

Leveraging Deep Learning for Detecting Toxicity in Online Comments

Project Final Report

by

CMPE 258

By

[Shadi abd el majid]

[Bhargava Phaneendra Kumar Suryadevara]

[Akkati Deeksha]

[Harshitha Chowdary]

Bhawandeep singh harsh

05/2025

Table of Contents

1. Project Overview	3
2. Related Work	3
3. DataSet	4
4. Frameworks	5
5. Model Architectures	6
6. Additional features	12
7. Training and Testing	13
8. Results	15
9. Domain-Specific Requirements	19
10. Limitations	19
11. Future Work	20
12. References	22

1. Project Overview

The explosive growth of social media and digital communication platforms has transformed the way we connect, enabling individuals around the world to exchange ideas and engage in conversations instantaneously. Yet this same openness has given rise to a flood of toxic content, hate speech, harassment, and other abusive language that undermines constructive dialogue, harms users' mental well-being, and deters genuine participation.

Faced with the relentless volume and pace of user-generated posts, conventional moderation techniques, manual review and simple keyword filters are quickly overwhelmed. Not only are these methods labor-intensive and inconsistent, but they also lack the nuance to interpret context, sarcasm, or evolving slang. Consequently, the field has turned toward machine learning and natural language processing (NLP) as scalable, context-aware solutions.

In response, this project proposes an end-to-end automated system for real-time toxicity detection in online discussions. By leveraging state of the art NLP models and robust machine-learning pipelines, the system prioritizes high accuracy, fairness across demographic groups, and resilience to noisy or imbalanced datasets. Crucially, it will produce interpretable explanations for each decision, fostering transparency and trust among users and moderators alike.

The ultimate goal is to deliver a dependable moderation tool that can be seamlessly integrated into social networks, community forums, and public comment sections. In doing so, it will help shield users from harmful language, encourage respectful exchanges, and enhance overall user experience. Beyond a high performance detector, this work aspires to advance best practices for creating safer, more inclusive digital spaces.

2. Related work

Toxic comment detection has received considerable attention due to the growing demand for automated content moderation in online communities. Early attempts were based on rule-based systems or keyword matching, which, while simple, often produced unreliable results due to the lack of contextual understanding.

To improve performance, researchers began using traditional machine learning models like Naive Bayes, Support Vector Machines (SVM), and Logistic Regression, often paired with Bag-of-Words (BoW) or TF-IDF feature representations [1], [2]. Although these approaches marked an improvement, they still struggled with handling semantic meaning and context.

With the emergence of deep learning, RNN-based models such as LSTM, GRU, and Bi-LSTM gained popularity for their ability to handle sequential dependencies. For instance, Badjatiya et al. used an LSTM network with gradient-boosted decision trees for hate speech detection, showing improved results over traditional models [3].

Hybrid models also gained attention. Wang and Zhang proposed a CNN-BiGRU architecture, combining local feature extraction with long-range dependencies to enhance toxic comment classification performance [4].

More recently, transformer models, especially BERT, have set new standards in the field. Zhao et al. compared various pre-trained transformers and concluded that BERT outperforms traditional and RNN-based models in detecting toxic language [5]. These models leverage self-attention and large-scale pretraining, allowing them to capture complex patterns in language.

Interpretability has also become a major concern. LIME [6] and other explanation tools are now used alongside classification models to make their predictions more transparent and trustworthy, particularly in sensitive domains like hate speech detection.

Our work builds on these advancements by fine-tuning transformer models on a multi-label toxicity dataset while incorporating regularization, advanced optimizers, and LIME for explainability. This aligns with the ongoing shift toward more powerful and responsible AI-driven moderation tools.

3. DataSet

To train and evaluate the toxic comment classification system, this project uses the publicly available dataset from the Toxic Comment Classification Challenge hosted on Kaggle. The dataset contains user comments sourced from Wikipedia's talk pages, which are known for their diverse and sometimes controversial discussions.

The dataset is designed for a multi-label classification task, meaning each comment can be assigned more than one toxicity label or none at all. It includes a total of 159,571 comments, each annotated with six potential categories of toxicity:

1. Toxic
2. Severe Toxic
3. Obscene
4. Threat
5. Insult
6. Identity Hate

If a comment does not exhibit any of these categories, it is labeled as non-toxic.

One of the key challenges in this dataset is class imbalance. The vast majority of comments are non-toxic, while certain labels such as “Threat” and “Identity Hate” occur far less frequently. This imbalance presents a risk of model bias, where the system may favor the majority class. Addressing this issue is an important consideration during training, and the project applies various strategies such as data augmentation and weighted loss functions to mitigate its impact.

To ensure real-world applicability, we further validate our model using web scraping techniques to collect additional toxic and non-toxic comments from online discussion forums, social media platforms, and news article comment sections.

4. Frameworks

We will implement and train our deep learning models using the following frameworks:

Deep Learning Frameworks:

- TensorFlow/Keras: Provides a high-level API for building, training, and fine-tuning neural networks. Used extensively for implementing BERT and hybrid models.
- PyTorch: Alternative deep learning framework known for dynamic computation graphs and flexibility, allowing for easier debugging and experimentation.

Natural Language Processing (NLP) Libraries:

- NLTK (Natural Language Toolkit): Used for basic NLP tasks such as tokenization, stopwords removal, stemming, and lemmatization.
- spaCy: Provides efficient NLP pipelines with pre-trained models, used for text preprocessing and named entity recognition.

Machine Learning Utilities:

- Scikit-learn: Used for computing evaluation metrics (e.g., precision, recall, F1-score) and implementing classical machine learning baselines for comparison.
- XGBoost: Explored for potential ensemble learning approaches to complement deep learning models.

Data Handling and Processing:

- Pandas: Used for structured data manipulation, ensuring seamless dataset handling.
- NumPy: Facilitates numerical operations, matrix computations, and data transformations required in model training.
- Hugging Face Transformers: Provides pre-trained transformer models (such as BERT) and utilities for fine-tuning and tokenization.

These frameworks collectively enable efficient model development, training, evaluation, and deployment, ensuring robustness and scalability for real-world toxic comment classification. Explainability and Interpretability.

