# Deep Learning for Comment Toxicity Detection with Streamlit

## Problem Statement:

- Online communities and social media platforms have become integral parts of modern communication, facilitating interactions and discussions on various topics. However, the prevalence of toxic comments, which include harassment, hate speech, and offensive language, poses significant challenges to maintaining healthy and constructive online discourse. To address this issue, there is a pressing need for automated systems capable of detecting and flagging toxic comments in real-time.

## Objective:

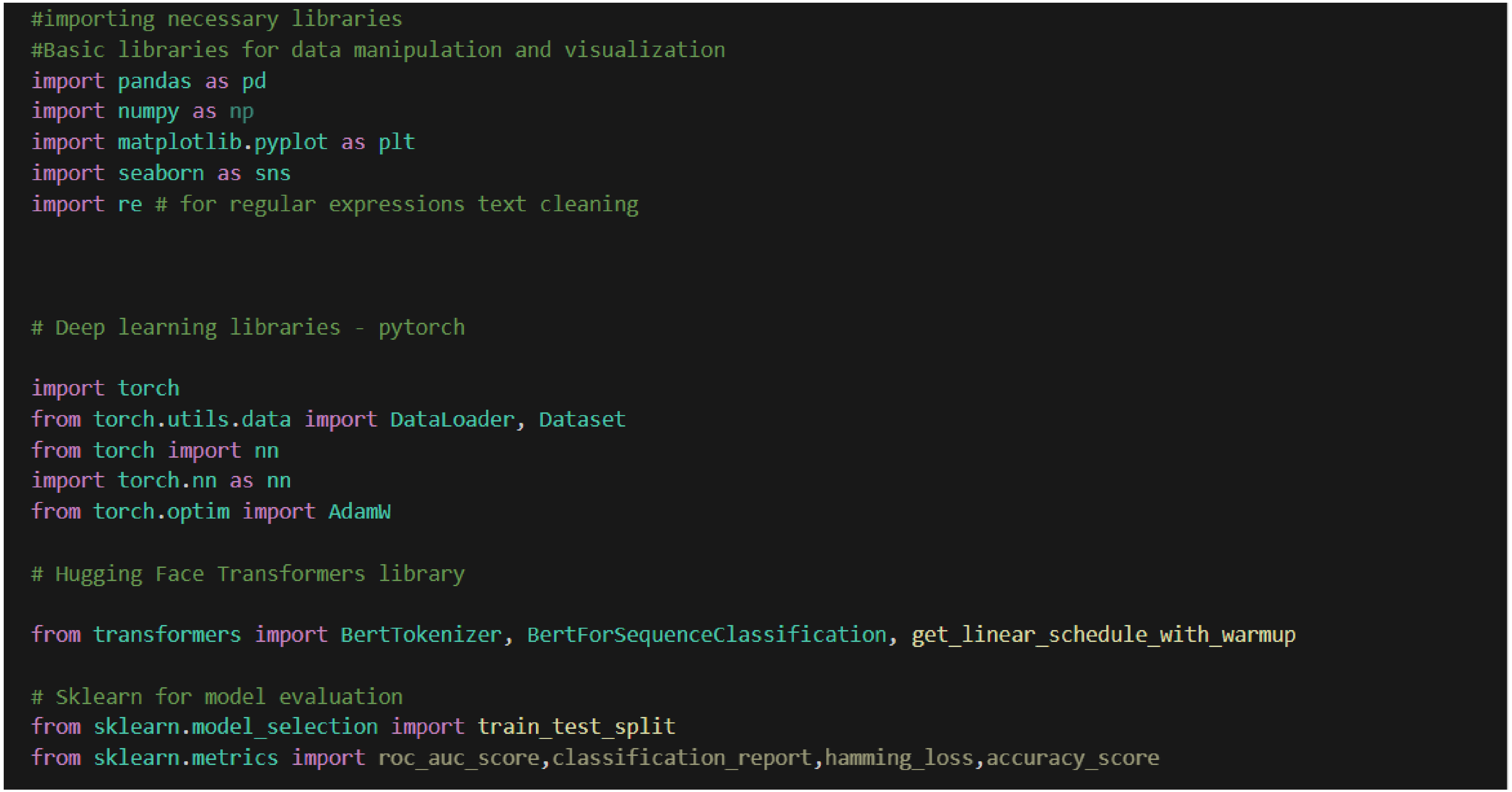
- The objective of this project is to develop a deep learning-based comment toxicity model using Python. This model will analyze text input from online comments and predict the likelihood of each comment being toxic. By accurately identifying toxic comments, the model will assist platform moderators and administrators in taking appropriate actions to mitigate the negative impact of toxic behavior, such as filtering, warning users, or initiating further review processes.

## Business Use Cases:

- **Social Media Platforms:** Social media platforms can utilize the developed comment toxicity model to automatically detect and filter out toxic comments in real-time.
- **Online Forums and Communities:** Forums and community websites can integrate the toxicity detection model to moderate user-generated content efficiently.
- **Content Moderation Services:** Companies offering content moderation services for online platforms can leverage the developed model to enhance their moderation capabilities.
- **Brand Safety and Reputation Management:** Brands and advertisers can use the toxicity detection model to ensure that their advertisements and sponsored content appear in safe and appropriate online environments.
- **E-learning Platforms and Educational Websites:** E-learning platforms and educational websites can employ the toxicity detection model to create safer online learning environments for students and educators.
- **News Websites and Media Outlets:** News websites and media outlets can utilize the toxicity detection model to moderate user comments on articles and posts.

