Western Norway University of Applied Sciences

Project report in DAT255 – Deep Learning Engineering

# Toxicity detection game DAT255

Date 25/04/2025

Candidate number 429

Githup repository: click here

I confirm that the work is self-prepared and that references/source references to all sources used in the work are provided, cf. Regulation relating to academic studies and examinations at the Western Norway University of Applied Sciences (HVL), §10.

# Problem description

The aim of this project is to design a deep learning model that can identify toxic or harmful content in text and this can be done to promote healthy communication, especially among children and teenagers. The idea was inspired by the growing need to moderate online conversations as toxic comments can cause harm and discomfort.

The final solution is planned to be used in a small interactive web application. In this application, users can input any sentence, and based on the model's prediction, the system displays an appropriate emoji (e.g., happy, sad, angry). It also provides a short explanation if the text is identified as toxic, such as obscene, threats, or insults along with a gentle educational message explaining why such language is harmful and how to express oneself in a better way.

This app aims to be used as a simple game or learning tool for young users to be able to grasp the effect of their words and how to be nice in their communication.

The reason why deep learning is suitable for this task is that it can capture the complex language patterns and contextual meaning which are often important in identifying toxicity. Other methods may not be able to handle such fine details. In particular, our use of Bidirectional LSTM layers enables the model to capture the meaning of words in both directions and thus improve the accuracy in detecting subtle toxicity.

Although there are other solutions like Google's Perspective API, this project is to develop a customized, lightweight and educational version that can be used by young users and can be used in the future. The model is saved after training and is ready for deployment in a web interface.

# Data

For this project, I used the Google Civil Comments dataset, available on Hugging Face. This dataset comprises over 2 million public comments made on news websites that were later labeled for various forms of toxicity by human raters. Each comment is labeled with binary values indicating whether it contains toxicity, insults, obscene language, identity attacks, and other harmful behaviours.

**Why this dataset?**

The Civil Comments dataset is perfect for training deep learning models for detecting toxic or harmful language in user-generated content. It is specifically useful for multi-label classification where each comment can have more than one type of toxicity. This is in line with the real life scenario of online moderation tasks.

**Pros:**

Large scale: Over 2 million comments, which helps prevent overfitting and enables deep models to generalize better.

Rich labeling: Multiple labels per comment covering different toxicity aspects.

Real-world data: Comments are taken from actual online discussions, providing realistic and diverse samples.

**Cons:**

Imbalanced classes: Some labels (like severe_toxicity) are much rarer than others, which can lead to biased predictions.

Language and grammar noise: The data includes informal writing, typos, and slang, which may affect performance.

Ethical concerns: The dataset contains sensitive and harmful content, so handling and usage must be done carefully.

**Alternative datasets:**

There are other datasets available for similar purposes, like the “Hate Speech and Offensive Language” dataset from Twitter, or smaller datasets focusing on specific domains (e.g. gaming or news comments). However, they are generally smaller in size or scope.

**Preprocessing and formatting:**

Before feeding the data into the model, several preprocessing steps were necessary:

Tokenization: Each comment was split into words using a tokenizer.

Sequence transformation: Words were converted into numerical sequences based on a vocabulary.

Padding: All sequences were made to have the same length by adding zeros at the end.

All labels were also converted into binary vectors to reflect the multi-label classification task.

Augmentation: Since the size of the dataset is big, I did not apply advanced text augmentation methods. However, some potential augmentations like synonym replacement or back-translation could be explored in future work to enhance generalization.

# Model implementation

### The first module:

I imported the data, explored it, and learned details about the data format and how it is organized. Then, I cleaned the data by converting all comment text to lowercase, removing spaces and special characters, and removing stop words. Then, I analysed the nature of the data, how many rows are in each category (column), and how many comments contain more than one label. In addition, I found out how many rows contain a label, and the percentage of label in a category. I also tried to analyse the correlation in the data.

I then researched and found that in natural language processing, the choice of text representation can significantly affect model performance. Traditional methods like Bag of Words *(BoW)* and *TF-IDF* (Term Frequency-Inverse Document Frequency) represent text as sparse vectors based on word counts or their weighted importance. While *BoW* captures simple frequency information, it ignores the context and semantics of words, often resulting in high-dimensional vectors. *TF-IDF* improves upon this by reducing the influence of common words and emphasizing rarer, more informative terms. However, both methods fail to capture the deeper relationships between words, such as similarity in meaning or syntactic roles.

To overcome these limitations, I chose to use learned word embeds via an embedding layer in my neural network. Embeddings map words to dense, low-dimensional vectors that preserve semantic relationships based on their usage in large corpora. This allows models to better understand context, handle synonyms, and generalize across similar patterns in language. For tasks involving nuanced understanding of text, such as multi-label toxic comment classification, embeddings provide a more powerful and context-aware foundation than *BoW* or *TF-IDF*. I used *TF-IDF* in a previous project in DAT158 and it was consuming a lot of resources, and I couldn't deploy because of that.

