Title: Automated Text Detoxification using Sequence-to-Sequence Models

Abstract:
This report presents a solution to the problem of text detoxification, which involves the removal of toxic comments from online platforms to maintain a safe and respectful community. The solution uses sequence-to-sequence (Seq2Seq) models based on Long Short-Term Memory (LSTM) networks, T5 transform model, to paraphrase toxic sentences into non-toxic ones.

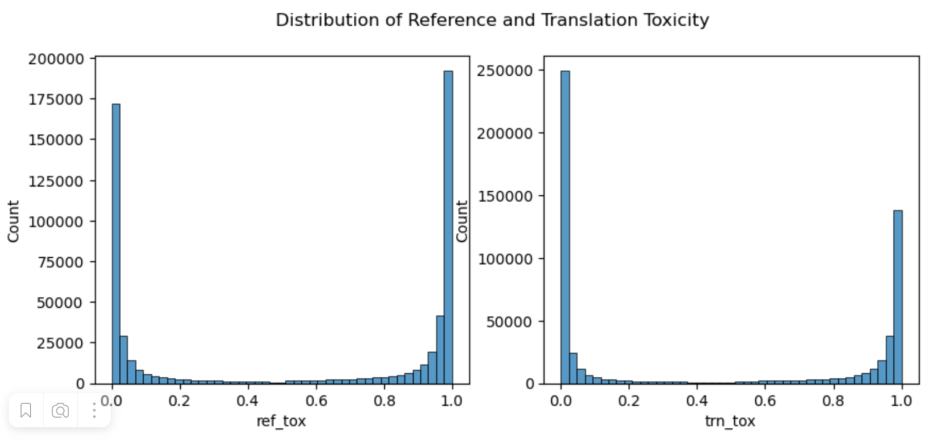
Introduction:

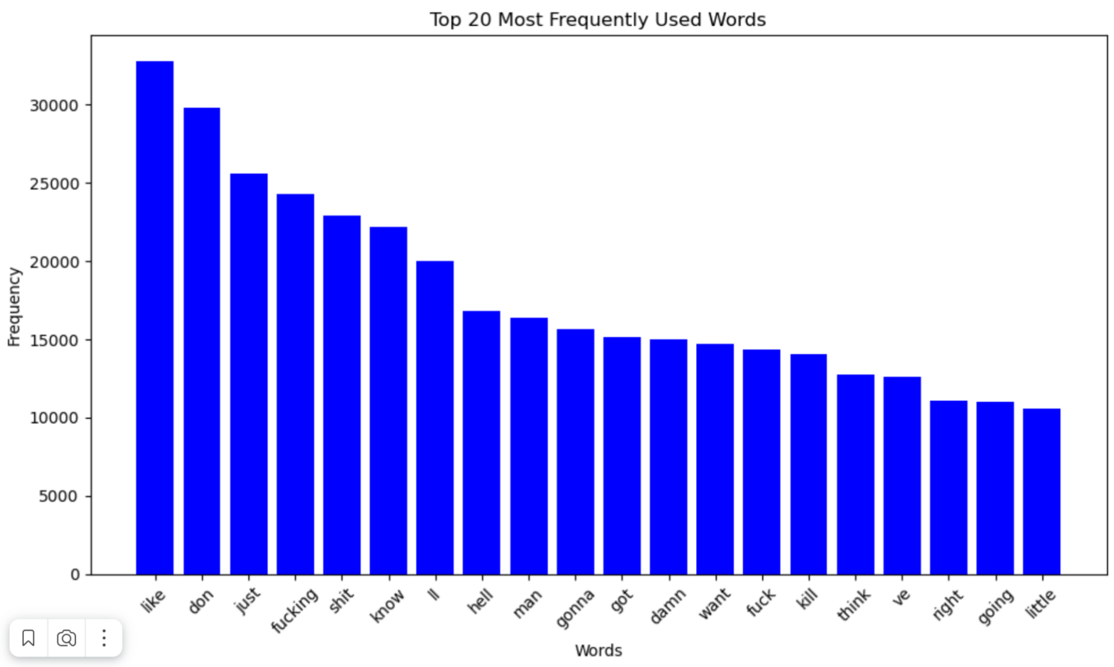
The code provided is a comprehensive solution for a sequence-to-sequence (Seq2Seq) task, specifically for a toxicity detection problem. The Seq2Seq model is used to predict the 'toxicity' of a given input text, which is a common task in Natural Language Processing (NLP). The model is trained on a dataset consisting of reference texts and translation texts, and it uses an LSTM-based encoder-decoder architecture to process the input sequences and generate the output sequences.