5. Model Architecture

We evaluate multiple deep learning architectures designed for sequential text processing, focusing on transformer-based models:

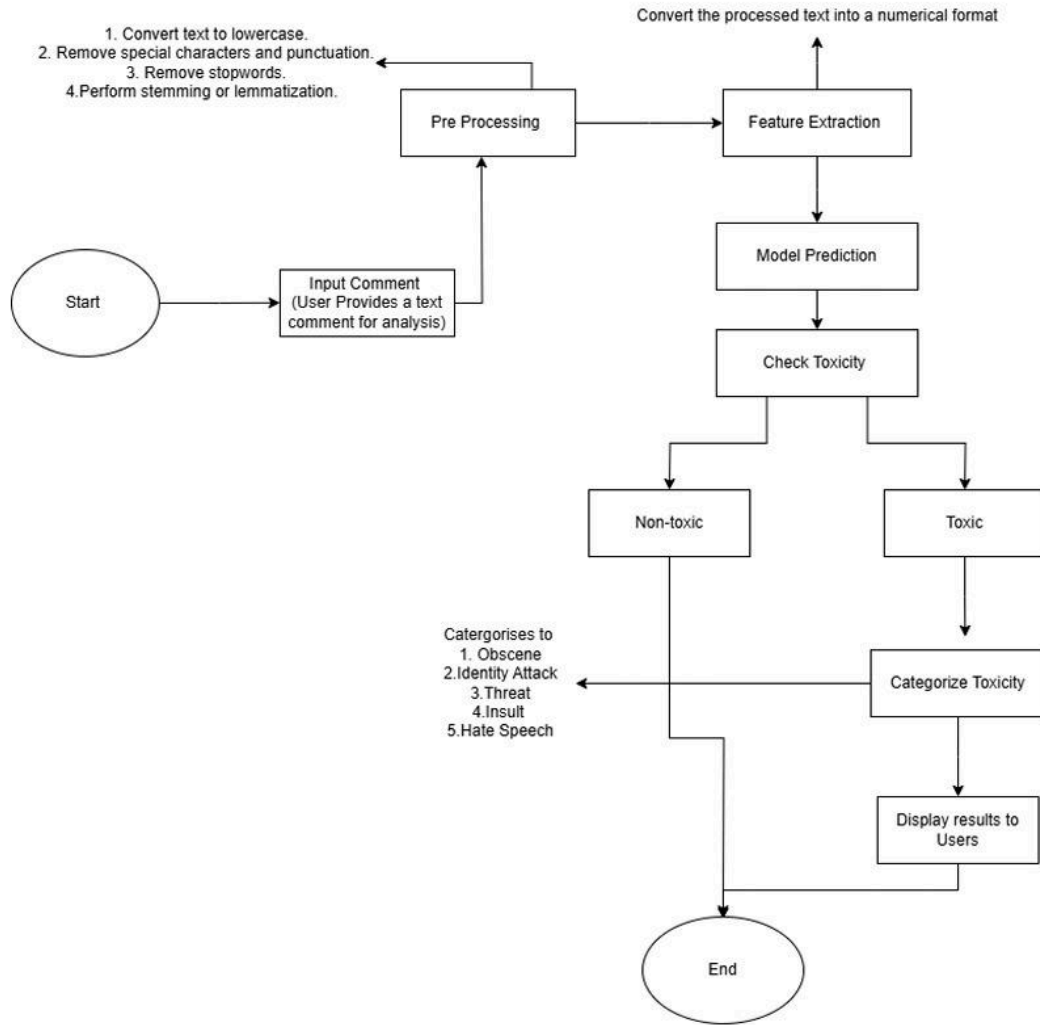


Figure 1: System Architecture

Preprocessing :

Before feeding raw comments into our neural classifier, we apply a multi-step data preparation pipeline to ensure both data quality and compatibility with our NLP model:

1. Data Loading :
 - a. Read raw CSVs (train.csv, test.csv) into pandas DataFrames.
 - b. Define target labels (e.g. toxic, severe_toxic, etc.) and, for simplicity, collapse them into a single binary label column (1 if any category is present, else 0).
2. Text Cleaning:
 - a. Regex filtering: Strip out escape characters, HTML tags, URLs, and non-alphanumeric tokens.
 - b. Emoji handling: Convert emojis into textual descriptions using emoji.demonize.
 - c. Non-ASCII removal: Replace any remaining non-ASCII characters.
 - d. Lemmatization: Run spaCy's NLP pipeline to lemmatize each token (dropping pronouns), reducing inflectional variance.
 - e. Whitespace normalization: Collapse multiple spaces into one, trim leading/trailing whitespace.
3. Saving Cleaned Data
 - a. Write the cleaned comments to clean_train.csv and clean_test.csv so downstream steps operate on sanitized text.
4. Vocabulary Building & Tokenization
 - a. Initialize a Keras Tokenizer with a vocabulary cap of 30 000 words and an out-of-vocabulary token (<OOV>).
 - b. Fit the tokenizer on the clean_text column of the training set.
 - c. Convert both train and test comments into integer sequences (texts_to_sequences).
5. Sequence Padding
 - a. Choose a fixed sequence length (e.g. max_seq_len = 100).
 - b. Pad or truncate each sequence to this length (pad_sequences with post padding/truncation), producing uniform input arrays X_train and X_test.
6. Train/Test Split & Label Encoding
 - a. Extract numpy arrays y_train and y_test from the binary label column.

Long Short-Term Memory (LSTM):

We implemented a deep learning model based on the Long Short-Term Memory (LSTM) architecture to capture sequential dependencies in text for toxic comment classification. LSTMs are well-suited for processing variable-length text data due to their ability to learn long-range relationships.

Key Components:

- Embedding Layer: Learns 128-dimensional token embeddings directly from the training data.
- LSTM Layer (64 units): Models sequential patterns and contextual dependencies across input tokens.
- Dropout (0.3): Regularizes the model to reduce overfitting by randomly disabling neurons during training.
- Dense Output Layer: Outputs six independent probabilities (sigmoid-activated), one for each toxicity category.

