removed the non-ASCII characters and stripped of special tokens such as usernames, URLs, hashtags and hyperlinks. We utilized the Python package "contractions" to expand common contractions (for example, "don"t" converted to "do not"). Then, we applied regex-based replacements to map common informal internet language and slang (for example, "dis" converted to "this", "dat" converted to "that", "idc" converted to "I do not care", "cus" converted to "because" and so on). Words with exaggerated character repetition were reduced to a standard form (for example, 'soooo" $\rightarrow$ 'so") using a vocabulary-aware approach, preserving semantically valid words from the "NLTK" English vocabulary list. Removal of irrelevant punctuation, extra whitespace, and malformed characters while retaining basic punctuation for semantic clarity. This multi-stage cleaning process was applied to both the original and augmented texts to ensure uniformity across the dataset.

C. Data Augmentation via Synonym Replacement

To enhance data diversity and model robustness, we incorporate synonym replacement using WordNet-based augmentation. For each original sentence, we randomly select one or more words and attempt to substitute them with semantically equivalent synonyms, excluding replacements that match the original term in stem or orthographic form. This augmentation considered only words with available WordNet synsets (synonym set). Synonyms had to be different from the original word and contextually appropriate. Augmented sentences were re-cleaned and included in the final dataset only if they were sufficiently different from the source and retained semantic validity. The resulting augmented samples were merged with the original dataset, effectively increasing the linguistic variability of the dataset, doubling the dataset and preserving the labels of the initial examples, contributing to improved generalization during model training. The final dataset has a total of 205692 samples.

D. Tokenization

Unlike traditional tokenization approaches that operate strictly at word or character levels, the WordPiece tokenizer performs subword tokenization, breaking down text into smaller, semantically meaningful units [6]. Subword tokenization balances vocabulary size and expressiveness by splitting rare or complex words into subword units, enabling the model to effectively handle out-of-vocabulary terms. For example, the word "unhappiness" may be tokenized as ["un", "##happi", "##ness"], where the prefix "##" signifies that the token is a continuation of the previous subword [1].

To ensure uniform input dimensions, each input text is tokenized with truncation and padding to a fixed maximum sequence length. Although some sequences can be as long as 350 tokens, our empirical analysis showed that the 95th percentile of tokenized input lengths is approximately 63 tokens, with a mean length of around 29 tokens. Therefore, a maximum sequence length of 128 tokens was selected as a trade-off between efficiency and coverage. This choice ensures that more than 95% of the samples are retained without truncation, while reducing memory usage and computational

cost during training. Using 64 tokens would have likely lead to unnecessary truncation for a non-trivial portion of data, and might have hurt performance.

The tokenizer outputs token indices "input_ids" and an "attention_mask" to distinguish real tokens from padding. These are converted into PyTorch tensors alongside the target label, enabling batching and training on the GPU.

The dataset is divided into stratified training and test sets using an 85/15 split. Furthermore, 15% of the training data is reserved for validation purposes. To counteract class imbalance, a weighted random sampler is used. The sampling weights are derived from the inverse class frequencies, ensuring that underrepresented classes are more likely to be included in each batch.

IV. Fine-Tuning the HateBERT model

We developed a modular training framework designed for the classification of hate speech using transformer-based architectures. Our framework builds on the pre-trained HateBERT model, integrating training strategies and diagnostic tools to ensure good performance in noisy and imbalanced social media data [3].

A. Model Initialization

The model initialization process begins with loading the pre-trained GroNLP/hateBERT architecture. For the purpose of hate speech detection, the model is configured for binary classification (distinguishing between hateful and non-hateful content) by setting the number of output labels to two.

During initialization, both the attention dropout and hidden layer dropout rates are configured to 0.2. These dropout mechanisms serve as regularization techniques to mitigate overfitting, particularly important when fine-tuning the model on smaller or imbalanced datasets, as is common in hate speech classification tasks.

To allow the model to discard the incompatible pre-trained classification head and initialize a new one dedicated to the current binary classification setting without raising errors, the parameter "ignore_mismatched_sizes" is set to "True".