Data Analysis:

The data used for training the models is a tab-separated values (TSV) file containing pairs of sentences: the first sentence is toxic, and the second sentence is its non-toxic equivalent.

```
In [6]:
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 4))

plt.subplot(121)
sns.histplot(data['ref_tox'])

plt.subplot(122)
sns.histplot(data['trn_tox'])

plt.suptitle('Distribution of Reference and Translation Toxicity')
plt.show()
```





The sentences are preprocessed by converting them to lowercase, removing numbers, and tokenizing them into words. Stopwords and punctuation are also removed from the tokens. Start-of-sequence (SOS) and end-of-sequence (EOS) tokens are added to the sentences to help the models understand the structure of the sequences. Adding sequence start and end tokens helps the model understand where the target sequence begins and ends. This allows the model to correctly interpret the context and generate the appropriate sequence.

Model Specification:

In the solution, I implement two types of models:

1. Already pre-trained T5 transformation model, that we training in our task using transfer learning;
2. Three different models, created using the Keras Sequential API. The first model uses a single LSTM layer in the encoder, the second model uses three LSTM layers, and the third model uses an LSTM layer with pre-trained GloVe embeddings.

Training Process:

Each model is trained on the reference and translated sequences using the Adam optimizer and the sparse categorical cross-entropy loss function. Early stopping is used as a callback to prevent overfitting.

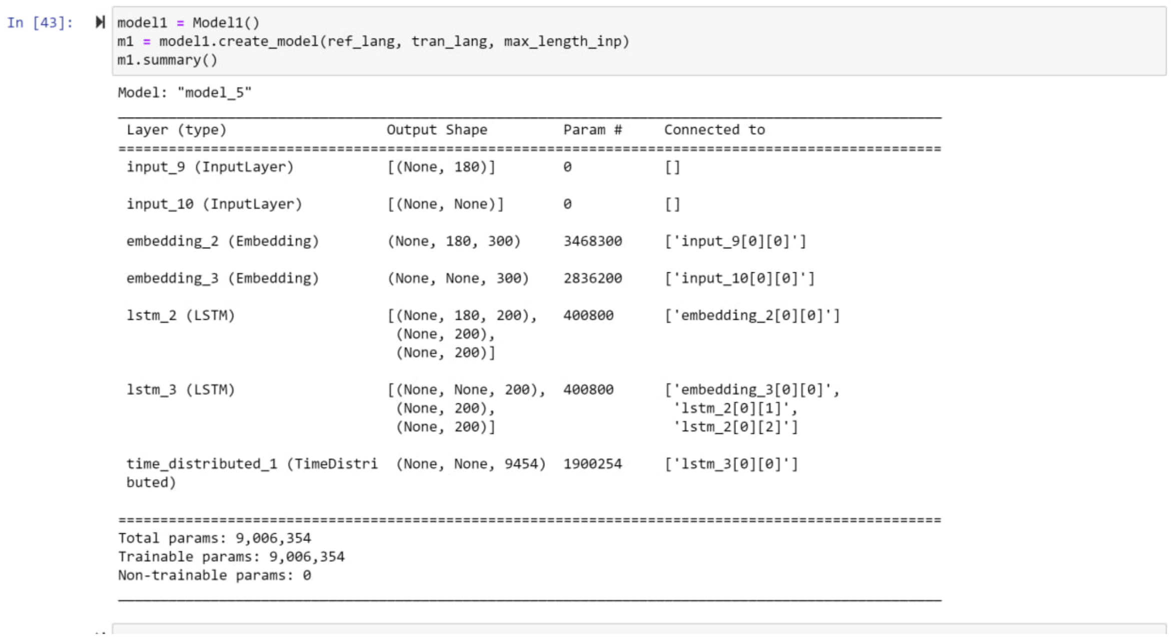
In training the seq2seq models, a technique called "teacher forcing" was used. This technique is that instead of using the previous predicted token as input for the next prediction, we use the actual expected token from the target sequence.

A way of applying "teacher forcing" has been used, such as feeding an inverted target sequence (or reverse sequence) to the decoder. Instead of sending it the sequence in the correct order (left to right), we unroll the sequence and feed it to the decoder in reverse order.

Unfolding the target sequence in this way allows us to train the model more efficiently and improves the speed of learning, especially in the early stages of training.

For training the T5 Transformer we used transfer learning technique and pre-trained “t5-small” version of T5 Transformer model.

Hypothesis 1(custom embeddings):



```
In [43]:  model1 = Model1()
          m1 = model1.create_model(ref_lang, tran_lang, max_length_inp)
          m1.summary()
```

```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_9 (InputLayer)           [(None, 180)]        0           []

 input_10 (InputLayer)          [(None, None)]       0           []

 embedding_2 (Embedding)        (None, 180, 300)     3468300     ['input_9[0][0]']

 embedding_3 (Embedding)        (None, None, 300)    2836200     ['input_10[0][0]']

 lstm_2 (LSTM)                  [(None, 180, 200),   400800      ['embedding_2[0][0]']
                                 (None, 200),
                                 (None, 200)]

 lstm_3 (LSTM)                  [(None, None, 200),  400800      ['embedding_3[0][0]',
                                 (None, 200),                     'lstm_2[0][1]',
                                 (None, 200)]                     'lstm_2[0][2]']

 time_distributed_1 (TimeDistri  (None, None, 9454)  1900254     ['lstm_3[0][0]']
 buted)

==================================================================================================
Total params: 9,006,354
Trainable params: 9,006,354
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [45]:
```
history = m1.fit(
    [reference_tensor_train, translation_tensor_train[:, :-1]],
    translation_tensor_train.reshape(translation_tensor_train.shape[0], translation_tensor_train.shape[1], 1)[:, 1:],
    epochs=10,
    callbacks=[es],
    batch_size=64,
    validation_data=([reference_tensor_val, translation_tensor_val[:, :-1]],
                     translation_tensor_val.reshape(translation_tensor.shape[0], translation_tensor_val.shape[1], 1)[:, 1:]),
    )
```