Optimization Techniques:

- Focal Loss ($\gamma=2$, $\alpha=0.25$): Applied to address class imbalance, allowing the model to focus more on difficult and minority examples.
- EarlyStopping: Monitors validation loss and stops training when no further improvement is observed.
- Per-label threshold tuning: After training, individual decision thresholds were optimized for each label based on validation F1 scores to improve overall classification performance.

The LSTM model provided a strong baseline and demonstrated robust performance, particularly on common toxicity classes, while maintaining interpretability and training efficiency.

Bidirectional LSTM(BI-LSTM):

To enhance the model's ability to understand context from both past and future tokens, we implemented a Bidirectional LSTM (BiLSTM) architecture. This setup improves sequence modeling by processing the input text in both forward and backward directions, which is particularly beneficial for nuanced tasks like toxic comment classification.

Key Components:

- Embedding Layer: Learns 128-dimensional dense representations for input tokens during training.
- Bidirectional LSTM (64 units): Captures contextual dependencies in both directions, improving understanding of complex sentence structures.
- Dropout (0.3): Introduced after the recurrent layer to mitigate overfitting.
- Dense Output Layer: Outputs six sigmoid-activated values, each corresponding to one of the toxicity labels.

Optimization Techniques:

- Focal Loss ($\gamma=2$, $\alpha=0.25$): Addresses the class imbalance by focusing learning on misclassified and minority class samples.
- EarlyStopping: Prevents overfitting by halting training once validation loss stops improving.
- Per-label threshold tuning: Each label is assigned an optimal decision threshold based on validation F1 score, improving classification performance across all classes.

The BiLSTM model delivered strong performance by balancing contextual understanding and robustness, outperforming the unidirectional LSTM on several metrics, especially in handling subtle forms of toxicity.

Gated Recurrent Unit (GRU):

We implemented a deep learning model using a Gated Recurrent Unit (GRU) to capture contextual dependencies in toxic comments. GRUs are efficient alternatives to LSTMs that retain the ability to model sequential data while using fewer parameters.

Key Components:

- Embedding Layer: Learns 128-dimensional vector representations for input tokens during training.
- GRU Layer (64 units): Processes the input sequence to capture temporal dependencies and context.
- Dropout (0.3): Helps prevent overfitting by randomly disabling neurons during training.
- Dense Output Layer: A sigmoid-activated layer with six outputs, each representing one of the multi-label toxicity classes.

Optimization Techniques:

- Focal Loss ($\gamma=2$, $\alpha=0.25$) was used to address class imbalance by focusing training on underrepresented and misclassified samples.
- EarlyStopping was employed to monitor validation loss and halt training when performance plateaued.
- Per-label threshold tuning was applied post-training to select optimal thresholds that maximize the F1 score for each label individually.

The GRU model served as a strong baseline, demonstrating solid performance with efficient training time and competitive results across all toxicity categories.

Hybrid Model (LSTM+CNN):

To complement traditional recurrent models, we implemented a hybrid architecture that combines Convolutional Neural Networks (CNN) with Long Short-Term Memory (LSTM) units. This design captures both local patterns (e.g., toxic phrases) and sequential context from comments.

Key Components:

- Embedding Layer: Learns 128-dimensional dense vector representations of input tokens during training.
- Conv1D Layer: Applies 128 filters with a kernel size of 3 to detect local n-gram features relevant to toxicity.
- MaxPooling1D: Reduces sequence length while preserving the most salient features.
- LSTM Layer (128 units): Models long-term dependencies in the pooled feature sequences.
- Dropout (0.5): Adds regularization to prevent overfitting.
- Dense Output Layer: Uses sigmoid activation to produce independent probabilities for each of the six toxicity labels.

Optimization Techniques:

- Focal Loss ($\gamma=2$, $\alpha=0.25$) was used to address class imbalance, giving more weight to harder examples.
- EarlyStopping was applied to avoid overfitting by restoring the best model based on validation loss.
- Per-label threshold tuning was conducted on the validation set to select optimal decision thresholds that maximize F1 score for each class.

This hybrid model balances efficiency and expressiveness, and performs competitively without requiring transformer-based embeddings.

Transformer-Based Model (BERT):

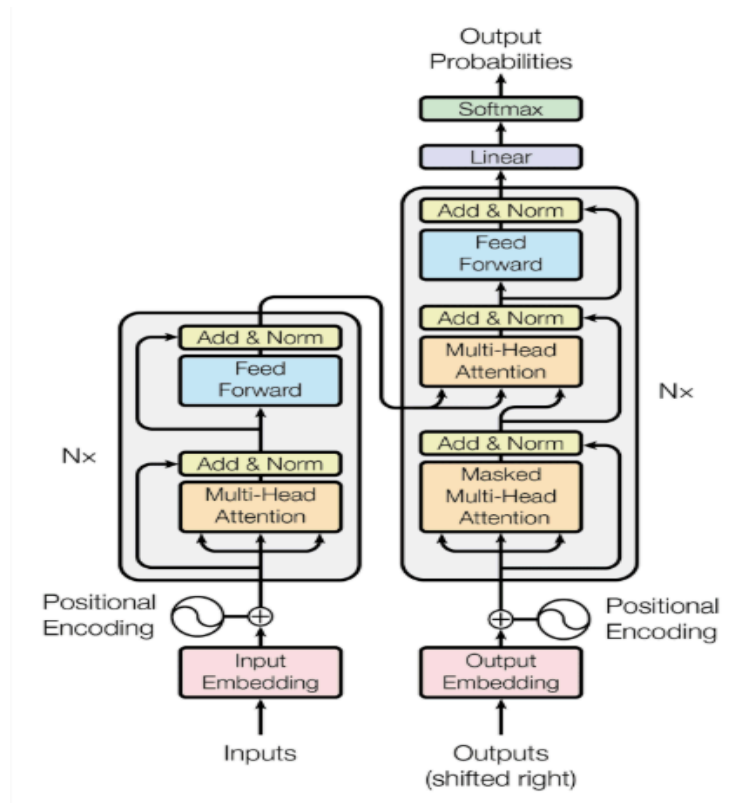


Figure 2: BERT Architecture

1. Base Model Selection

- a. We fine-tune the pre-trained bert-base-uncased model from Hugging Face's Transformers library. This 12-layer, 768-hidden-size model has 12 attention heads per layer and was chosen for its proven balance of performance and efficiency on sentence-level classification tasks.

