Python For Data Science Project Report

Toxic Comment Classification

Group Id: 6

Team Members:

Suyash Tiwari (S20220020315)
Shrijeet Kumar (S20220020312)
Unnati Agarwal (S20220020318)
Aditya Pande (S20220010007)

Contents

1 Introduction

1.1 Problem Statement

We are working on 'Toxic Comments Classification' problem statement. In this project, we have tried to do analysis and build some machine learning model that can accurately label the type of comment and classify it.

1.2 Objectives

Recently, The world has witnessed an exponential growth in social medium platform and on-line communities, Albeit such growth create various opportunities for connection and communication,It also has a flip side to it; toxic speech, hateful content and on-line harassment.Such behavior can be detrimental for various individuals and if not handled with care can unleash a bigger chaos. To tackle such issues and prevent them from escalating further, we have developed models that leverage the power of advanced techniques like Natural Language Processing, Deep Learning, Mlflow, etc. to detect such toxic behaviors in its nascent stage, preventing it from causing further harm to society and individuals,.

1.3 Project Scope

In our project we have focused mainly on the analysis part and also on the best model building part in an experimental environment. This project is a kind of research and experimentation and not a product. But this project can be used to build some amazing products and integrate them with social networks.

Figure 1: Visual representation of the problem we're addressing

2 Data Source and Information

2.1 Dataset Overview

The dataset has been taken from Kaggle, named Hate Speech Detection in Social Media dataset.

2.2 Data Description

In this dataset we have in total 8 columns : id , comment_text , toxic , severe_toxic , obscene , threat , insult , identity_hate. Each comment has a unique id with labels , which has been given value as either 0 or 1 .

Table 1: Dataset Summary

Feature	Type	Description	Missing Values (%)
comment_text	object	Text content of the comment	0.00
toxic	int64	Binary label indicating toxicity	0.00
severe_toxic	int64	Binary label for severe toxicity	0.00
obscene	int64	Binary label for obscenity	0.00
threat	int64	Binary label for threats	0.00
insult	int64	Binary label for insults	0.00
identity_hate	int64	Binary label for identity-based hate	0.00

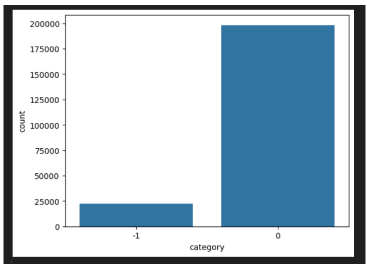
2.3 Exploratory Data Analysis (EDA)

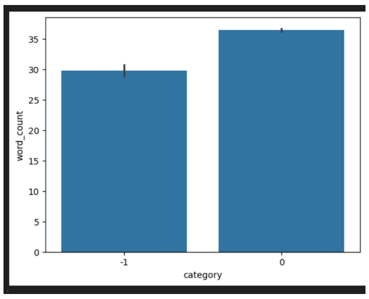
Our dataset initially comprised 223,549 entries and seven columns: `comment_text`, `toxic`, `severe_toxic`, `obscene`, `threat`, `insult`, and `identity_hate`. An initial inspection revealed 2,871 missing values in the `comment_text` column (of type string), while the other toxicity columns (of type int64) had no missing values.

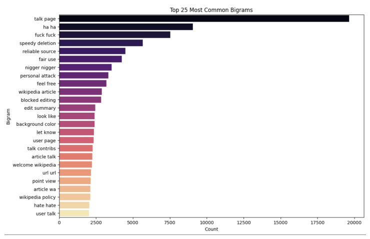
Since the `comment_text` column contained a significant number of missing values, we removed the corresponding rows, reducing the dataset to 220,678 entries. Duplicate rows were checked and none were found. To simplify classification, a new column `category` was generated such that a row is labeled 0 if any toxicity label is 1 (toxic comment) and 1 if none are 1 (non-toxic comment). The distribution was found to be approximately 92.75% for category 0 and 7.25% for category 1. The discrepant (-1) data was removed as it did not contribute meaningfully.