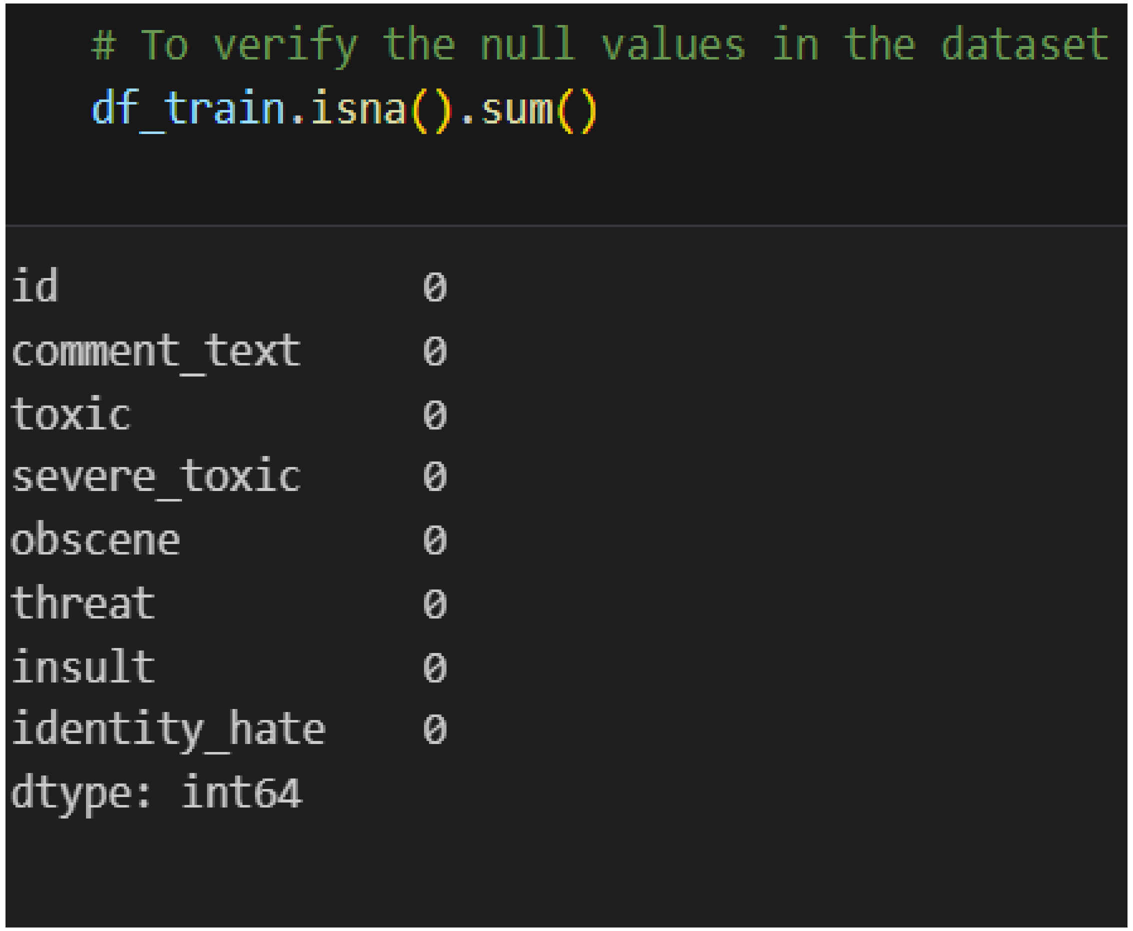
## Technical Tags:

- *  Python
- *  Deep Learning
- *  Neural Networks
- *  NLP
- *  Model Training
- *  Model Evaluation
- *  Streamlit
- *  Model Deployment



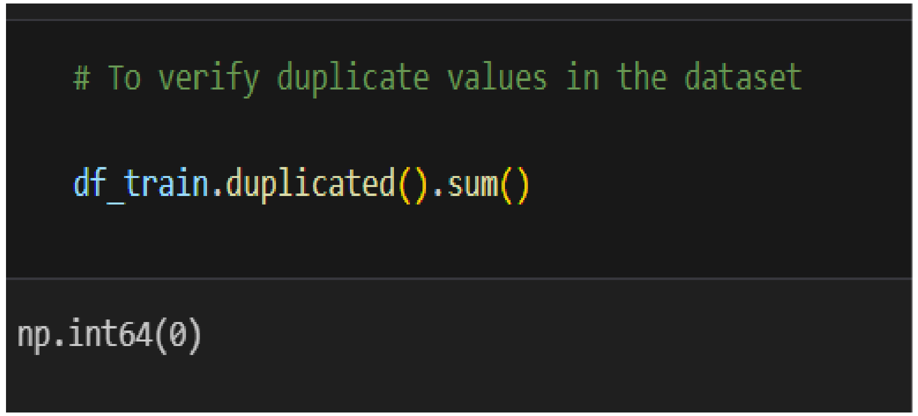
```python
#importing necessary libraries
#Basic libraries for data manipulation and visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re # for regular expressions text cleaning



# Deep learning libraries - pytorch

import torch
from torch.utils.data import DataLoader, Dataset
from torch import nn
import torch.nn as nn
from torch.optim import AdamW

# Hugging Face Transformers library

from transformers import BertTokenizer, BertForSequenceClassification, get_linear_schedule_with_warmup

# Sklearn for model evaluation
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score,classification_report,hamming_loss,accuracy_score
```

## 1.Data Cleaning



```
# To verify the null values in the dataset
df_train.isna().sum()
```

```
id               0
comment_text     0
toxic            0
severe_toxic     0
obscene          0
threat           0
insult           0
identity_hate    0
dtype: int64
```



```
# To verify duplicate values in the dataset

df_train.duplicated().sum()
```

```
np.int64(0)
```

# 2. EDA

## Comment Length Distribution

```
#Comment length and its histogram plot
```

X-axis (Comment Length): Ranges from 0 to about 6000 characters.