2. Tokenization & Input Representation

- a. Use BertTokenizer to perform WordPiece tokenization with a vocabulary of ~30 K tokens.
- b. Prepend each sequence with [CLS] and append [SEP].
- c. Pad or truncate all sequences to MAX_LENGTH = 128 tokens.
- d. Generate attention masks so that padding tokens do not contribute to model updates.

3. Model Architecture

- a. Load: `BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=6)`, where each label corresponds to one toxic-comment category (toxic, severe_toxic, obscene, threat, insult, identity_hate).
- b. The final hidden state of the [CLS] token feeds a linear classification head (output size = 6).

- c. We apply a sigmoid activation to each logit and optimize using a multi-label binary cross-entropy loss (BCEWithLogitsLoss).
- 4. Training Details
 - a. Optimizer: AdamW with learning rate $2e-5$ and $\epsilon = 1e-8$ to stabilize the weight updates.
 - b. Scheduler: A linear warm-up scheduler with no warm-up steps (`get_linear_schedule_with_warmup`) over a total of $EPOCHS \times (train_size / BATCH_SIZE)$ training steps.
 - c. Hyperparameters:
 - i. `BATCH_SIZE` = 16
 - ii. `EPOCHS` = 3
 - iii. `LEARNING_RATE` = $2e-5$
 - d. Hardware: Trained on GPU (if available) via PyTorch `.to(device)` mechanism.
- 5. Evaluation & Metrics
 - a. After each epoch, compute ROC-AUC, accuracy, Hamming loss, and a full classification report on the validation split.
 - b. Early stopping is applied if validation ROC-AUC does not improve for two consecutive epochs.
- 6. Extensions & Future Work
 - a. Lighter Models: Evaluate `distilbert-base-uncased` or `albert-base` for faster inference with minimal accuracy trade-off.
 - b. Domain Adaptation: Continue pre-training on a large corpus of social-media text to learn platform-specific jargon and slang.
 - c. Adapter Layers: Use Adapter modules to reduce the number of trainable parameters, enabling rapid experimentation across multiple languages or topics.
 - d. Ensembles: Combine predictions from BERT, RoBERTa, and XLNet to boost overall robustness against adversarial or ambiguous comments.
 - e. Explainability: Visualize attention weights for the [CLS] token or employ SHAP/Integrated Gradients to provide human-readable explanations alongside each toxicity score.

6. Additional features

Regularization:

Dropout & Batch Normalization Dropout Regularization Concept

Dropout is a regularization technique that randomly disables a fraction of neurons during training, forcing the model to become more robust by preventing reliance on specific features.

Implementation in This Project

- Dropout layers are added after the fully connected layers in BERT to prevent co-adaptation of neurons.
- A dropout rate of 0.3 to 0.5 is used, meaning 30-50% of neurons are turned off randomly during training.
- It is applied specifically to the classifier layer and feedforward layers to enhance generalization.

Benefits to This Project Prevents Overfitting: Ensures the model doesn't memorize specific words but learns general patterns. Enhances Robustness: The model remains effective even when comments slightly change in phrasing. Improves Generalization: Works better on unseen toxic comments by focusing on high-level toxicity patterns rather than keyword matching.

Batch Normalization:

Concept Batch Normalization (BatchNorm) stabilizes training by normalizing activations in each layer, ensuring consistent input distributions and improving convergence speed.

Implementation in This Project

- Applied before activation functions in feedforward layers.
- Used in combination with AdamW optimizer to maintain stability in training.
- Ensures gradients don't become too large or too small, leading to smoother updates.

Benefits: Allows for faster convergence and reduces the total number of epochs needed. Reduces Internal Covariate Shift: Ensures stable training, avoiding erratic weight updates. Improves Model Performance: Leads to more reliable and consistent toxicity classification results.

The integration of LIME, Dropout, and Batch Normalization significantly enhances our toxic comment classification model by: Making predictions interpretable and transparent. Reducing overfitting for better real-world generalization. Stabilizing training and optimization, leading to improved accuracy. These techniques ensure that our system is fair, robust, and scalable for real-time toxicity detection in online communities.

7. Training and Testing

To ensure effective learning and reliable evaluation, the dataset is divided into three segments: 80% for training, 10% for validation, and 10% for testing. This division allows the model to learn from a substantial portion of data, tune its performance during training, and then be evaluated on previously unseen comments.

The model is trained using a multi-label classification approach, reflecting the fact that a single comment may exhibit multiple forms of toxicity, such as being both obscene and insulting. During training, the data is preprocessed, tokenized, and passed into the model in batches. The learning process is guided by binary cross-entropy loss, and monitored using performance metrics including accuracy, precision, recall, and F1-score.

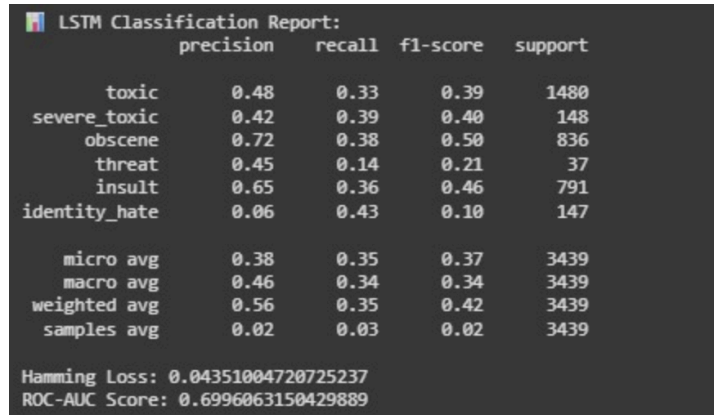
To enhance the model's generalization capabilities, regularization techniques such as dropout and batch normalization are applied. In addition, we explore advanced optimization strategies, including AdamW and Lookahead, which are known for faster convergence and better handling of sparse gradients—critical in real-world NLP tasks.