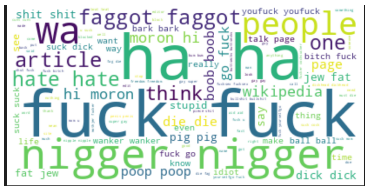
Additionally, we introduced a new column `word_count` to capture the number of words in each comment. A box plot of `word_count` against `category` revealed that toxic comments (category 0) differ in length from non-toxic comments (category 1), indicating the need for further text preprocessing (e.g., URL removal, abbreviation handling, and lemmatization).

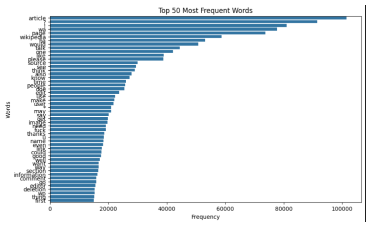
Further exploratory visualizations were carried out using bigrams, which showed that phrases like “talk page”, “ha ha”, and “fuckfuck” are among the most frequent, highlighting the presence of offensive language. In addition, lemmatization was applied and the top 50 most frequent words were plotted (with “article”, “page”, and “wikipedia” emerging as common terms), and a wordcloud was generated to further visualize these patterns.











2.4 Advanced Preprocessing

Based on the insights gained from the EDA, advanced preprocessing steps were applied. These included:

- Removing missing values from `comment_text` to ensure data quality.
- Dropping duplicate rows.
- Creating a new binary `category` column to streamline toxic versus non-toxic classification.
- Computing a `word_count` for each comment to analyze text length differences.
- Applying advanced text processing techniques such as URL removal, abbreviation handling, and lemmatization.