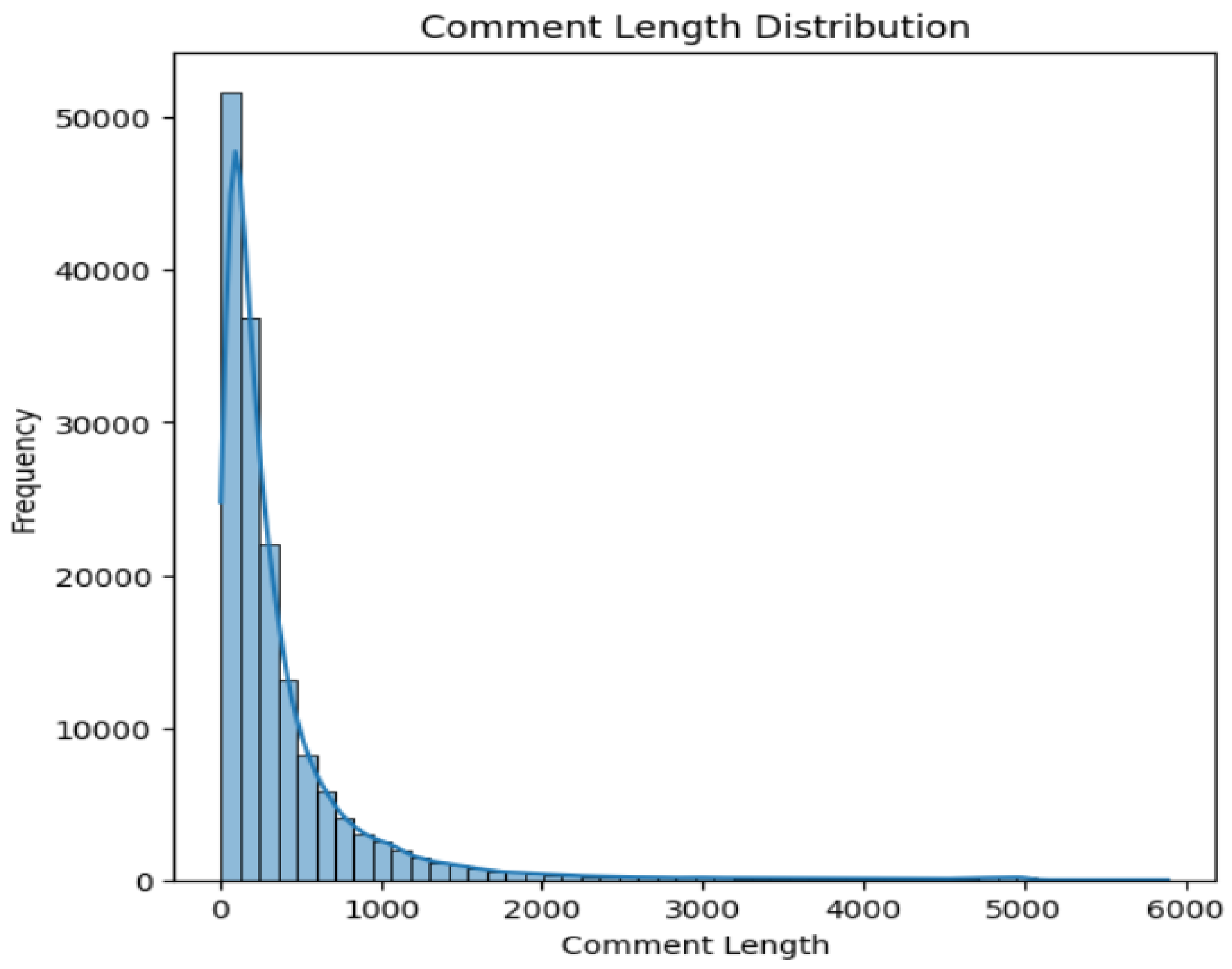
Y-axis (Frequency): Peaks over 50,000 at the lowest lengths.

The distribution is right-skewed:

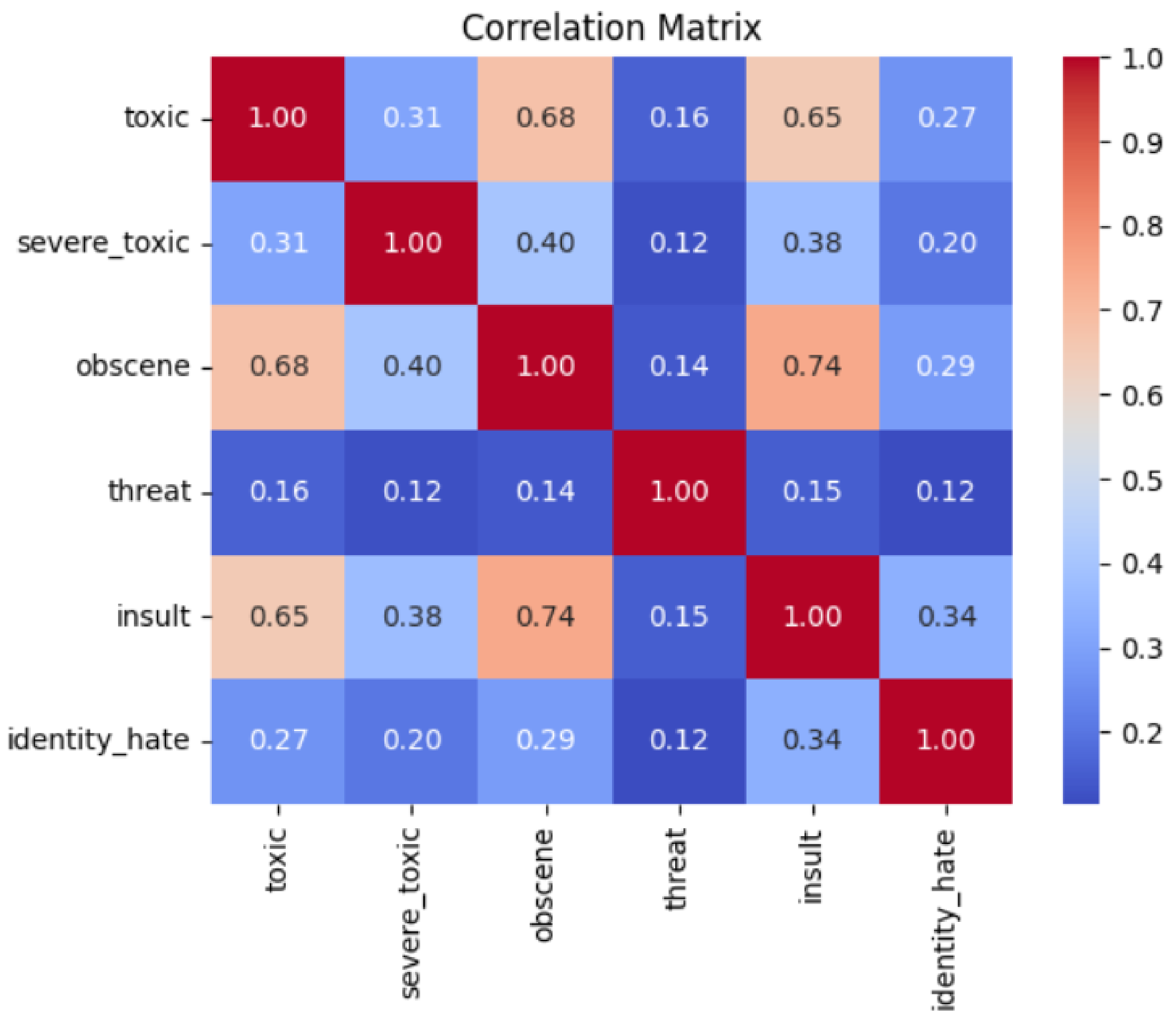
Most comments are very short (under 500 characters).

The frequency drops steeply as comment length increases.

A long tail indicates the presence of much longer comments, but these are rare.