To balance between leveraging pre-trained knowledge and enabling task-specific adaptation, a selective layer freezing strategy is employed. Specifically, all transformer encoder layers except the final one (12th layer) are frozen during the initial training phase. This allows the model to retain general language understanding acquired during pre-training while enabling the last layer to adapt to the target domain's task-specific signals. Additionally, the classification head, which maps the contextualized representations to label logits, is fully trainable from the start.

The embedding layer is also kept trainable. This decision provides flexibility in refining token embeddings to better capture vocabulary nuances that may be unique to the hate speech domain, including informal language, slang, or coded terms. By fine-tuning the embeddings and the final transformer layer, the model can effectively adapt to the linguistic and semantic characteristics of the new domain without the risk of catastrophic forgetting that may arise from fully updating all layers too early in training [3].

B. Optimizer and Learning Rate Scheduling

The training configuration employs the AdamW optimizer, which is particularly well suited for fine-tuning transformer-based architectures because of its decoupling of weight decay and gradient update. The learning rate is set to $2e-5$, a commonly adopted value that promotes stable and gradual convergence during training [1]. A weight decay is applied with a coefficient of $1e-4$ to regularize the model and reduce the risk of overfitting by discouraging excessively large weights [8]. An epsilon value of $1e-8$ is used to avoid division-by-zero errors during the optimization steps.

To further improve convergence and stability, a linear learning rate schedule is used with a warm-up phase comprising the first 10% of the total training steps. During this phase, the learning rate increases linearly from zero to the base value, helping to mitigate sudden shocks to the model weights at the beginning of training. After the warm-up phase, the learning rate linearly decays to zero over the remaining steps [8].

The model is trained with a batch size of 32. However, due to memory constraints, a gradient accumulation over two steps is used to simulate a larger effective batch size of 64. This approach allows for the benefits of a larger batch size (for example, more stable gradient estimates) without exceeding hardware limitations. To address the issue of exploding gradients, which can destabilize training, gradient clipping is applied with a maximum norm of 1.0. This ensures that the gradient norms are constrained, maintaining the stability of training even in the presence of sharp loss landscapes.

C. Loss Function and Class Imbalance Handling

To effectively address the issue of class imbalance, we use a custom implementation of the Focal Loss function. Traditional classification tasks often rely on the cross-entropy loss, which assumes balanced class distributions. However, in hate speech datasets, the majority of samples typically belong to non-hateful or neutral categories, causing the model to become biased toward these overrepresented classes and neglect the minority hateful instances.

The Focal Loss modifies the standard cross-entropy objective to place greater emphasis on difficult or misclassified examples, introducing two key parameters: the class balancing factor α and the focus parameter γ. In this configuration, α is set to 0.25, which assigns higher importance to the minority class, compensating for the skewed class distribution. The focusing parameter γ is set to 2.0, which reduces the relative loss contribution from well-classified or easy examples. As a result, the model concentrates its learning capacity on more difficult examples that are more likely to be misclassified, such as borderline or implicit hate speech.

Mathematically, the Focal Loss for a binary classification task is defined as:

$$\mathcal{L}_{\text{focal}} = -\alpha_t (1 - p_t)^{\gamma} \log(p_t) \tag{3}$$

where p_t is the predicted probability of the true class, α_t is the weighting factor for class t, and γ adjusts the rate at which easy examples are down-weighted.

D. Gradual Unfreezing Schedule

To promote stable and effective fine-tuning of the pre-trained transformer model, a gradual unfreezing strategy is employed. This technique is particularly useful when transferring a large pre-trained model to a task with limited training data, as it mitigates the risk of catastrophic forgetting and overfitting by controlling which parts of the model are allowed to update during different stages of training.

Initially, most of the transformer layers are frozen, meaning that their weights are kept constant and excluded from gradient updates. Only the final transformer layer (layer 11), the classification head, and the embedding layer are unfrozen and actively trained. As training progresses, additional layers are incrementally unfrozen following a predefined schedule, allowing the model to gradually adapt deeper parts of its architecture to the task-specific data.