This decision is inspired by findings such as those by Mikolov et al. (2013), who demonstrated that word embeds significantly improve the performance of NLP models by capturing linguistic regularities and semantic relationships in vector space.

Of course we need to convert the text to numbers, so I used a tokenizer from *tensorflow.keras.preprocessing.text* and used the *texts_to_sequences()* method transforms the text data into a sequence of integers. Each word in the text is mapped to its integer index from the tokenizer's vocabulary. This step converts text into a form suitable for input into machine learning models. Then The *pad_sequences()* method ensures that all sequences have the same length by adding padding tokens (usually zeros) to the end of shorter sequences. The max length parameter defines the target length, and any sequence longer than this will be truncated. Due to my computer's limitations, I chose max length 100.

I then trained the Bidirectional LSTM model with a tokenizer set to length 100, *activation= sigmoid, optimizer=Adam, loss= binary_crossentropy*, and used EarlyStopping to avoid unhelpful overtraining. The results were not satisfactory, as

the model only predicted very well in the dominant class, toxicity, while its performance in the other classes was very weak. I also tried handling class imbalance with sample weights, and the results were also not much different from the previous ones, but still weak and unacceptable. The result is also that the model is very large, about half a gigabyte, which will make things difficult when deploying the model.

After analyzing the poor performance of the model in the Toxicity2 notebook, I investigated potential issues related to the text preprocessing pipeline. I found that using a more modern and integrated preprocessing approach such as *(TextVectorization)* from TensorFlow can lead to better results compared to traditional tokenization with manual preprocessing. The *(TextVectorization)* layer allows for built-in standardization, tokenization, vocabulary creation, and optional stop word filtering, all within the model pipeline, which improves consistency and performance.

Furthermore, I realized that removing stop words might harm the model's understanding of the text in this particular context. In toxicity detection tasks, stop words such as "you", "are", or "not" can carry significant semantic weight and contribute to the tone and intent of a sentence. Removing them might lead to a loss of important contextual or syntactic information.

Based on these observations and best practices in modern NLP, I will now use the *TextVectorization* layer without explicitly removing stop words, to preserve the natural structure of the sentences and allow the model to learn from the full linguistic context.

### The second model:

I faced many problems in training this model and due to the limited storage space available on my computer with 16 GB RAM, I tried on another computer that has more storage space and 8 GB RAM and I faced problems with the RAM and in addition to that some errors in the code and here I decided to start from scratch in the third model by using my device to prepare the code and set up everything and the training is only done on *Google Colab*.

### The third model:

Since the dataset contains a very large number of rows 1,800,000 and is also unbalanced, I first balanced the dataset by taking all rows with a threshold greater than or equal 0.5 and calculating their number, which is 148,547. I also took the same number of rows that did not contain any toxicity. This resulted in a balanced dataset with 297,094 rows.

I then cleaned the text in this dataset and saved it to the ***cleaned_balanced_civil_comments*** folder.

This way, the dataset I was working on was balanced and the text in it was cleaned.

I then converted the dataset to a DataFrame, checked for null values and empty text, and processed the data. Then I separated the texts from the labels.

I started creating a vectorizer with

MAX_WORDS=20000, *output_sequence_length*=300

I then adapted all the texts.

Since the dataset was still large and my computer's capabilities weren't strong, it was necessary to use a Data Pipeline *(tf.data.Dataset)* from TensorFlow. I used it to calculate the vectorizer, which saves a lot of RAMS and provides better speed

Then I divided the dataset into a 70% training section, a 20% evaluation section, and a 10% testing section.

**Building the Model**

1. Specify the same input size as the one selected in *sequence_length*.

2. Add an Embedding layer and use the MAX_WORDS value you specified previously.

3. Add a Bidirectional LSTM layer with activation tanh. I previously mentioned why we chose this model with activation tanh.

4. Add three fully connected feature extractor layers: 128, 256, and 128, with activation *relu*.

5. Add a final flatten layer with 7, since we have 7 labels, and an activation sigmoid, since we have a multi-label classification.

I used Adam optimizer and *binary_crossentropy* as the loss function, assuming each class is a binary bisection, and used precision, recall, and *auc* to track model performance.

Early Stopping was used to save unnecessary training time by waiting for 2 epochs, then the model was trained on 6 epochs and *validation_data* was used from the evaluation section in the dataset.

Unfortunately, the results were unsatisfactory: Precision: 0.0, Recall: 0.0, Accuracy: 0.99

**The fourth model:**

The poor results in Model 3 were the result of incorrect handling of the Data Pipeline, where the dataset had to be properly split and then tf.data.Dataset.from_tensor_slices applied.

So, this model is a fix for Model 3. *ModelCheckpoint* was also added because I increased the number of epochs to 10 and the tolerance to 4. The properly split data was saved to the *dataset_splits.npz* file.

### The fifth model:

To achieve the best results, the number of words increased from 20,000 to 40,000 and the *output_sequence_length* from 300 to 600.