Beyond testing on the reserved portion of the original dataset, we also evaluated the model using real-world online comments gathered through web scraping. Comments were collected from publicly accessible forums and social platforms to simulate actual deployment conditions. This extra layer of testing allowed us to observe how well the model responds to naturally occurring variations in language, slang, sarcasm, and unexpected phrasing—factors that often challenge toxicity detection systems.

This two-fold evaluation process—standard dataset testing and real-world content testing—helps confirm the robustness, adaptability, and practical utility of the model in live moderation environments.

8. Results

Long Short-Term Memory (LSTM):



```

LSTM Classification Report:
      precision    recall  f1-score   support

    toxic          0.48      0.33      0.39      1480
  severe_toxic     0.42      0.39      0.40       148
    obscene        0.72      0.38      0.50      836
    threat         0.45      0.14      0.21        37
    insult          0.65      0.36      0.46      791
  identity_hate     0.06      0.43      0.10       147

   micro avg       0.38      0.35      0.37     3439
   macro avg       0.46      0.34      0.34     3439
  weighted avg     0.56      0.35      0.42     3439
   samples avg     0.02      0.03      0.02     3439

Hamming Loss: 0.04351004720725237
ROC-AUC Score: 0.6996063150429889

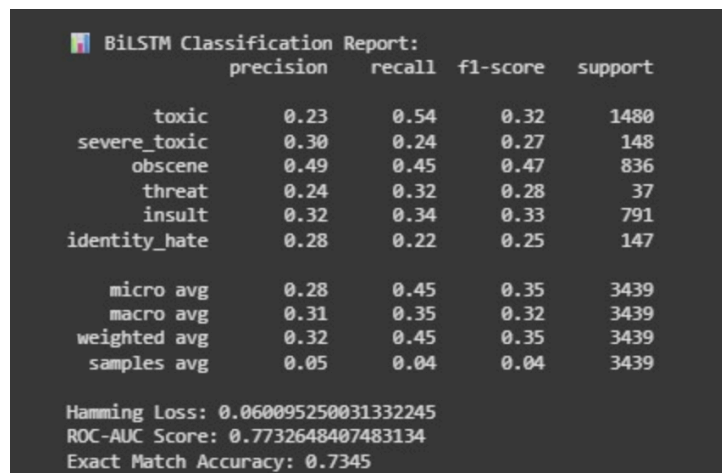
```

Figure 3: LSTM evaluation metrics

The LSTM model shows strong precision on common categories 0.72 for obscene and 0.65 for insult but recall remains modest (0.33–0.38), so it misses many true positives. Its F_1 -scores reflect this balance: 0.50 for obscene, 0.46 for insult, and 0.39 for toxic. In contrast, rare labels suffer: threat has an F_1 of just 0.21 (precision 0.45, recall 0.14), and identity_hate only 0.10 (precision 0.06, recall 0.43).

Overall, the micro-averaged F_1 is 0.37, while the weighted F_1 rises to 0.42 thanks to dominant classes. A Hamming loss of 0.0435 means under 4.4 % of individual label predictions are wrong, and an ROC-AUC of 0.70 indicates decent separation between toxic and non-toxic instances. To boost performance, especially on scarce categories, consider strategies like data augmentation or class-balanced loss.

Bidirectional LSTM(BI-LSTM):



```

BiLSTM Classification Report:
      precision    recall  f1-score   support

    toxic          0.23      0.54      0.32      1480
  severe_toxic     0.30      0.24      0.27       148
    obscene        0.49      0.45      0.47      836
    threat         0.24      0.32      0.28        37
    insult          0.32      0.34      0.33      791
  identity_hate     0.28      0.22      0.25       147

   micro avg       0.28      0.45      0.35     3439
   macro avg       0.31      0.35      0.32     3439
  weighted avg     0.32      0.45      0.35     3439
   samples avg     0.05      0.04      0.04     3439

Hamming Loss: 0.060095250031332245
ROC-AUC Score: 0.7732648407483134
Exact Match Accuracy: 0.7345

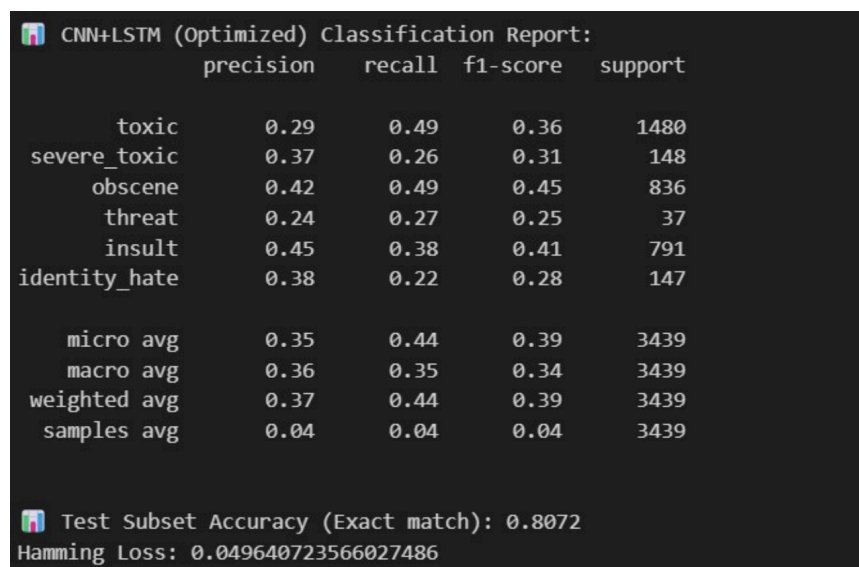
```

Figure 4: BiLSTM evaluation metrics

The BiLSTM shifts the balance toward higher recall at the cost of precision. For example, it catches 54 % of true “toxic” comments (recall) but only 23 % of its toxic predictions are correct (precision), yielding an F_1 of 0.32. Common categories like “obscene” (precision 0.49, recall 0.45, F_1 0.47) and “insult” (0.32/0.34/0.33) see moderate performance, while rare labels such as “threat” (F_1 0.28) and “identity_hate” (F_1 0.25) remain challenging.