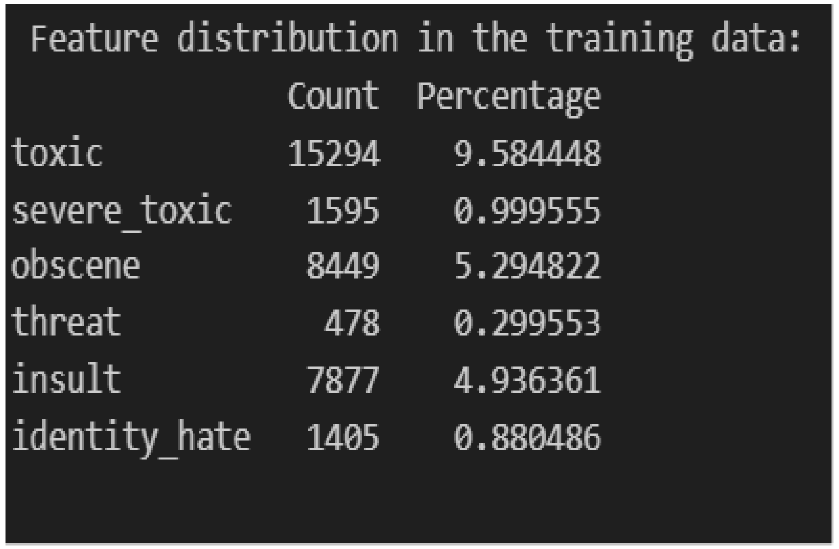
# Correlation matrix



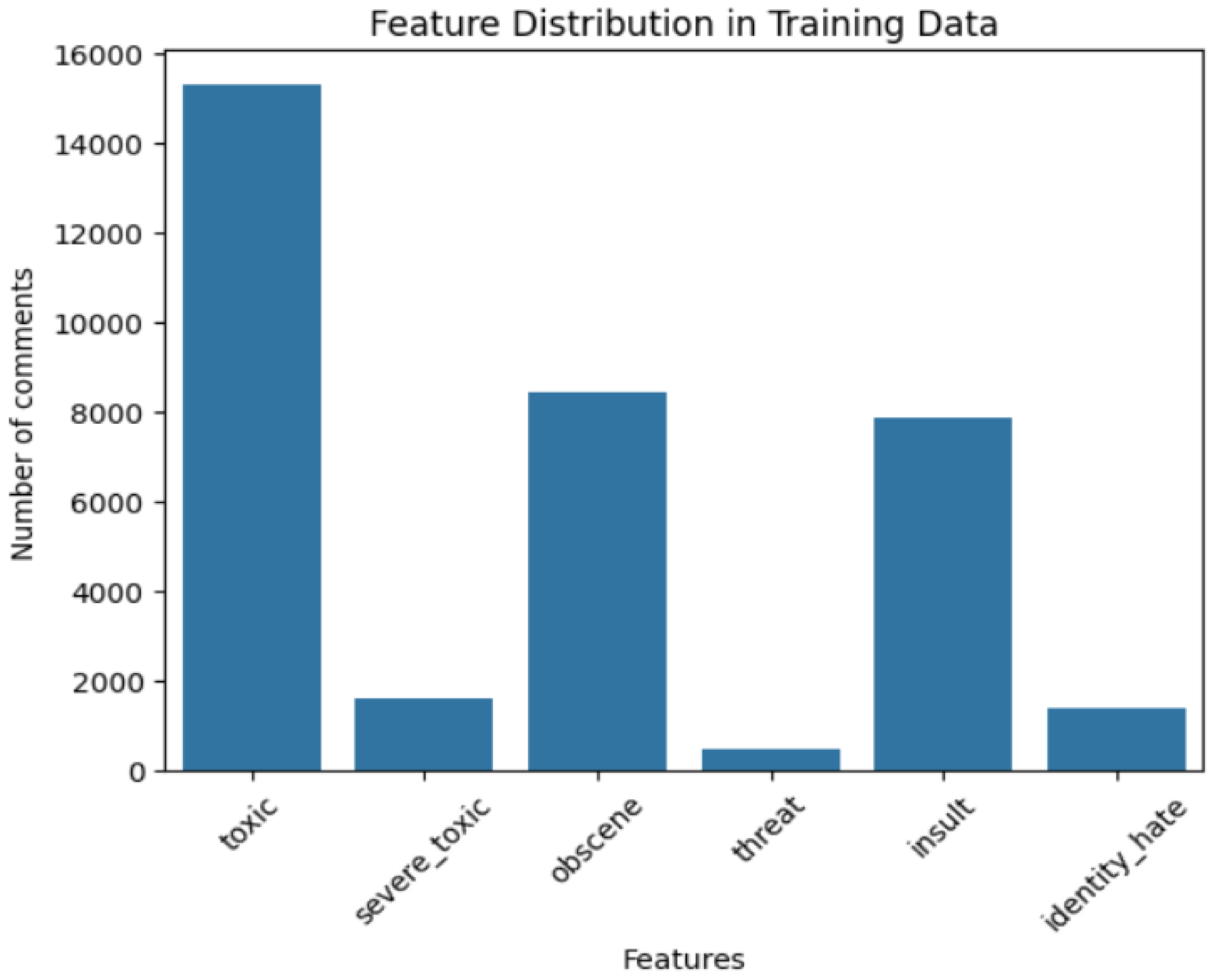
| | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|
| toxic | 1.00 | 0.31 | 0.68 | 0.16 | 0.65 | 0.27 |
| severe_toxic | 0.31 | 1.00 | 0.40 | 0.12 | 0.38 | 0.20 |
| obscene | 0.68 | 0.40 | 1.00 | 0.14 | 0.74 | 0.29 |
| threat | 0.16 | 0.12 | 0.14 | 1.00 | 0.15 | 0.12 |
| insult | 0.65 | 0.38 | 0.74 | 0.15 | 1.00 | 0.34 |
| identity_hate | 0.27 | 0.20 | 0.29 | 0.12 | 0.34 | 1.00 |

- toxic & insult 0.65 Strong positive correlation. Comments labeled as "toxic" are often also marked as "insult".
- toxic & obscene 0.68 Strong positive correlation. Obscene language is commonly part of toxic comments.
- toxic & severe_toxic 0.31 Moderate correlation. Not all toxic comments are considered severely toxic.
- toxic & threat 0.16 Weak correlation. Most toxic comments are not threats.
- toxic & identity_hate 0.27 Weak to moderate correlation. Some overlap exists but not strong.
- obscene & insult | 0.74 | Very strong correlation. Obscene comments are usually also insulting. |
- | obscene & severe_toxic | 0.40 | Moderate correlation. Severe toxicity may often include obscene words. |
- | severe_toxic & threat | 0.12 | Weak. Severe toxicity doesn't necessarily imply threats. |
- | threat & any other label | ~0.12 - 0.16 | Consistently low correlations. Threats are a more distinct category. |
- | identity_hate & others | Mostly ~0.20 - 0.34 | Weak to moderate. Some overlap but less predictable co-occurrence. |

# Feature distribution in the training data



```
Feature distribution in the training data:
               Count  Percentage
toxic          15294    9.584448
severe_toxic    1595    0.999555
obscene         8449    5.294822
threat           478    0.299553
insult          7877    4.936361
identity_hate   1405    0.880486
```



# 3. Preprocessing and Tokenization ( BERT)

## 3.1BERT

- **BERT (Bidirectional Encoder Representations from Transformers)** is a powerful language representation model developed by Google in 2018.
- It's based on the Transformer architecture, and it's designed to understand the context of a word bidirectionally — meaning it looks at both the left and right context in a sentence.