The results for the entire model were good: Precision: 0.91, Recall: 0.70, Accuracy: 0.95.

However, the results were not good in class identification, and there was a clear weakness in class identification. Therefore, we specified the number of rows in the train section where the value >= 0.5 for each class and calculated the distribution of classes in the train section and the evaluation section.

Since we have rare classes with a small number of rows, which causes the model to ignore these classes, the solution may be to assign weights to these classes and train the model on these weights using class_weights_dict. Here, I used binary classification on each row, but this did not yield any better results or fix the problem.

### The sixth and final model:

To address the model's bias and its poor ability to predict different classes, I defined a function called *compute_class_weights*, which calculates a weight for each class based on its frequency. The function calculates the percentage of positive examples for each class in the training set, then assigns higher weights to rare classes by inverting their relative frequencies.

The weights were as follows:

'toxicity' 1.0, 'severe toxicity' 21.42, 'obscene' 5.86, 'threat' 14.43, 'insult' 1.17, 'identity attack' 5.55, 'sexual explicit' 16.01

These calculated weights are then stored in the *class_weights_dict* dictionary for use in a custom loss function called *weighted_binary_crossentropy*. This function adjusts the standard binary cross-entropy loss by multiplying it by the predefined weights.

The following were applied to the final model:

**Dropout**: Dropout is applied after the bidirectional LSTM layer to help prevent overfitting. A 30% dropout was used, which helped the model generalize.

**Attention**: Attention is applied to the LSTM output. Attention helps the model focus on specific parts of the input sequence that are most relevant to the prediction task. Instead of treating every word equally, attention calculates a weighted importance for each word, allowing the model to prioritize certain words over others based on their contribution to the final output.

**GlobalMaxPooling1D:** After the attention layer, GlobalMaxPooling1D is used to reduce the weighted output sequence to a single, fixed-size vector. Unlike mean pooling, which captures the overall context, max pooling selects the most prominent features by taking the maximum value across the time dimension. This helps the

model retain the most significant signals from the monitored features, resulting in a compact and informative representation.

# Evaluation

To effectively evaluate the success of the model and its implementation, a set of quantitative and qualitative metrics were used to measure the quality of predictions and the accuracy of model performance, along with a deeper analysis of the performance of each classification.

**Indices Used to Evaluate Performance**

Several standard metrics were used to evaluate the model, namely:

Precision: Measures the percentage of correct predictions out of all positive predictions.

Recall: Measures of the percentage of correct predictions out of all actual positive cases.

Binary Accuracy: The overall percentage of correct predictions.

Area Under Curve (AUC): This was used to estimate the quality of class separation.

Using a validation set, the results of these metrics were as follows:

Precision: 0.89, Recall: 0.71, Accuracy: 0.95

These values demonstrate the model's strong performance, particularly in terms of overall accuracy, meaning that the model successfully reduces the number of false predictions. The metrics were calculated globally as well as individually for each of the seven labels: toxicity, severe toxicity, obscene, threat, insult, identity attack, and sexual explicit.

To further validate the quality of the predictions, a confusion matrix was used in addition to an ROC curve plot. The results of the confusion matrix indicate a significant number of correct predictions for the negative class (356,312) and a significant number for the positive class (40,729), reflecting the model's ability to distinguish between different cases.

Performance for each class was also evaluated individually using precision, recall, and F1-score metrics, and the results were as follows:

| Label | Precision | Recall | F1-Score |
|---|---|---|---|
| **toxicity** | 0.96 | 0.76 | 0.85 |
| **severe_toxicity** | 0.00 | 0.00 | 0.00 |
| **obscene** | 0.68 | 0.62 | 0.65 |
| **threat** | 0.65 | 0.47 | 0.54 |
| **insult** | 0.89 | 0.70 | 0.79 |
| **identity_attack** | 0.65 | 0.57 | 0.61 |
| **sexual_explicit** | 0.62 | 0.63 | 0.62 |