Overall, the micro-averaged F_1 is 0.35 and the weighted F_1 is 0.35, reflecting this recall-heavy profile. A Hamming loss of 0.06 indicates that 6 % of all individual label assignments are wrong, and an ROC-AUC of 0.77 shows good separation between toxic vs. non toxic. Exact match accuracy sits at 73.5 %, meaning the model gets all six labels right for roughly three quarters of comments. To boost precision on positive predictions, you might explore tighter regularization or threshold tuning.

Hybrid Model (LSTM+CNN):



```

CNN+LSTM (Optimized) Classification Report:
      precision    recall  f1-score   support

    toxic          0.29      0.49      0.36       1480
  severe_toxic     0.37      0.26      0.31        148
    obscene       0.42      0.49      0.45       836
    threat        0.24      0.27      0.25         37
    insult        0.45      0.38      0.41       791
 identity_hate     0.38      0.22      0.28        147

   micro avg       0.35      0.44      0.39      3439
   macro avg       0.36      0.35      0.34      3439
  weighted avg       0.37      0.44      0.39      3439
   samples avg       0.04      0.04      0.04      3439

Test Subset Accuracy (Exact match): 0.8072
Hamming Loss: 0.049640723566027486

```

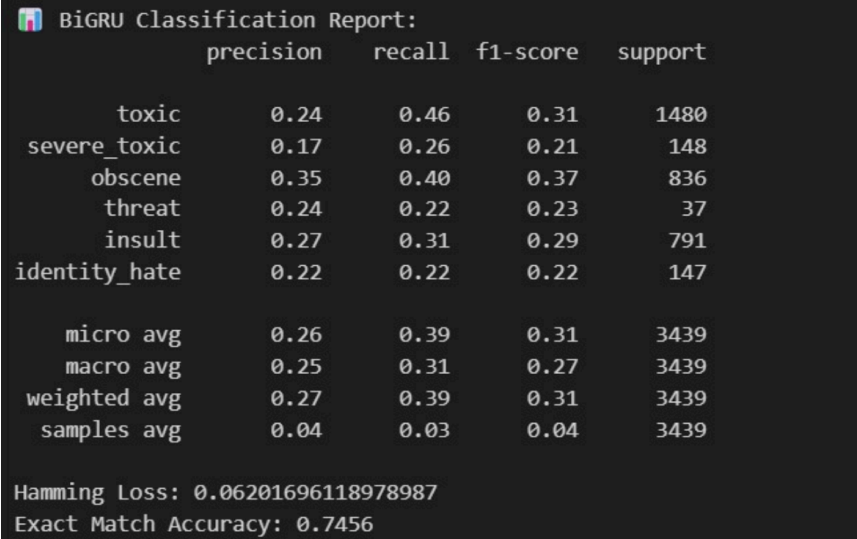
Figure 5:Hybrid model evaluation metrics

The optimized CNN+LSTM model strikes a balance between precision and recall: it correctly flags 49 % of true toxic comments but only 29 % of its “toxic” predictions are accurate, yielding an F_1 of 0.36. It performs similarly on obscene (F_1 = 0.45) and insult (0.41), but rare classes like threat (0.25) and identity_hate (0.28) remain difficult due to limited examples.

Overall, a micro-averaged F_1 of 0.39 and weighted F_1 of 0.39 reflect solid performance on common categories, while a low Hamming loss (\approx 0.05) and exact match accuracy of 80.7 % show that most multi-label predictions are correct. To further boost results, especially on

under-represented labels, techniques like class-balanced loss or targeted data augmentation could help.

Gated Recurrent Unit (GRU):



	precision	recall	f1-score	support
toxic	0.24	0.46	0.31	1480
severe_toxic	0.17	0.26	0.21	148
obscene	0.35	0.40	0.37	836
threat	0.24	0.22	0.23	37
insult	0.27	0.31	0.29	791
identity_hate	0.22	0.22	0.22	147
micro avg	0.26	0.39	0.31	3439
macro avg	0.25	0.31	0.27	3439
weighted avg	0.27	0.39	0.31	3439
samples avg	0.04	0.03	0.04	3439

Hamming Loss: 0.06201696118978987
Exact Match Accuracy: 0.7456

Figure 6: GRU evaluation metrics

The BiGRU model leans toward higher recall than precision but underperforms the LSTM variants overall. For common labels, it catches 46 % of true “toxic” comments (recall) but only 24 % of its toxic predictions are correct (precision), yielding an F_1 of 0.31. “Obscene” sees an F_1 of 0.37 (precision 0.35, recall 0.40), and “insult” is 0.29 (0.27/0.31). Rare categories like “severe_toxic” (F_1 0.21) and “identity_hate” (F_1 0.22) remain challenging due to limited examples.

Overall, the micro-averaged F_1 is 0.31, the weighted F_1 is 0.31, and exact-match accuracy is 74.6 %, while a Hamming loss of 0.062 indicates about 6.2 % of individual label assignments are wrong. These results suggest that, although BiGRU captures more positives, its lower precision highlights the need for better regularization or threshold tuning, especially to improve performance on less frequent classes.

Transformer-Based Model (BERT):