🔍 **Key Features of BERT:**

- Bidirectional context.
- Unlike previous models that read text left-to-right or right-to-left, BERT reads in both directions simultaneously.
- Pre-trained and fine-tuned.
- BERT is pre-trained on a large corpus (like Wikipedia), then fine-tuned for specific tasks (e.g., sentiment analysis, question answering).
- Transformer-based.
- Built using the encoder part of the Transformer architecture, allowing it to handle long-range dependencies in text.

🧠 **How BERT Works:**

- Pre-training tasks:
- Masked Language Modeling (MLM): Randomly masks words in the input and trains the model to predict them
- Next Sentence Prediction (NSP): Trains the model to predict if one sentence follows another in contex
- Fine-tuning: After pre-training, BERT is adapted to specific tasks like classification, named entity recognition, or QA with a relatively small dataset.

## 3.2 Text Cleaning

✅ Recommended Text Cleaning for BERT:

- Lowercasing (if using uncased models like bert-base-uncased)
- Remove excessive whitespace
- Replace HTML entities (&amp;, &lt;, etc.)
- Normalize unicode (e.g., unicodedata.normalize)
- Optionally: remove URLs/emails if irrelevant

❌ Do Not:

- Do not remove stop words (they're important for context)
- Do not stem or lemmatize (BERT is subword-based)
- Do not aggressively strip punctuation

## 3.3 Tokenization with BERT tokenizer

- Tokenization with the BERT tokenizer is a key step in preparing text data for input into BERT models. Here's a concise overview of how it works:

◆ **What is Tokenization?**

- Tokenization is the process of splitting text into smaller units called tokens. BERT uses WordPiece tokenization, which breaks words into subword units.

◆ **BERT Tokenizer Characteristics**

- Lowercasing (optional): Converts text to lowercase (e.g., "Hello" → "hello") if using the bert-base-uncased model.
- WordPiece Tokenizer: Splits unknown or rare words into subwords, prefixed with ## if they're a continuation of a word.
- Special Tokens:
- [CLS]: Classification token added at the beginning.
- [SEP]: Separator token used between sentence pairs or at the end of a single sentence.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

## 3.4 Creating Pytorch Dataset and DataLoader

- Creating a Dataset and DataLoader in PyTorch is essential for efficiently handling and feeding data into a model during training or evaluation.

🔧 **Key Features of PyTorch:**

*Dynamic Computation Graphs (Define-by-Run):* You can change the architecture on-the-fly, which is great for debugging and flexibility

*Pythonic API:* Feels natural to use if you're familiar with Python and NumPy.

*GPU Acceleration:* Native support for CUDA to run models on NVIDIA GPUs.

*TorchScript:* Allows you to serialize and optimize models for production.

*Strong Community & Ecosystem:* Tools like torchvision, torchaudio, torchtext, and PyTorch Lightning.

🔍 **Use Cases:**

Computer vision (with torchvision)

Natural language processing (transformers from Hugging Face are built on PyTorch)

Reinforcement learning

Generative AI (GANs, VAEs, etc.)