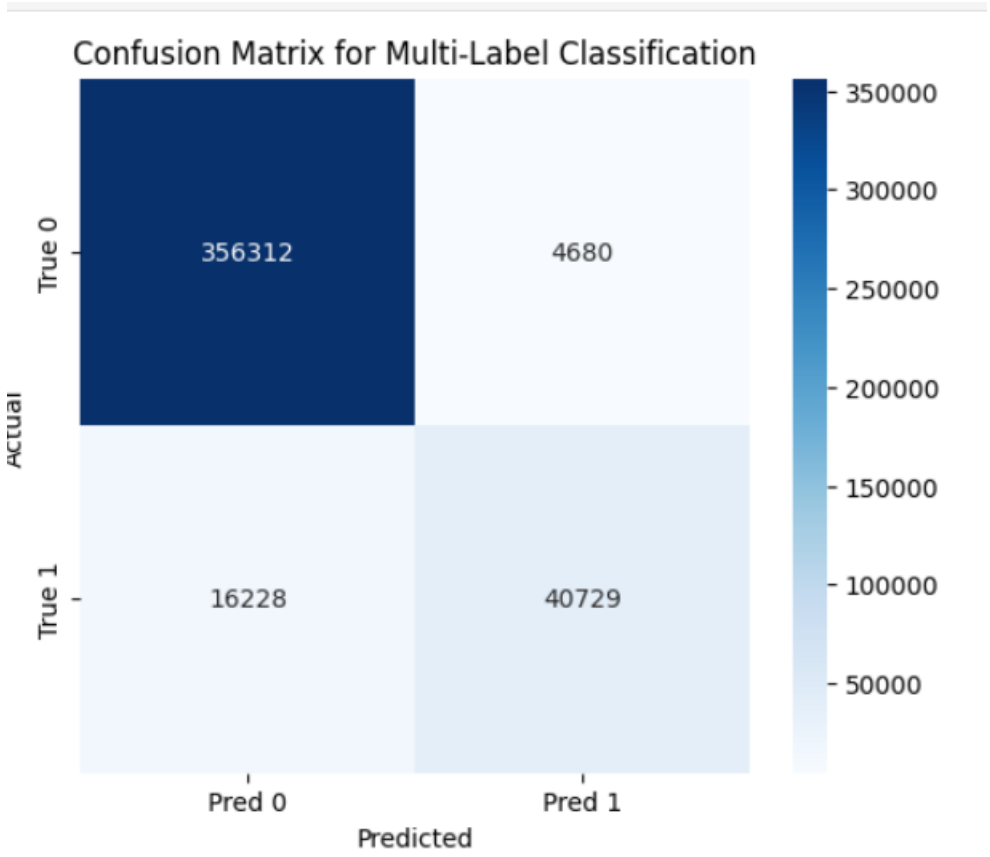
We notice that severe toxicity gives zero results, which is normal because the number of elements that have a value greater than or equal to 0.5 is only 12. It was necessary to remove these elements and this label, but I did not do so due to the lack of time and the long time required to retrain the model.

**Baseline performance**

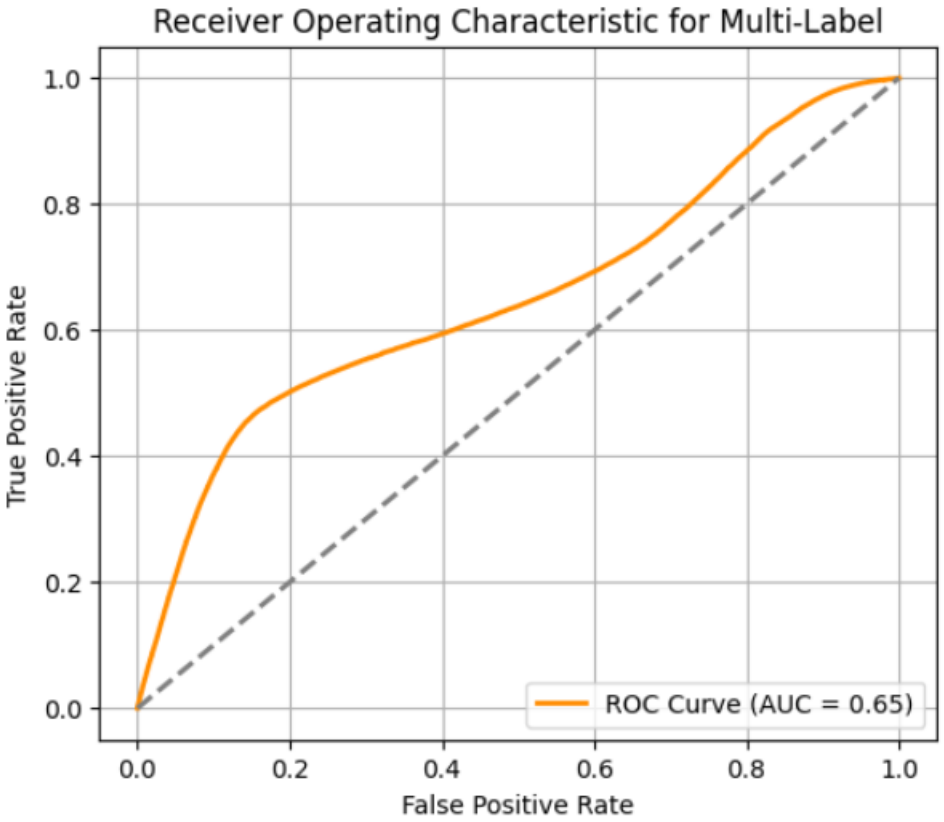
I made a simple baseline based on Random Guessing and it gave me Precision: 0.13, Recall: 0.49, Accuracy: 0.49 and the model give Precision: 0.89, Recall: 0.71, Accuracy: 0.95