2.5 Work Flow And Architecture

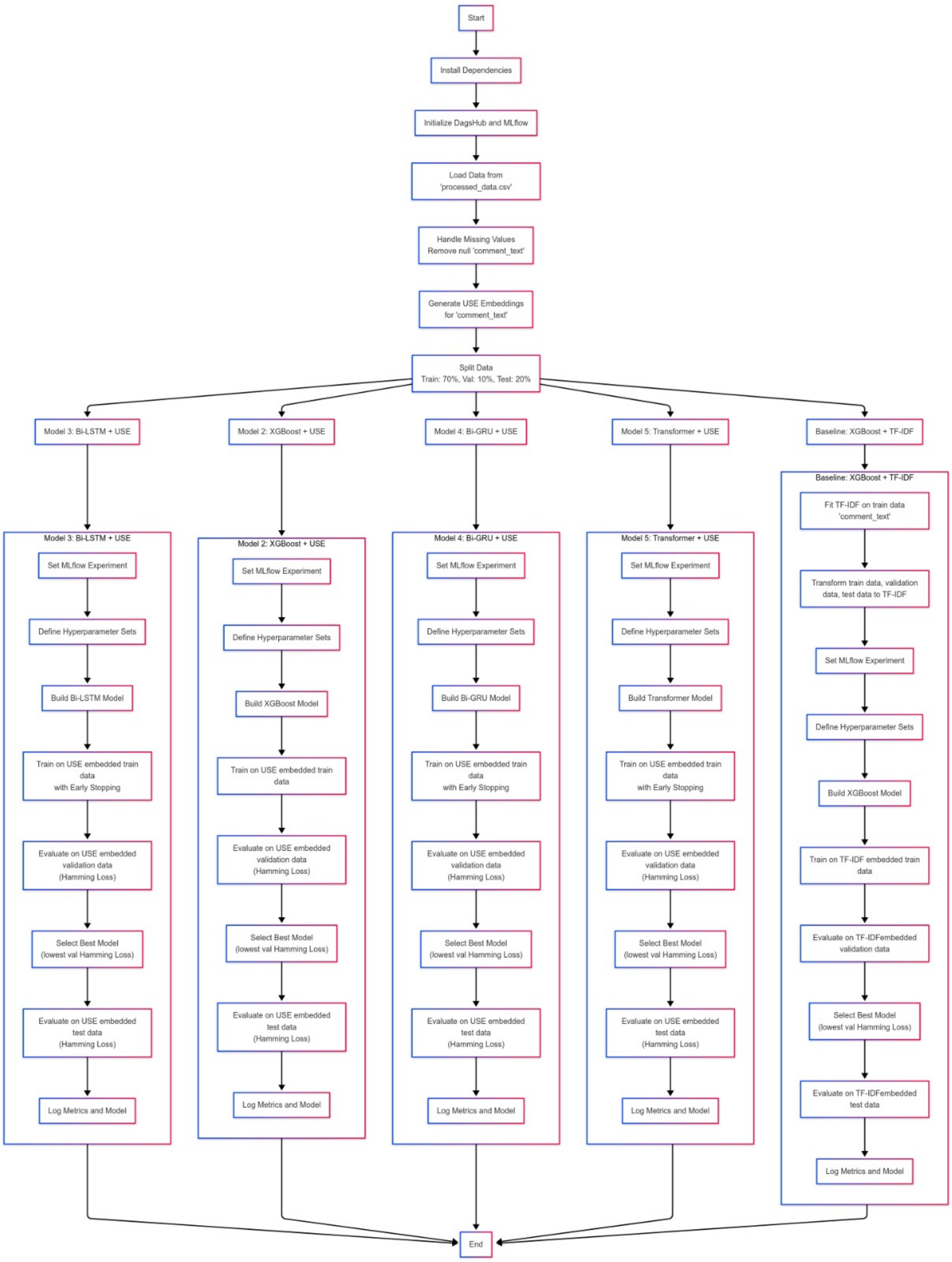

Start
Install Dependencies
Initialize DagsHub and MLflow
Load Data from 'processed_data.csv'
Handle Missing Values Remove null 'comment_text'
Generate USE Embeddings for 'comment_text'
Split Data Train: 70%, Val: 10%, Test: 20%
Model 3: Bi-LSTM + USE
Model 2: XGBoost + USE
Model 4: Bi-GRU + USE
Model 5: Transformer + USE
Baseline: XGBoost + TF-IDF
Model 3: Bi-LSTM + USE
Set MLflow Experiment
Define Hyperparameter Sets
Build Bi-LSTM Model
Train on USE embedded train data with Early Stopping
Evaluate on USE embedded validation data (Hamming Loss)
Select Best Model (lowest val Hamming Loss)
Evaluate on USE embedded test data (Hamming Loss)
Log Metrics and Model
Model 2: XGBoost + USE
Set MLflow Experiment
Define Hyperparameter Sets
Build XGBoost Model
Train on USE embedded train data
Evaluate on USE embedded validation data (Hamming Loss)
Select Best Model (lowest val Hamming Loss)
Evaluate on USE embedded test data (Hamming Loss)
Log Metrics and Model
Model 4: Bi-GRU + USE
Set MLflow Experiment
Define Hyperparameter Sets
Build Bi-GRU Model
Train on USE embedded train data with Early Stopping
Evaluate on USE embedded validation data (Hamming Loss)
Select Best Model (lowest val Hamming Loss)
Evaluate on USE embedded test data (Hamming Loss)
Log Metrics and Model
Model 5: Transformer + USE
Set MLflow Experiment
Define Hyperparameter Sets
Build Transformer Model
Train on USE embedded train data with Early Stopping
Evaluate on USE embedded validation data (Hamming Loss)
Select Best Model (lowest val Hamming Loss)
Evaluate on USE embedded test data (Hamming Loss)
Log Metrics and Model
Baseline: XGBoost + TF-IDF
Fit TF-IDF on train data 'comment_text'
Transform train data, validation data, test data to TF-IDF
Set MLflow Experiment
Define Hyperparameter Sets
Build XGBoost Model
Train on TF-IDF embedded train data
Evaluate on TF-IDFembedded validation data
Select Best Model (lowest val Hamming Loss)
Evaluate on TF-IDFembedded test data
Log Metrics and Model
End

Figure 2: Entire Work Flow

3 Experimental Models

We evaluated multiple models for toxic comment classification, progressively exploring more sophisticated architectures. All experiments except the baseline used the Universal Sentence Encoder (USE) for text embeddings.