## 4.Model Definition

- BertForSequenceClassification = BERT + classification head
- Automatically handles loss computation if labels are provided
- Use with tokenized inputs: input_ids, attention_mask, and optional labels

```
# Load the pre-trained BERT model for sequence classification
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=len(target_features))
# Move the model to designated device (GPU or CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

## 5.Model Training

## 5.1 Setup Optimizer and Scheduler

- The AdamW optimizer is the default optimizer used in most modern BERT implementations because it effectively handles weight decay.

🧠 **Why AdamW with BERT?**

- BERT has LayerNorm and bias terms that should not have weight decay.
- AdamW separates weight decay from gradient updates, which is correct behavior for L2 regularization in transformers.

## 5.2 Training Loop

- 📌 **Key Concepts**

| **Step** | | **Purpose** |
|---|---|---|
| • model.train() | - | Enables training behaviors like dropout |
| • optimizer.zero_grad() | - | Clears previously accumulated gradients |
| • loss.backward() | - | Computes gradients via backpropagation |
| • optimizer.step() | - | Updates model parameters using optimizer |
| • scheduler.step() | - | Adjusts learning rate (optional) |

## 5.3 Evaluation loop

- The evaluation loop  runs the model on a validation or test dataset without
- updating weights to measure performance.

✅ Purpose

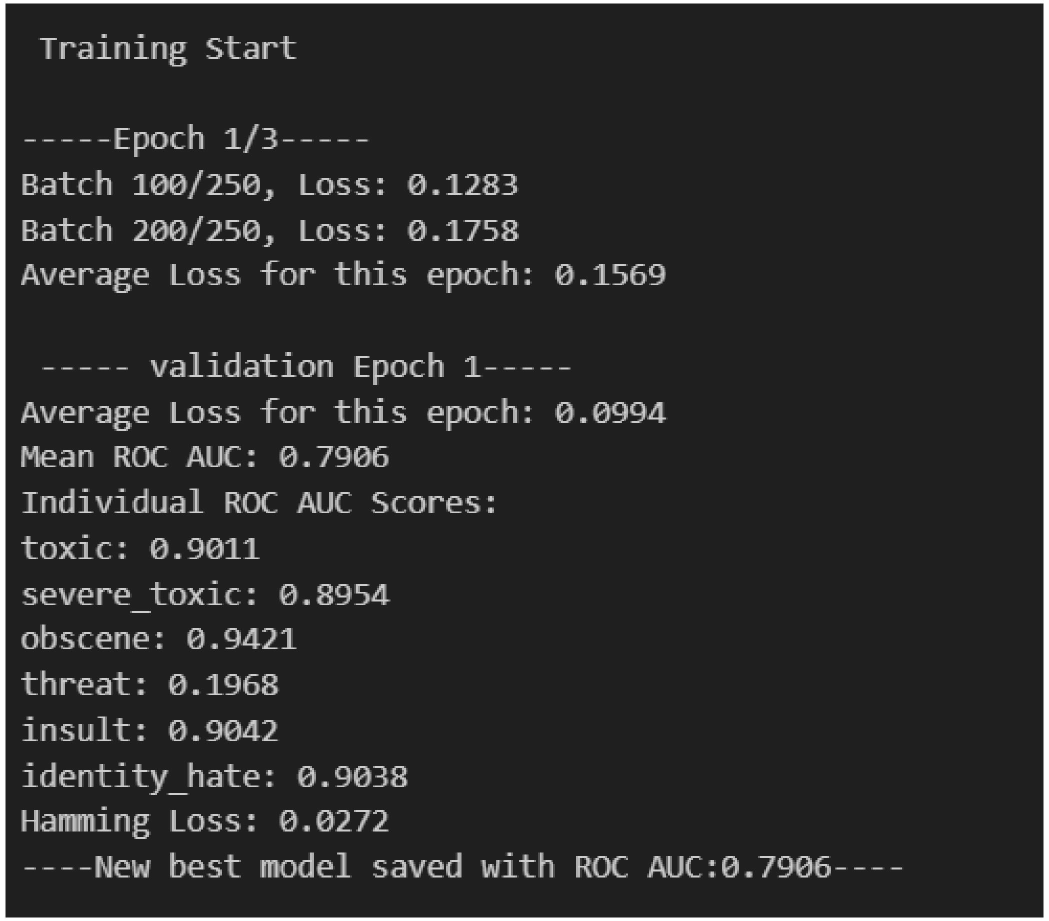
- Check model performance during or after training.
- Compute metrics: loss, accuracy, F1, etc.
- Ensure model.eval() is called to disable dropout, etc.
- No gradients: We use torch.no_grad() for efficiency.

📌 Key Concepts

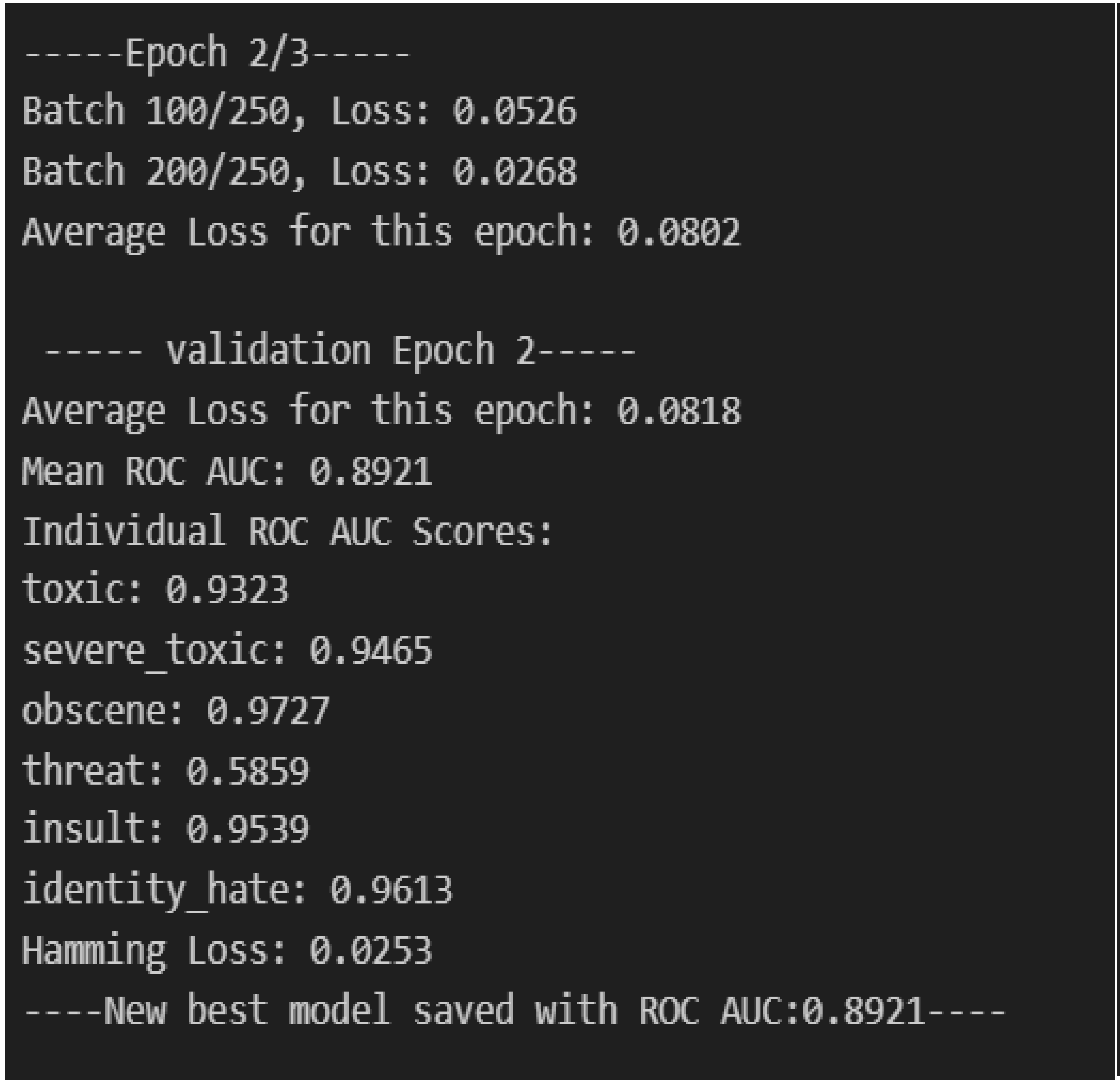
| Step | Description |
|---|---|
| model.eval() | Disables dropout, layernorm updates |
| torch.no_grad() | Speeds up computation, avoids storing gradients |
| loss_fn(outputs, labels) | Same loss as training (e.g., BCEWithLogitsLoss) |
| torch.sigmoid(outputs) > 0.5 | Converts logits to binary predictions |

## 5.4 Execute training and Evaluation Loop

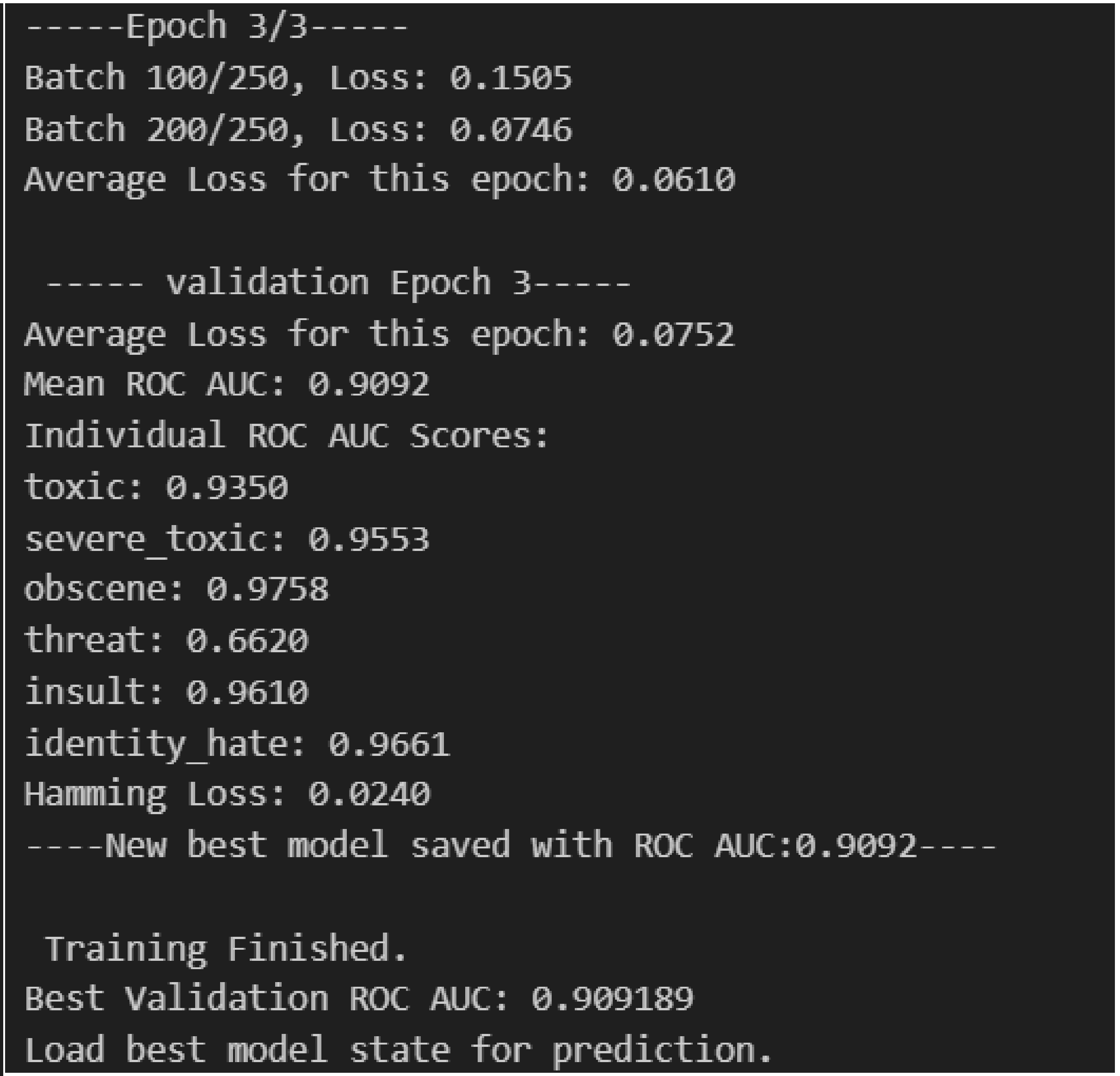
- Trains a PyTorch model over multiple epochs.
- Tracks and saves the best model (based on validation ROC AUC).
- Loads the best model after training for future use.