At each milestone epoch (2, 4, 6, 8, and 10), progressively larger subsets of transformer layers are unfrozen. Specifically:

- At epoch 2, layers 10 and 11 are unfrozen.
- At epoch 4, layers 8 to 11 are unfrozen.
- At epoch 6, layers 6 to 11 are unfrozen.
- At epoch 8, layers 4 to 11 are unfrozen.
- At epoch 10, all 12 transformer layers are unfrozen.

This progressive adaptation allows the model to begin with shallow, task-specific tuning and gradually extend fine-tuning to deeper layers, enabling more global adaptation as training advances.

After each unfreezing step, the optimizer is reinitialized to incorporate the newly trainable parameters. This ensures that appropriate learning rates and gradients are applied to the expanded parameter set [8].

E. The Training Loop

Training is carried out using mixed arithmetic precision via torch.cuda.amp, which leverages automatic mixed precision (AMP) to reduce memory consumption and speed up computation while preserving numerical stability. This approach allows for larger batch sizes and faster training without compromising model performance.

To avoid overfitting, early stopping is employed with patience for three consecutive epochs. If the validation performance does not improve within this window, the training is stopped automatically. This ensures that the model retains generalization capability without over-adaptation to the training set.

F. Model Evaluation

Initially, a baseline is established by evaluating the pre-trained `HateBERT` model without fine-tuning. This sets a reference point for measuring the effectiveness of domain adaptation and task-specific training [1].

During fine-tuning, validation performance is assessed after every epoch using indicators such as loss, macro-averaged F1-score, Area Under the Receiver Operating Characteristic Curve (ROC AUC) and Precision-Recall AUC (PR AUC). Although the loss serves as the primary optimization signal, the macro F1-score and AUC offer complementary insights into model performance, particularly in scenarios involving class imbalance.

a) Macro F1-Score: measures the harmonic mean of precision and recall for each class independently, and then averages them:

$$\text{F1}_{\text{macro}} = \frac{1}{C} \sum_{i=1}^{C} \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{4}$$

where C is the number of classes.

b) ROC AUC: reflects the area under the Receiver Operating Characteristic curve, which plots the true positive rate (TPR) against the false positive rate (FPR) at various thresholds:

$$\text{TPR} = \frac{TP}{TP + FN} \tag{5}$$

$$\text{FPR} = \frac{FP}{FP + TN} \tag{6}$$

c) PR AUC: represents the area under the Precision-Recall curve, which is particularly informative in imbalanced settings:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{7}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{8}$$

The final evaluation phase is conducted on a held-out test set to report the model's generalization ability. In addition to quantitative evaluation, a curated set of manual test cases is used for qualitative analysis. These cases include examples of coded, sarcastic and implicit hate speech to assess the robustness of the model to linguistic nuances.

V. Classification

To perform inference, we define a predict function that supports both single-instance and batch predictions using a fine-tuned transformer-based hate speech classification model.

The input is tokenized using a pre-trained subword tokenizer, with truncation and padding applied to ensure uniform sequence lengths. A fixed maximum sequence length of 128 tokens is used to balance memory efficiency with context retention. Special tokens are automatically added to match the format expected by the model, and token type IDs are omitted for simplicity.

During inference, the model is set in evaluation mode and run with mixed arithmetic precision if the device is a CUDA-capable GPU. The forward pass yields raw output logits, which are unbounded scores that represent the model's internal confidence before applying any normalization. To control the sharpness of the resulting probability distribution, logits are divided by a temperature parameter, which is set to 0.7.

Lowering the temperature value (for example, from 1.0 to 0.7) makes the softmax output more peaked, assigning higher probabilities to the most likely classes. This can help the model make more decisive predictions, which is particularly useful in risk-sensitive contexts. In contrast, higher temperatures smooth the distribution, introducing greater uncertainty.