3.1 XGBoost with TF-IDF (Baseline)

XGBoost with TF-IDF features serves as our baseline. TF-IDF transforms text into sparse vectors where each element represents word importance:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{1}$$

$$\text{IDF}(t) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2}$$

where t is a term, d is a document, N is the total number of documents, and $|\{d \in D : t \in d\}|$ is the number of documents containing term t. XGBoost then builds an ensemble of decision trees using gradient boosting to minimize the objective function:

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{3}$$

where l is the loss function, Ω is the regularization term, and f_k represents the k-th tree.

XGBoost improves upon traditional boosting methods by introducing regularization techniques to prevent overfitting, handling missing values efficiently, and utilizing a sparsity-aware algorithm for optimal feature selection. This makes it a strong baseline for text classification.

3.2 XGBoost with Universal Sentence Encoder

This model replaces TF-IDF with USE embeddings, which capture semantic meaning more effectively than sparse vectors. USE maps sentences to 512-dimensional vectors using a transformer network, preserving contextual relationships between words.

Unlike TF-IDF, which relies on statistical word frequency, USE leverages deep learning to encode meaning, making it more robust to variations in wording and phrasing. This allows for improved classification performance, especially in detecting nuanced forms of toxicity.

3.3 BiLSTM with Universal Sentence Encoder

Bidirectional Long Short-Term Memory networks process USE embeddings to capture sequential patterns in both directions. The BiLSTM computes:

$$\overrightarrow{h_t} = \text{LSTM}(\overrightarrow{h_{t-1}}, x_t) \tag{4}$$

$$\overleftarrow{h_t} = \text{LSTM}(\overleftarrow{h_{t+1}}, x_t) \tag{5}$$

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}] \tag{6}$$

where $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ are forward and backward hidden states, and x_t is the USE embedding at position t.

LSTMs are designed to address the vanishing gradient problem in traditional RNNs, allowing them to retain long-range dependencies. The bidirectional nature of BiLSTM enhances context comprehension, making it suitable for toxicity detection where word order and contextual relationships matter.

3.4 BiGRU with Universal Sentence Encoder

Bidirectional Gated Recurrent Units offer a simplified architecture compared to LSTMs while maintaining effective sequence modeling. The GRU updates are:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{7}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{8}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t]) \tag{9}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{10}$$

where z_t is the update gate, r_t is the reset gate, $\tilde{h}_t$ is the candidate state, and $\odot$ represents element-wise multiplication.

GRUs simplify LSTM architecture by combining the forget and input gates into a single update gate. This reduces the number of parameters and computational complexity while maintaining comparable performance, making GRUs an efficient alternative for sequential text analysis.

3.5 Transformer with Universal Sentence Encoder

Our transformer model processes USE embeddings through self-attention mechanisms:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{11}$$

with multi-head attention allowing parallel focus on different representation subspaces:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{12}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{13}$$

where W^Q, W^K, W^V, and W^O are parameter matrices.

Transformers eliminate recurrence, using self-attention to model relationships between words regardless of their distance in the text. This enables parallel computation, making transformers highly efficient and capable of capturing intricate patterns in toxic comment classification.

Overall, each model progressively enhances contextual understanding, making them increasingly effective for the classification task.

4 Experimental Results

4.1 Experiment 1: XG-Boost + TfIdf

This experiment investigates the performance of XG-Boost with TfIdf. The model is configured with the following parameters and evaluated based on key performance metrics.