```
 Training Start

-----Epoch 1/3-----
Batch 100/250, Loss: 0.1283
Batch 200/250, Loss: 0.1758
Average Loss for this epoch: 0.1569

 ----- validation Epoch 1-----
Average Loss for this epoch: 0.0994
Mean ROC AUC: 0.7906
Individual ROC AUC Scores:
toxic: 0.9011
severe_toxic: 0.8954
obscene: 0.9421
threat: 0.1968
insult: 0.9042
identity_hate: 0.9038
Hamming Loss: 0.0272
----New best model saved with ROC AUC:0.7906----
```

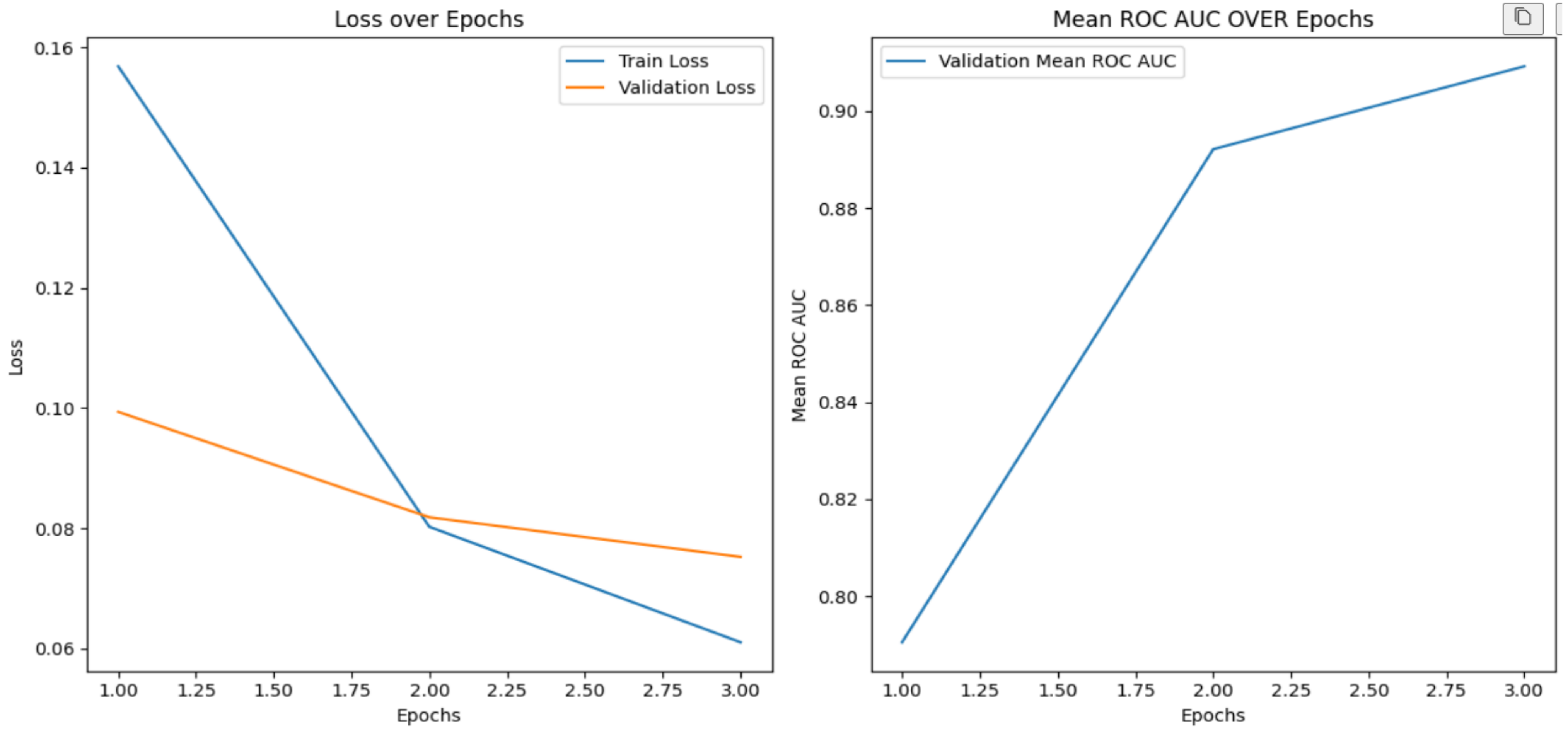