The softmax function is used to convert logits into class probabilities. The probability associated with the hateful class is extracted and clipped to remain within a numerically stable range. A binary decision is then made by comparing the hateful probability against a configurable decision threshold, which is set to 0.5. If the probability exceeds this threshold, the instance is labeled as "Hate", otherwise being labeled as "Not Hate".

In addition to the predicted label, the function returns the model's confidence score (the maximum class probability), the logarithm of the odds ratio, and the full class probability distribution.

VI. Using a LLM-Based Judgment

An embedded LLM reviewer, Mistral-7B-Instruct-v0.1, a state-of-the-art open-weight language model, is employed for post-evaluation of the classifier's outputs. The model is loaded in half-precision (FP16) on CUDA-enabled devices to optimize memory usage and inference speed.

To improve interpretability and provide expert-style validation, the method constructs a detailed prompt that includes the text input, the predicted class label, the model's confidence score, and the hate probability.

The prompt is carefully engineered to frame the language model as a domain expert in hate speech detection. It explicitly defines the role of the model and provides structured input, followed by clear, step-by-step instructions that direct it to assess classification, consider contextual and tonal subtleties, justify its reasoning, and conclude with a definitive verdict in the format "Final Verdict: HATE" or "Final Verdict: NOT HATE".

This prompt is passed to the Mistral-7B model, which is instructed to perform a critical analysis of the prediction, taking into account linguistic nuances and potential indicators of subtle or implicit hate speech. The model responds with a natural-language explanation culminating in a decisive judgment. This interpretive output improves transparency and provides human-readable justifications, thereby improving the trustworthiness and accountability of the overall system.

During inference, some parameters are applied to shape the generative behavior of the language model. The pad_token_id is set to the LLM tokenizer's eos_token_id to ensure correct padding behavior, avoiding generation artifacts and maintaining consistency with the model's end-of-sequence expectations. The temperature parameter is set to 0.7 to control the randomness of token sampling, encouraging focused yet variable output. The top_p value of 0.9 implements nucleus sampling, restricting token selection to the smallest subset whose cumulative probability exceeds 90%, thus balancing output diversity and coherence.

Furthermore, the entire pipeline is integrated into a PyQt-based graphical user interface (GUI), allowing users to input custom text for evaluation. The text can be classified by both the fine-tuned HateBERT model and the original pre-trained HateBERT model to easily observe differences between the two. The resulting prediction of the fine-tuned model can be passed to the LLM judge, which generates a final verdict and rationale, offering a comprehensive, multi-perspective analysis within an interactive application.

VII. Results

A. The Obtained Metric Values

When tested using the test set (which contains 33,508 examples), the model achieved an overall accuracy of 81%, a macro F1-score of 0.8062, a PR AUC of 0.8711, and a ROC AUC of 0.8967, indicating strong discriminative performance. The optimal classification threshold was found to be 0.3938, selected based on maximizing the F1 score.