Model Parameters

Table 2: Model Parameters for Experiment 1

Parameter	Value
estimator learning rate	0.1
estimator max depth	6
estimator n estimators	200

Performance Metrics

Table 3: Performance Metrics for Experiment 1

Metric	Value
Best Hamming loss	0.02263458401305057 %

F1 Score Visualization

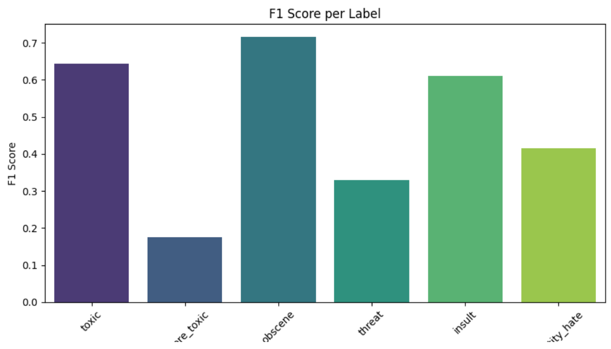


Figure 3: F1 Score Visualization for Experiment 1

4.2 Experiment 2: XGBoost + Universal Sentence Encoder

In this experiment, XGBoost + Universal Sentence Encoder is evaluated using a slightly modified configuration. Below are the parameters and performance metrics recorded.

Model Parameters

Table 4: Model Parameters for Experiment 1

Parameter	**Value**
estimator learning rate	0.1
estimator max depth	3
estimator n estimators	200

Performance Metrics

Table 5: Performance Metrics for Experiment 1

Metric	**Value**
Best Hamming loss	0.023514440215092743 %

F1 Score Visualization

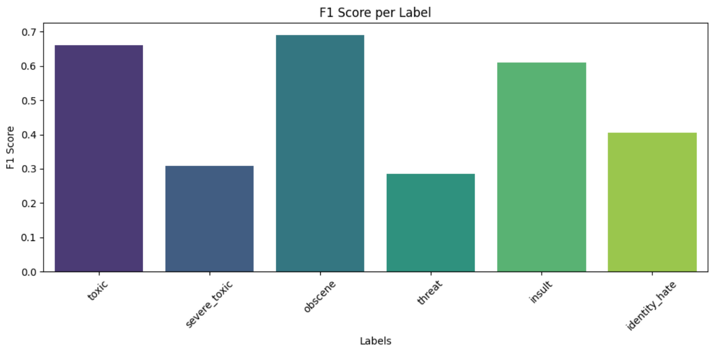


Figure 4: F1 Score Visualization for Experiment 1

4.3 Experiment 3: BiLSTM Model + Universal Sentence Encoder Model

This section presents the results of Experiment 3, where the BiLSTM Model was further refined to capture additional features through parameter tuning and architecture optimization.

Model Parameters

Table 6: Model Parameters for Experiment 3

Parameter	**Value**
stm_units	128
dropout	0.3
learning_rate	0.001

Performance Metrics

Table 7: Performance Metrics for Experiment 3

Metric	Value
f1_toxic	0.6853
f1_identity_hate	0.4877
test_auc	0.9727
f1_severe_toxic	0.2087
f1_insult	0.6479
test_loss	0.0599
test_accuracy	0.9942
hamming_loss	0.0220
f1_obscene	0.7205
f1_threat	0.2637

4.4 Experiment 4: BiGRU-Grid-Search + Universal Sentence Encoder Model

In Experiment 4, we further optimize the BiGRU model using grid-search and advanced techniques, combined with the Universal Sentence Encoder for richer semantic representations. The model parameters and evaluation metrics are summarized below.