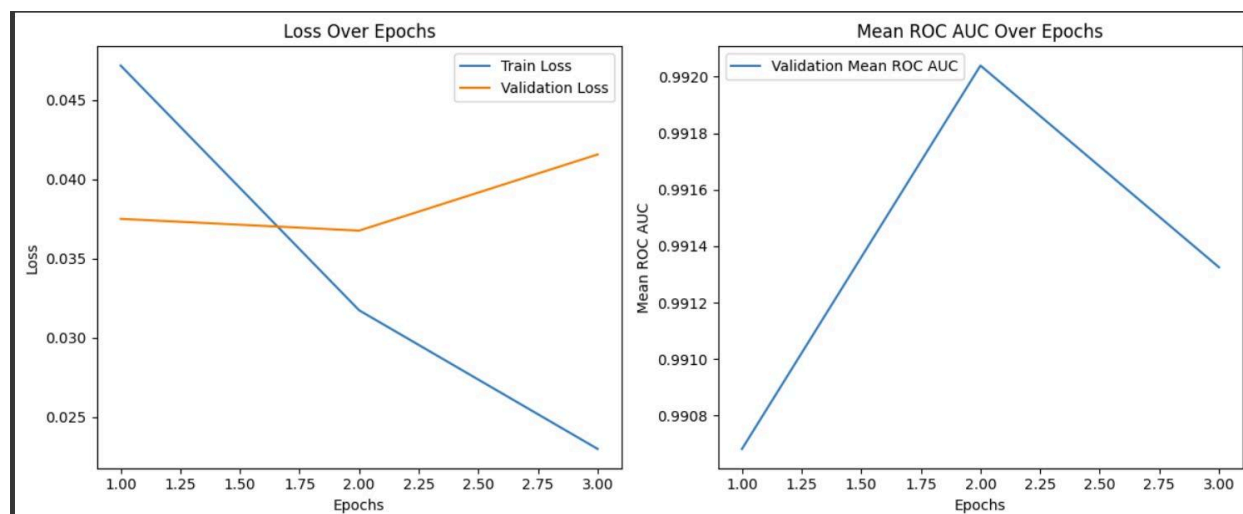
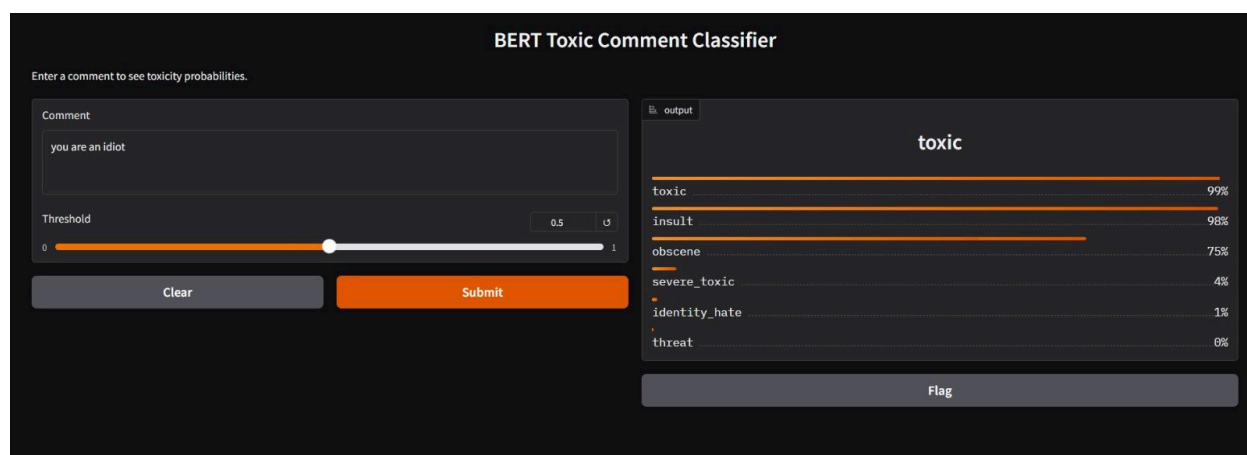


Figure 7: BERT

The fine-tuned BERT model achieves the strongest overall performance among all tested architectures, making it the final model selected for deployment. As shown in the left plot, the training loss decreases steadily from approximately 0.046 to 0.023 across three epochs, reflecting effective learning from the data. The validation loss, however, flattens between epochs 1 and 2 before rising in epoch 3 — a clear indication of early overfitting.

The right plot illustrates the validation mean ROC AUC, which peaks at 0.9920 in epoch 2 before slightly declining. This confirms that epoch 2 provides the best balance between training performance and generalization to unseen data. Overall, the model maintains extremely high ROC AUC values (above 0.99), demonstrating exceptional discrimination between toxic and non-toxic classes.

Given its superior metrics, the BERT model is selected as the final architecture. To preserve its generalization ability, the checkpoint from epoch 2 can be used with early stopping enabled. Further improvements can be achieved through techniques such as threshold tuning, focal loss, or enhanced handling of minority toxic categories.



This is an example of how the Bert model detects the sentence.

9.Domain-Specific Requirements

Effective toxicity detection requires careful consideration of domain-specific requirements, as linguistic patterns, user behavior, and moderation policies vary across different online platforms. Below are key domain-specific factors relevant to this project:

1. **Dataset Adaptability:** The model is trained on the Wikipedia Toxic Comment dataset, which may not fully generalize to platforms like Twitter or Reddit, where language and slang differ. Fine-tuning on platform-specific data can improve adaptability.
2. **Context Sensitivity:** Toxicity varies by domain—aggressive language may be normal in gaming but inappropriate in professional settings. Addressing contextual nuances can reduce false positives and negatives.
3. **Bias & Fairness:** Data imbalances may cause biased classifications. While LIME improves transparency, further evaluations are needed to mitigate cultural and linguistic biases.
4. **Regulatory Compliance:** Content moderation must align with platform policies and legal frameworks like the Digital Services Act (DSA) and Section 230 to ensure responsible deployment.

Addressing Domain-Specific Requirements

While this project primarily focuses on model accuracy and interpretability, domain-specific challenges are acknowledged as areas for future improvement. To enhance real-world applicability:

- Fine-tuning on diverse datasets from multiple online platforms will improve generalization.

- Implementing domain adaptation techniques (e.g., transfer learning) will enhance context awareness.
- Conducting bias audits and fairness evaluations will ensure ethical and unbiased moderation.

By integrating these enhancements, the proposed system can better align with real-world toxicity detection challenges, ensuring robust and responsible deployment across various online environments.