**Explanation or interpretation techniques**

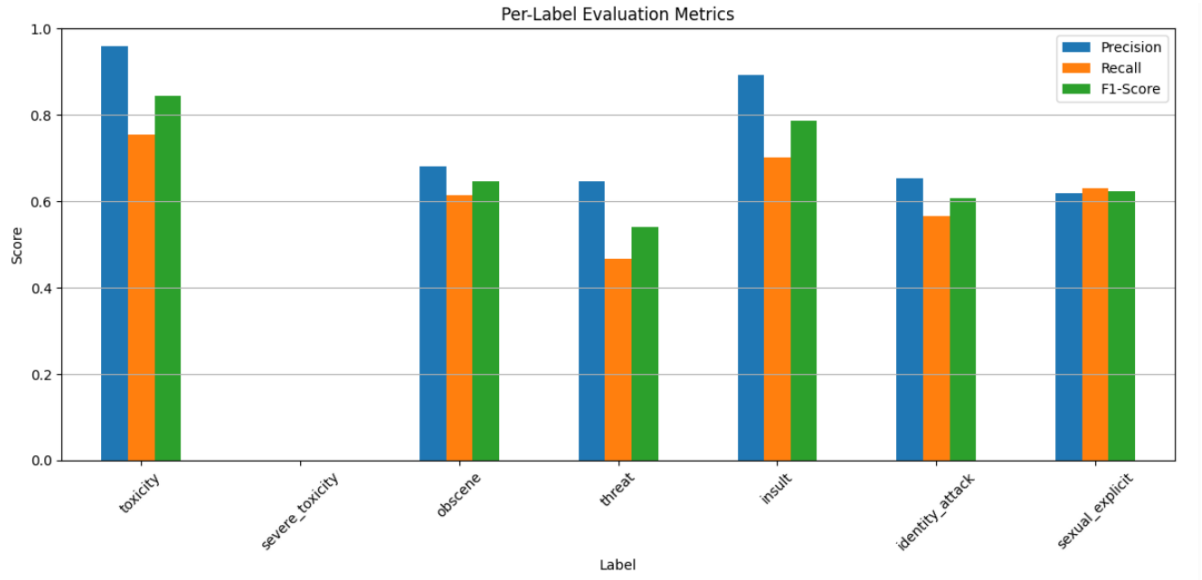
Confusion Matrix Heatmap: This visualization provides insights into the overall performance of the model, showing how often the model correctly or incorrectly predicted the labels.




ROC Curve with AUC: Enabled us to understand the model's discriminatory power across thresholds.




Per-label Metric Bar Charts: Visualized precision, recall, and F1-Score per category, helping us spot underperforming classes.




**Model architecture:**

We have already discussed how the model was developed from one model to another, and how I modified model architecture to achieve the best results but here I will mention the most important things I did in building the model architecture:

Bidirectional LSTM captured contextual information from both directions of the input sequence.

Attention Mechanism allowed the model to focus on the most informative parts of the text.

GlobalMaxPooling1D extracted the most prominent features from the attention outputs.

Dropout helped regularize the model and avoid overfitting.

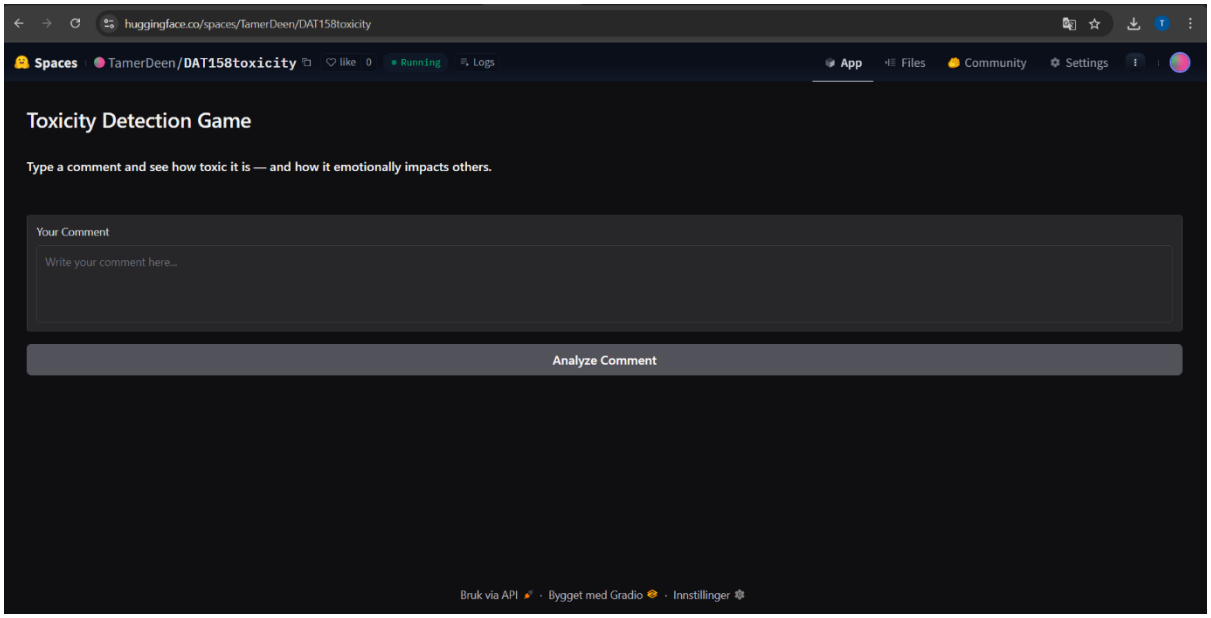
Dense Layers expanded the model’s capacity to learn high-level abstractions.