Model Parameters

Table 8: Model Parameters for Experiment 4

Parameter	Value
gru_units	128
dropout	0.3
learning_rate	0.001

Performance Metrics

Table 9: Performance Metrics for Experiment 4

Metric	Value
test_auc	0.9722
test_loss	0.0601
hamming_loss	0.0220
f1_toxic	0.6855
f1_severe_toxic	0.1753
test_accuracy	0.9927
f1_obscene	0.7068
f1_threat	0.2527
f1_insult	0.6259
f1_identity_hate	0.4528

F1 Score Visualization

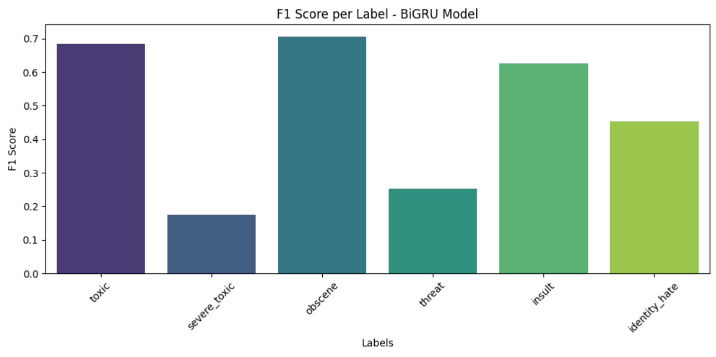


Figure 5: F1 Score Visualization for Experiment 4

4.5 Experiment 5: Transformer-Grid-Search + Universal Sentence Encoder Model

The final experiment, Experiment 5, explores the performance of the Transformer-Grid-Search model with the Universal Sentence Encoder under the latest configuration. The results, including key parameters and performance metrics, are provided below.

Model Parameters

Table 10: Model Parameters for Experiment 5

Parameter	Value
best_embed_dim	128
best_head_size	64
best_num_heads	2
best_ff_dim	128
best_num_transformer_blocks	2
best_mlp_units	[64]
best_dropout	0.1
best_mlp_dropout	0.2
best_learning_rate	0.001

Performance Metrics

Table 11: Performance Metrics for Experiment 5

Metric	Value
test_loss	0.06024
test_accuracy	0.99536
test_auc	0.97363
hamming_loss	0.02241
f1_toxic	0.67149
f1_insult	0.63523
f1_severe_toxic	0.14865
f1_obscene	0.70386
f1_threat	0.03947
f1_identity_hate	0.25475

F1 Score Visualization

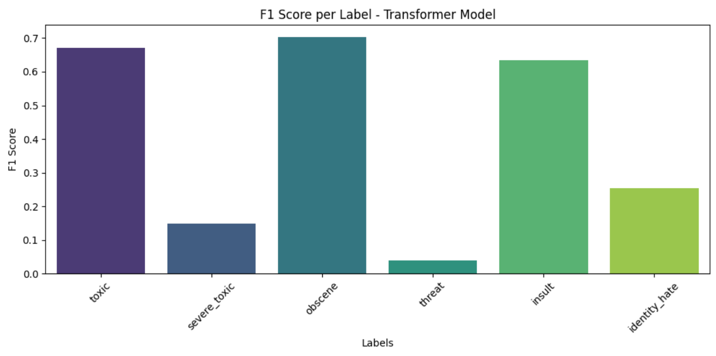


Figure 6: F1 Score Visualization for Experiment 5

5 Comparative Analysis

5.1 Hamming Loss Comparison

Below is a table summarizing the Hamming Loss across all experiments:

Table 12: Hamming Loss Comparison Across All Experiments

Experiment	**Hamming Loss**
Exp 1: XGBoost + TfIdf	0.02263
Exp 2: XGBoost + USE	0.02351
Exp 3: BiLSTM + USE	0.02200
Exp 4: BiGRU + USE	0.02200
Exp 5: Transformer + USE	0.02241