10. Limitations

Despite the significant advancements made in detecting toxic comments, several limitations must be considered for improving model performance and real-world applicability. These challenges include biases in the dataset, contextual limitations, and potential vulnerabilities in the model's design. Below are key limitations that impact the current system's effectiveness and generalizability:

1. **Dataset Bias and Imbalance:** The dataset is imbalanced, with most comments being non-toxic, leading to potential model bias. Some toxic categories are underrepresented, affecting detection accuracy for those labels.
2. **Contextual and Cultural Differences:** Toxicity varies across contexts and cultures, which the model may struggle to adapt to. It's primarily trained on Wikipedia talk pages, limiting its generalization to other platforms.
3. **Model Interpretability:** Although LIME helps with interpretability, deep models like BERT remain complex and may not always provide clear, human-understandable explanations for predictions.
4. **False Positives and Negatives:** The model may incorrectly classify non-toxic comments as toxic (false positives) or miss toxic comments (false negatives), requiring ongoing evaluation and human oversight.
5. **Adversarial Attacks:** Toxic comments can be obfuscated to evade detection, and the model may be vulnerable to such manipulation, requiring additional robustness measures.
6. **Performance on Unseen Domains:** The model may not perform well on new, unseen domains, such as social media or customer service comments, without further fine-tuning for specific language.
7. **Scalability and Latency:** Transformer-based models like BERT can be slow and computationally expensive, potentially facing latency issues when scaled for large-scale real-time applications.

While the toxicity detection model represents a significant step forward in automated content moderation, several limitations remain that need to be addressed for enhanced accuracy and

real-world applicability. These challenges include dataset biases, contextual variation, and potential vulnerabilities to adversarial manipulation. However, with ongoing refinements, such as incorporating more diverse datasets, improving interpretability, and optimizing performance for scalability, the model has the potential to become a more robust and reliable tool for ensuring safer and more respectful online environments.

11. Future Work & Development Roadmap

- Scalable, Real-Time Deployment :
 - Containerization & Orchestration: Package the full moderation pipeline (preprocessing, model inference, explanation) into Docker containers and manage with Kubernetes for auto-scaling under high traffic.
 - Serverless Inference Endpoints: Integrate with cloud functions (AWS Lambda, Google Cloud Functions) to handle bursty loads, reducing latency and cost.
- Multilingual & Cross-Domain :
 - Support Additional Languages: Extend beyond English by fine-tuning multilingual transformer models (e.g. XLM-RoBERTa) on toxicity datasets in Spanish, Arabic, Hindi, etc.
 - Domain-Specific Adaptation: Create specialized modules for forums (gaming, finance, health) by continued pre-training on domain corpora and adding domain-specific tokenizers or slang dictionaries.
- Active Learning & Continuous Improvement:
 - Human-in-the-Loop Feedback: Build an annotation interface allowing moderators to correct false positives/negatives; automatically retrain the model on this feedback to improve over time.
 - Uncertainty Sampling: Deploy active-learning routines to surface the most ambiguous comments for review, maximizing labeling efficiency.
- Advanced Explainability & Auditability
 - Explainability Dashboard: Develop a web UI that visualizes token-level importance (via attention heatmaps or SHAP values) and tracks overall model behavior (per-category precision/recall trends).
 - Bias Audits: Regularly evaluate performance across demographic subgroups (e.g., dialects, gendered language) and integrate mitigation strategies (e.g., adversarial debiasing, counterfactual data augmentation).
- Robustness & Adversarial Defense:
 - Adversarial Training: Incorporate noise, typos, and obfuscated slurs (character swaps, leetspeak) during training to harden the model against evasion.
 - Ensemble Methods: Combine multiple architectures (BERT, CNN-GRU, BiLSTM) in an ensemble or stacked model to reduce single-model weaknesses and improve stability.
- Lightweight & Edge Deployment

- Model Compression: Apply quantization or pruning to transformer and RNN models so they can run on edge devices or mobile apps, enabling on-device moderation with minimal latency.
- TinyML Integration: Explore microcontroller-compatible architectures for offline moderation in low-connectivity scenarios (e.g., community kiosks).
- API & Ecosystem Integration:
 - RESTful & gRPC APIs: Expose moderation services via robust, authenticated APIs so third-party platforms, CMSs, or chat apps can easily plug in.
 - Plugin Architecture: Develop plugins or SDKs for popular platforms (WordPress, Discourse, Slack, Discord) to streamline adoption by online communities.
- Comprehensive User Studies & A/B Testing:
 - Human Factors Evaluation: Conduct user studies measuring moderator efficiency, user satisfaction, and error rates before/after integration.
 - A/B Feature Rollouts: Experiment with different UI presentations of moderation decisions (e.g., immediate removal vs. warning), tracking engagement and appeal rates.
- Extended Toxicity Dimensions:
 - Beyond Binary Labels: Introduce severity scoring or toxicity “intensity” levels to allow tiered moderation actions (e.g., soft warning vs. account suspension).
 - Additional Abuse Categories: Expand taxonomy to cover misinformation, spam, self-harm content, or coordinated harassment campaigns.
- Compliance & Privacy Enhancements:
 - Privacy-Preserving Training: Research federated learning or differential privacy techniques so platforms can improve models without centralizing user data.
 - Audit Logging & Governance: Implement immutable audit logs for every moderation decision to meet legal or organizational compliance requirements.

10. References

- [1] A. Schmidt and M. Wiegand, “A survey on hate speech detection using natural language processing,” in *Proc. 5th Workshop on Natural Language Processing for Social Media*, 2017, pp. 1–10.
- [2] T. Davidson, D. Warmley, 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)*, 2017, pp. 512–515.
- [3] P. Badjatiya, S. Gupta, M. Gupta, and V. Varma, “Deep learning for hate speech detection in tweets,” in *Proc. 26th Int. Conf. on World Wide Web Companion (WWW)*, 2017, pp. 759–760.

- [4] Z. Wang and B. Zhang, “Toxic comment classification based on bidirectional gated recurrent unit and convolutional neural network,” *ACM Trans. Asian and Low-Resource Language Information Processing*, vol. 21, no. 3, pp. 1–12, 2021.
- [5] Z. Zhao, Z. Zhang, and F. Hopfgartner, “A comparative study of using pre-trained language models for toxic comment classification,” in *Proc. Web Conf. Companion*, 2021, pp. 500–507.
- [6] M. T. Ribeiro, S. Singh, and C. Guestrin, ““Why should I trust you?": Explaining the predictions of any classifier,” in *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.