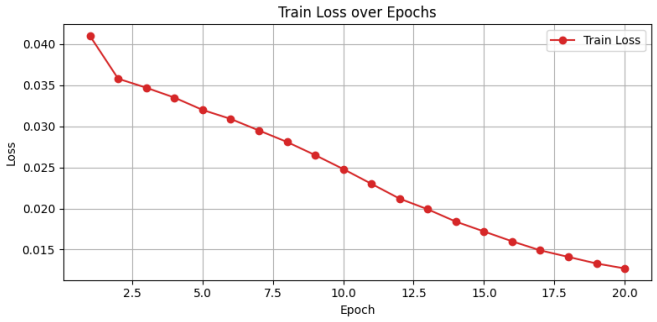

Fig. 1. Training loss over 20 epochs

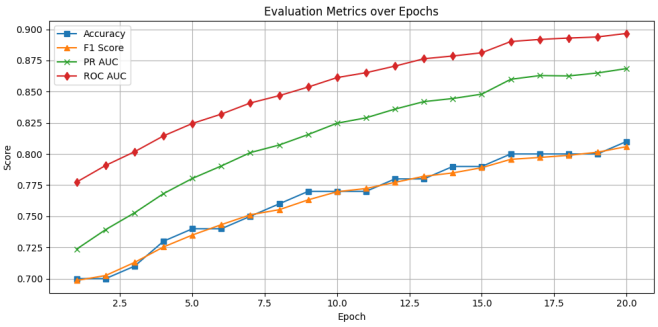

Fig. 2. Evaluation Metrics over Epochs

The following image is the confusion matrix for the original HateBERT model. It reveals significant performance disparities in classifying text as "Hate" or "Not Hate". The model shows strong performance in identifying "Not Hate" content, with 19047 true negatives and only 761 false positives. However, it performs poorly in detecting "Hate," correctly identifying only 171 out of 6418 actual hate instances, resulting in 6247 false negatives. This indicates a very low recall for the "Hate" class and a substantial bias toward predicting the majority class, "Not Hate".

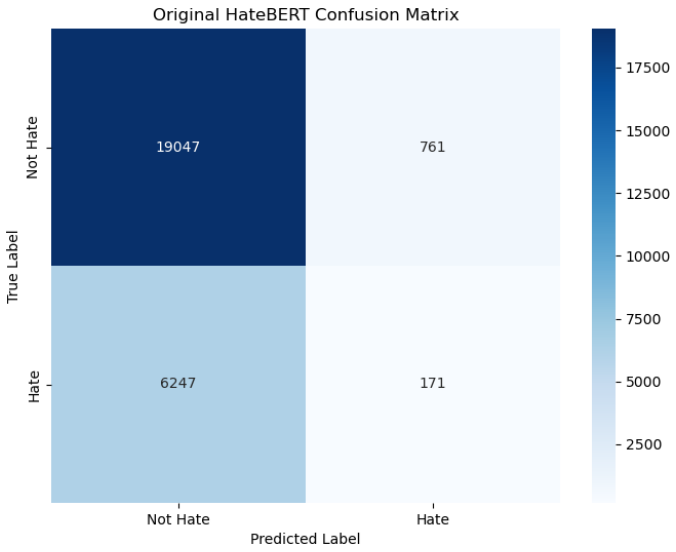

Fig. 3. Original HateBERT Confusion Matrix on testing dataset

The next image is the confusion matrix obtained by testing the fine-tuned HateBERT model. It shows that the model correctly predicted 15,069 instances of "Not Hate" and 12,001 instances of "Hate". It misclassified 3,768 "Not Hate" instances as "Hate" (false positives) and 2,670 "Hate" instances as "Not Hate" (false negatives).

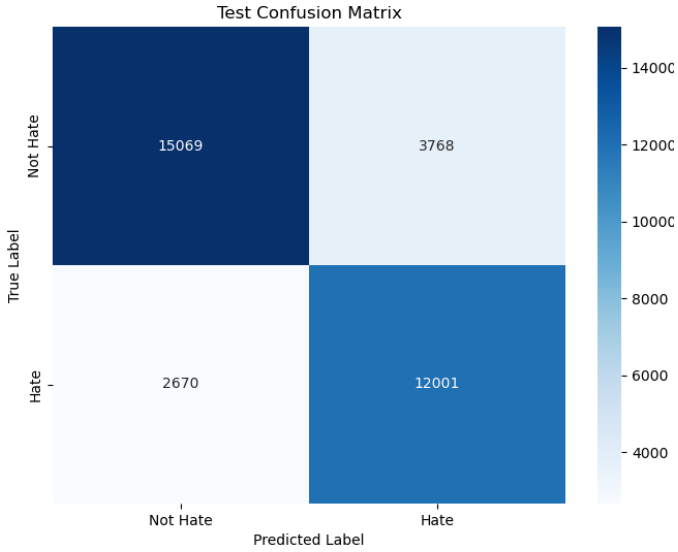

Fig. 4. Test Confusion Matrix for the Fine-Tuned HateBERT Model

B. Results Generated by The Models and LLM

The following sentence was used as an example: *"I genuinely despise everything you stand for - your presence, your opinions, your entire way of life disgusts me to my core."*

The fine-tuned version of HateBERT classified this sentence as hateful speech, assigning a hate probability of 86.82%, while the original HateBERT also labeled it as *Hate*, but with a lower hate probability of only 59.67%.