```
-----Epoch 2/3-----
Batch 100/250, Loss: 0.0526
Batch 200/250, Loss: 0.0268
Average Loss for this epoch: 0.0802

 ----- validation Epoch 2-----
Average Loss for this epoch: 0.0818
Mean ROC AUC: 0.8921
Individual ROC AUC Scores:
toxic: 0.9323
severe_toxic: 0.9465
obscene: 0.9727
threat: 0.5859
insult: 0.9539
identity_hate: 0.9613
Hamming Loss: 0.0253
----New best model saved with ROC AUC:0.8921----
```



```
-----Epoch 3/3-----
Batch 100/250, Loss: 0.1505
Batch 200/250, Loss: 0.0746
Average Loss for this epoch: 0.0610

 ----- validation Epoch 3-----
Average Loss for this epoch: 0.0752
Mean ROC AUC: 0.9092
Individual ROC AUC Scores:
toxic: 0.9350
severe_toxic: 0.9553
obscene: 0.9758
threat: 0.6620
insult: 0.9610
identity_hate: 0.9661
Hamming Loss: 0.0240
----New best model saved with ROC AUC:0.9092----

 Training Finished.
Best Validation ROC AUC: 0.909189
Load best model state for prediction.
```

# Plot Training History



## 6.Prediction on Test Set

- Use the trained (best) model to generate predictions on the unseen test data

### 6.1 Prepare test Data Loader

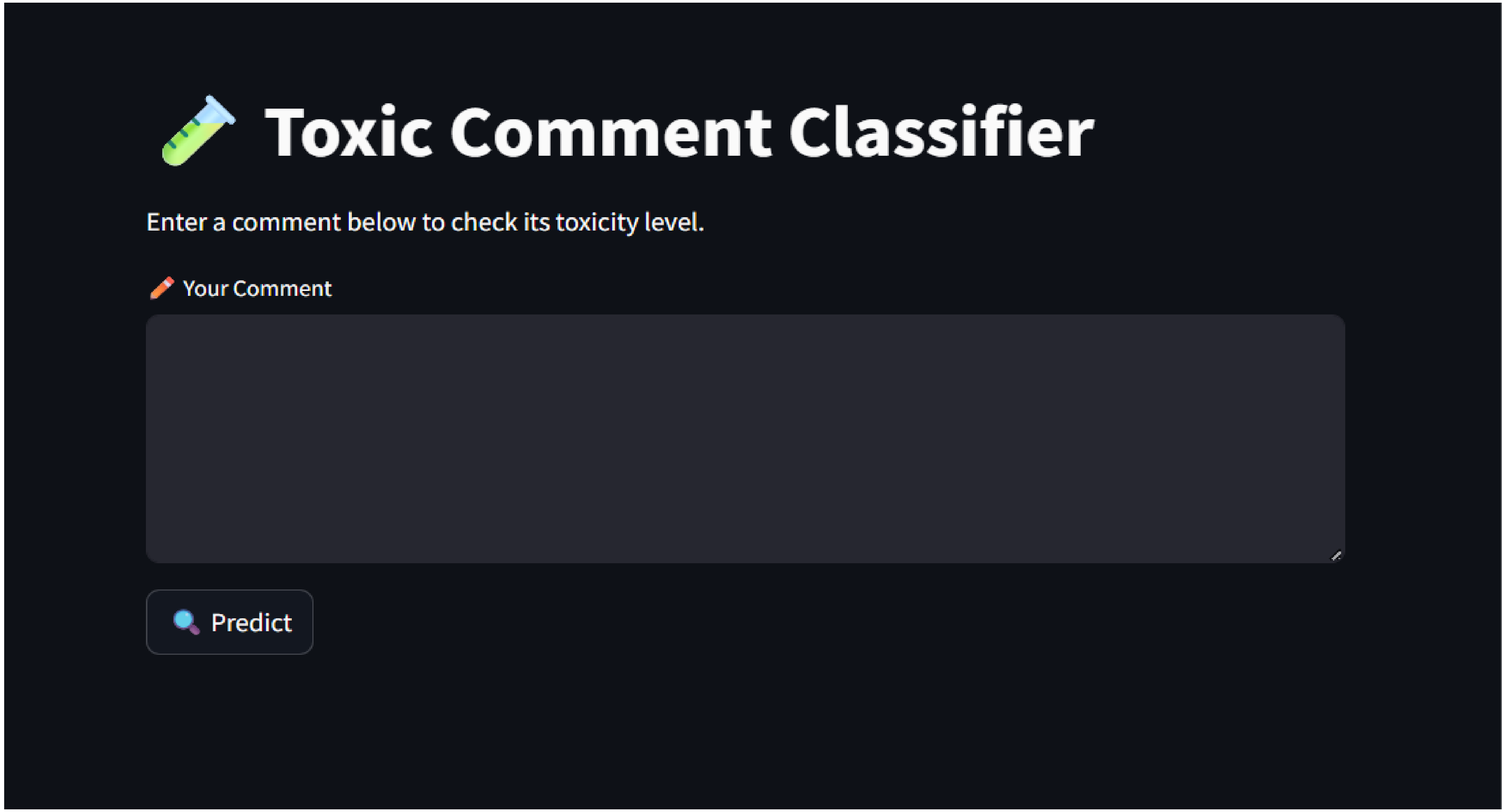
- Create a Dataset and DataLoader for the test set, similar to the training/
- validation sets, but without labels.

### 6.2 Generate Predictions

Run the model in evaluation mode on the test data loader

### 6.3 Format submission File

# Streamlit app



# Toxic Comment Classifier

Enter a comment below to check its toxicity level.