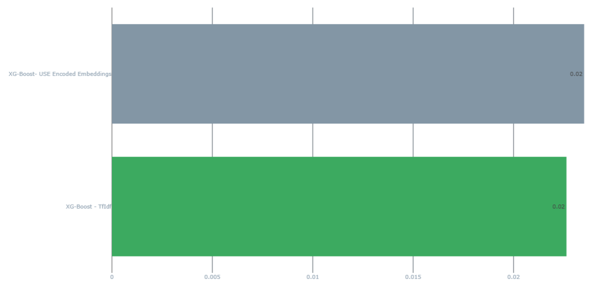


Figure 7: Hamming Loss Visualization for the Best ML Model (Exp 1: XGBoost + TfIdf)

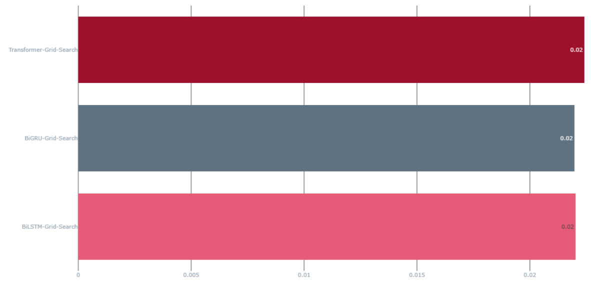
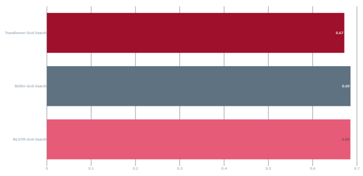


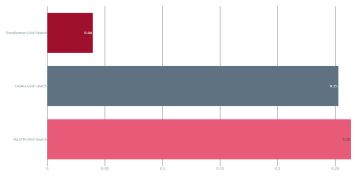
Figure 8: Hamming Loss Visualization for DL Models

5.2 F1 Scores by Category

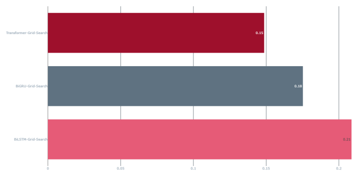
Below are F1 score plots for each toxicity category across the experiments.



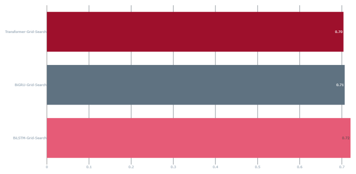
(a) F1 Score (Toxic)



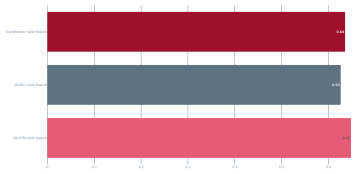
(b) F1 Score (Threat)



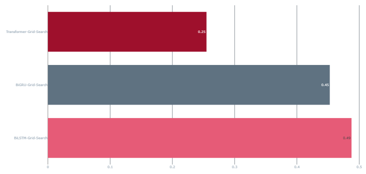
(c) F1 Score (Severe Toxic)



(d) F1 Score (Obscene)



(e) F1 Score (Insult)



(f) F1 Score (Identity Hate)

Figure 9: F1 Score Comparisons for Different Toxicity Categories

6 Results and Conclusion

6.1 Summary of Findings

The comparative analysis of our experiments shows that the deep learning models achieved very similar Hamming Loss values, ranging from 0.02200 to 0.02241. Specifically, the XGBoost-based model (Experiment 1) had a Hamming Loss of 0.02263, which is marginally higher compared to the DL models. Additionally, the F1 score plots across various toxicity categories indicate that while each model has its strengths, the transformer-based approach (Experiment 5) consistently performed well across all categories.

6.2 Best Model Selection

After reviewing the performance metrics and visualizations, the Transformer-Grid-Search model combined with the Universal Sentence Encoder (Experiment 5) is selected as the best model. This model not only achieved a competitive Hamming Loss of 0.02241 and high test accuracy but also delivered robust and balanced F1 scores across different toxicity classes. Its ability to capture semantic nuances makes it a strong candidate for toxic comment classification.