To verify these results and to have a last opinion about the correctness of the classification, the Mistral model was used as a judge and provided the following explanation: *"The prediction is accurate as the text expresses strong negative emotions towards the speaker's opinions and way of life, indicating a potential for hate speech. The context, tone, and subtle hate are all present in this text. The use of the word "despise"" and phrases like "your presence"" and "your entire way of life"" suggest a deep-seated hatred. Additionally, words like "disgusts"" and "core"" further reinforce this sentiment. Based on these factors, this text meets the criteria for hate speech."*

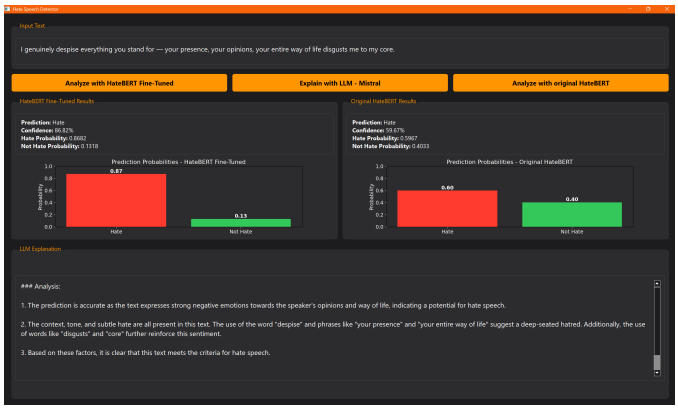

Fig. 5. PyQT User Interface

VIII. Conclusion

This project highlights the effectiveness of transformer-based models for hate speech detection by fine-tuning HateBERT on domain-specific data and comparing it to the original version. Fine-tuning significantly improved the model's ability to detect nuanced hateful language, yielding superior classification performance. These results confirm that combining domain-adaptive pretraining with targeted fine-tuning improves both the robustness and accuracy of hate speech detection models.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, Minneapolis, MN, USA, Jun. 2019, pp. 4171–4186.

[2] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[3] T. Caselli, V. Basile, J. Mitrović, L. Kallmeyer, and J. Bos, "HateBERT: Retraining BERT for abusive language detection in English," in *Proc. 5th Workshop on Online Abuse and Harms (WOAH)*, 2021.

[4] Z. Zhang, D. Robinson, and J. Tepper, "Detecting hate speech on social media: An analysis of model performance and robustness," in *Proc. 14th Int. AAAI Conf. Web Social Media*, 2018.

[5] C. Li, D. Jin, X. Ren, and Q. Yu, "Towards explainable hate speech detection with hybrid CNN-LSTM and attention mechanisms," *IEEE Trans. Comput. Social Syst.*, 2021.

[6] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.

[7] D. Sap et al., "Social Bias Inference Corpus (SBIC): A Dataset for Studying Social Biases in Language," in *Proc. AAAI Conf. Artificial Intelligence*, New York, NY, USA, Feb. 2020, pp. 13214–13225.

[8] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. EMNLP*, 2020.
[9] B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee, "HateXplain: A benchmark dataset for explainable hate speech detection," in *Proc. AAAI Conf. on Artificial Intelligence*, 2021.
[10] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Proc. 11th Int. AAAI Conf. on Web and Social Media (ICWSM)*, Montreal, Canada, 2017, pp. 512–515.
[11] M. Sap, S. Gabriel, L. Qin, D. Jurafsky, N. A. Smith, and Y. Choi, "Social bias frames: Reasoning about social and power implications of language," in *Proc. 58th Annu. Meeting of the Assoc. for Computational Linguistics (ACL)*, Online, 2020, pp. 5477–5490.