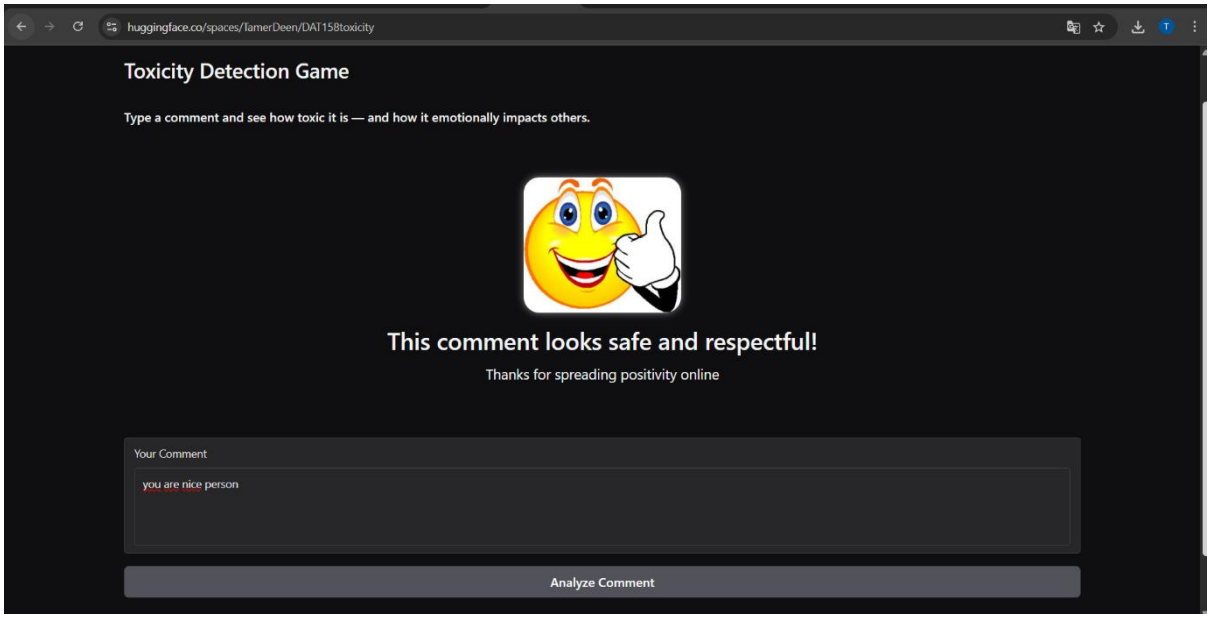
Additionally, I used vectorized inputs and efficient dataset pipelines with tf.data API to fully leverage TensorFlow’s performance optimizations.

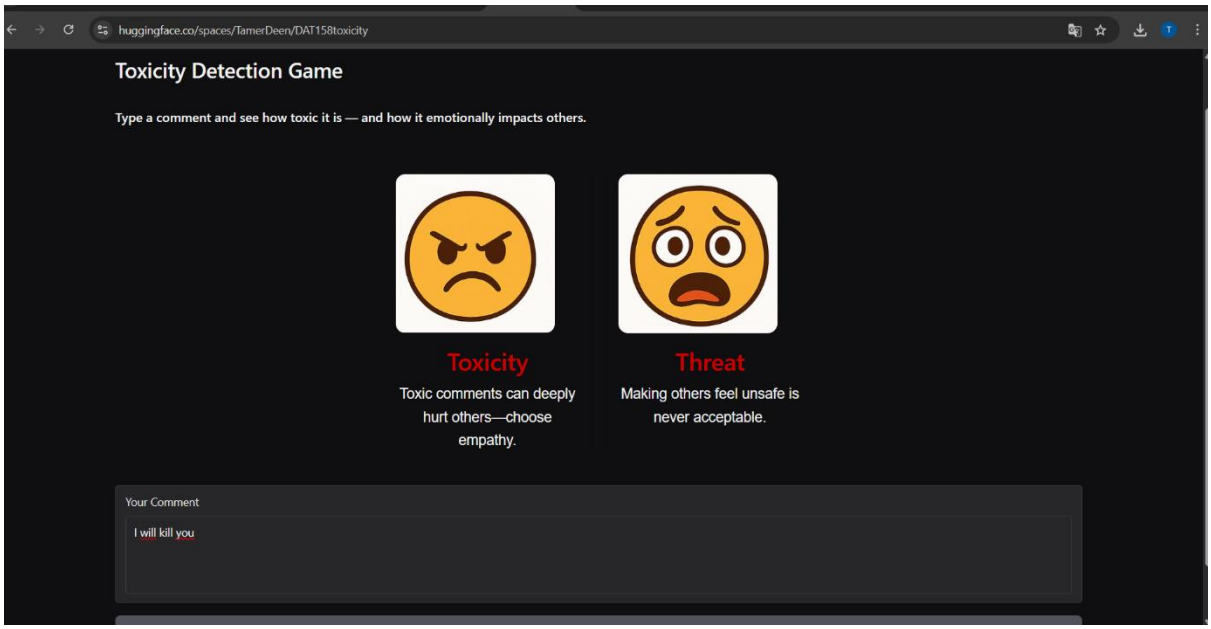
Although I didn't use any supplementary metrics relevant to the use or deployment of the model, I did consider the fact that I'm working on simple computers, so I chose a moderate number of parameters (dim = 32, LSTM = 32), making it deployable even on limited hardware. The 64-bit batch size and coaching also help reduce RAM usage and improve throughput during evaluation. The use of (TextVectorization) and batching with prefetching also ensures minimal input pipeline latency.

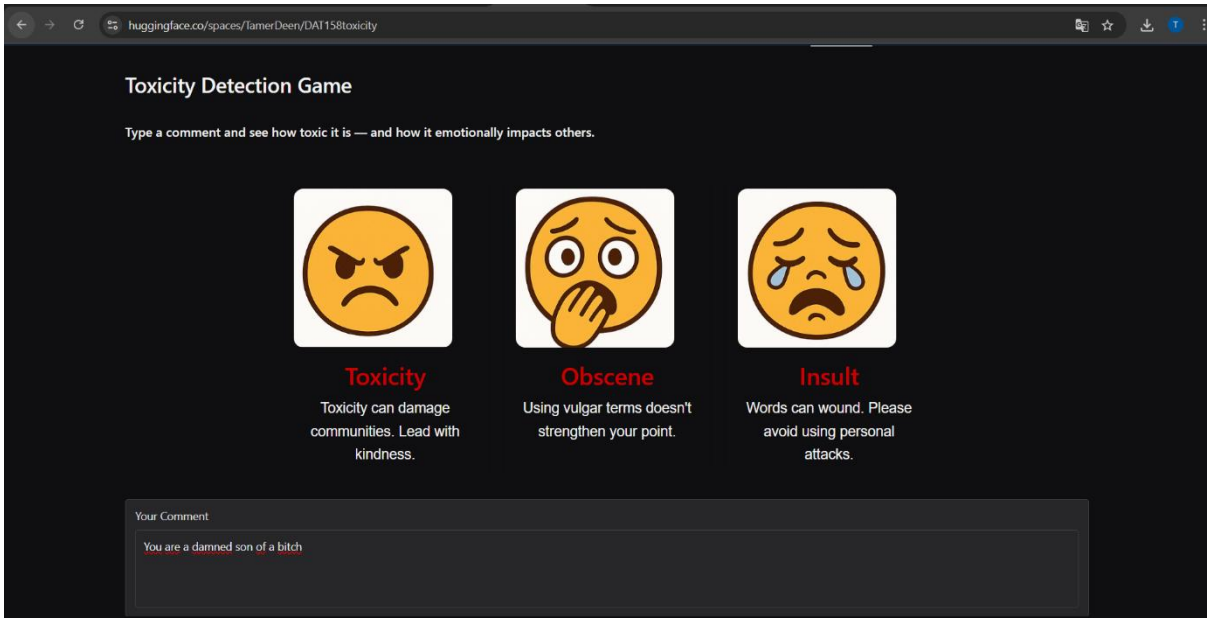
# Deployment

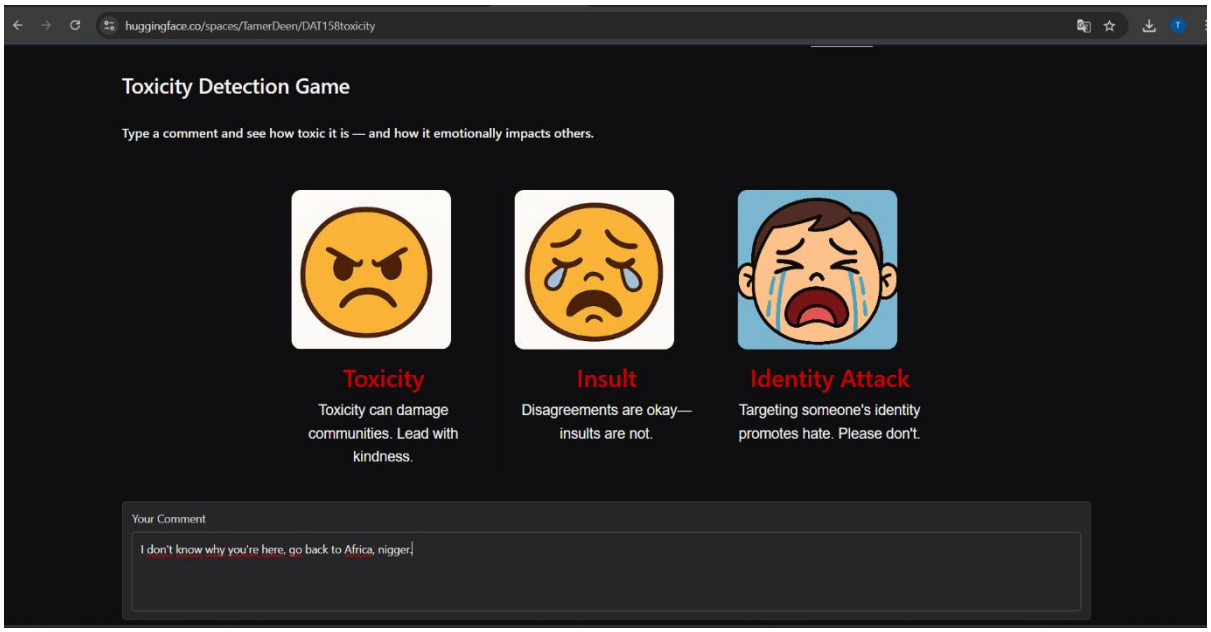
To see the demonstration please visit me Hugging Face Spaces


huggingface.co/spaces/TamerDeen/DAT158toxicity
Spaces
TamerDeen/DAT158toxicity
like 0
Running
Logs
App
Files
Community
Settings
Toxicity Detection Game
Type a comment and see how toxic it is — and how it emotionally impacts others.
Your Comment
Write your comment here...
Analyze Comment
Bruk via API
Bygget med Gradio
Innstillinger


huggingface.co/spaces/TamerDeen/DAT158toxicity
Toxicity Detection Game
Type a comment and see how toxic it is — and how it emotionally impacts others.
This comment looks safe and respectful!
Thanks for spreading positivity online
Your Comment
you are nice person
Analyze Comment


huggingface.co/spaces/TamerDeen/DAT158toxicity
Toxicity Detection Game
Type a comment and see how toxic it is — and how it emotionally impacts others.
Toxicity
Toxic comments can deeply hurt others—choose empathy.
Threat
Making others feel unsafe is never acceptable.
Your Comment
I will kill you




I used the Gradio library to create an interactive interface and application through which I could display and test the model. A simple game idea was implemented for teenagers, where the user enters text and clicks on "Text Analysis" or "Comment." The model analyzes the text and displays a cartoon image for each detected category, including text that does not contain any toxicity. The result of the text analysis is displayed with a cartoon image and simple awareness text as a form of fun and simple awareness. HTML and CSS were applied within the Gradio app to make the interface a little prettier, and the cartoons appear beautiful and simple.

I believe this framework, along with the improvements mentioned above, was sufficient for a simple and enjoyable presentation of the results.

**Expanding the scope of the project:**

I had started adding the Norwegian language, but I was unable to complete it due to time constraints. I thought it would be possible, for example, to use Google Translate to create a new database from the same old database and then retrain the model on the Norwegian database.

Or use the same model now in English, and when entering text, it is translated from Norwegian to English and applied to the same existing model, of course with an interface that supports the Norwegian language.

It is also possible to develop the images, making them, for example, animated, and to increase the number of images and awareness messages. It's also possible to use a more professional interface, for example, using Flask. Among other things, it's possible to use a lower threshold, such as 0.5, for example 0.3. This means a larger number of elements will be available than the current number, allowing the model to train more efficiently. However, this will require more RAM and greater computing power.

# Conclusion

The model was tested by an American native English speaker, and the model demonstrated a very good ability to identify all categories well, except for the severe toxicity category. The reason was previously mentioned.

During model training and transitioning from one model to another, or more precisely, developing the model properly, we relied heavily on four main metrics: accuracy, precision, recall, and F1-score, especially in the context of imbalanced classes. Since some categories (e.g., severe toxicity or threat), these metrics helped us better understand the model's performance and work to improve these metrics until we achieved satisfactory results.

**Was Deep Learning a Good Solution?**

Last semester, I worked on a similar task in text classification. Although it was originally a simpler task, as it was binary classification, I noticed a difference in the results, as well as the ability of deep learning to better understand the context of the text and to deploy the model more effectively. So, I can confidently say that deep learning provides deep and powerful solutions, and I think this project is proof of that, as a model was developed without using pre-trained models such as BERT or RoBERTa, and the results are still good and acceptable.

# References

TensorFlow TextVectorization documentation – https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space, https://arxiv.org/abs/1301.3781). arXiv preprint arXiv:1301.3781.