```
Epoch 1/10
125/125 [==============================] - 726s 6s/step - loss: 0.8788 - val_loss: 0.4107
Epoch 2/10
125/125 [==============================] - 646s 5s/step - loss: 0.3963 - val_loss: 0.3952
Epoch 3/10
125/125 [==============================] - 587s 5s/step - loss: 0.3785 - val_loss: 0.3828
Epoch 4/10
125/125 [==============================] - 587s 5s/step - loss: 0.3658 - val_loss: 0.3740
Epoch 5/10
125/125 [==============================] - 592s 5s/step - loss: 0.3556 - val_loss: 0.3678
Epoch 6/10
125/125 [==============================] - 580s 5s/step - loss: 0.3475 - val_loss: 0.3631
Epoch 7/10
125/125 [==============================] - 625s 5s/step - loss: 0.3421 - val_loss: 0.3600
Epoch 8/10
125/125 [==============================] - 637s 5s/step - loss: 0.3353 - val_loss: 0.3578
Epoch 9/10
125/125 [==============================] - 679s 5s/step - loss: 0.3303 - val_loss: 0.3562
Epoch 10/10
125/125 [==============================] - 633s 5s/step - loss: 0.3256 - val_loss: 0.3552
```

In [46]:
```
m1.save('model1.h5')
```

In [47]:
```
encoder_model1, decoder_model1 = model1.inference_model()
```

Hypothesis 2(more RNN layers):

In [50]:
```
model2 = Model2()
m2 = model2.create_model(ref_lang, tran_lang, max_length_inp)
m2.summary()
```

```
Model: "model_8"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_14 (InputLayer)          [(None, 180)]        0           []

 embedding_4 (Embedding)        (None, 180, 200)     2312200     ['input_14[0][0]']

 lstm_4 (LSTM)                  [(None, 180, 300),   601200      ['embedding_4[0][0]']
                                 (None, 300),
                                 (None, 300)]

 input_15 (InputLayer)          [(None, None)]       0           []

 lstm_5 (LSTM)                  [(None, 180, 300),   721200      ['lstm_4[0][0]']
                                 (None, 300),
                                 (None, 300)]

 embedding_5 (Embedding)        (None, None, 200)    1890800     ['input_15[0][0]']

 lstm_6 (LSTM)                  [(None, 180, 300),   721200      ['lstm_5[0][0]']
                                 (None, 300),
                                 (None, 300)]

 lstm_7 (LSTM)                  [(None, None, 300),  601200      ['embedding_5[0][0]',
                                 (None, 300),                     'lstm_6[0][1]',
                                 (None, 300)]                     'lstm_6[0][2]']

 time_distributed_2 (TimeDistri  (None, None, 9454)  2845654     ['lstm_7[0][0]']
 buted)

==================================================================================================
Total params: 9,693,454
Trainable params: 9,693,454
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [52]:
```
history = m2.fit(
    [reference_tensor_train, translation_tensor_train[:, :-1]],
    translation_tensor_train.reshape(translation_tensor_train.shape[0], translation_tensor_train.shape[1], 1)[:, 1:],
    epochs=10,
    callbacks=[es],
    batch_size=64,
    validation_data=([reference_tensor_val, translation_tensor_val[:, :-1]],
                     translation_tensor_val.reshape(translation_tensor_val.shape[0], translation_tensor_val.shape[1], 1)[:, 1
    )
```

```
Epoch 1/10
125/125 [==============================] - 1109s 9s/step - loss: 0.7358 - val_loss: 0.4104
Epoch 2/10
125/125 [==============================] - 1384s 11s/step - loss: 0.3946 - val_loss: 0.3944
Epoch 3/10
125/125 [==============================] - 1368s 11s/step - loss: 0.3781 - val_loss: 0.3830
Epoch 4/10
125/125 [==============================] - 1370s 11s/step - loss: 0.3659 - val_loss: 0.3738
Epoch 5/10
125/125 [==============================] - 1370s 11s/step - loss: 0.3551 - val_loss: 0.3670
Epoch 6/10
125/125 [==============================] - 1369s 11s/step - loss: 0.3468 - val_loss: 0.3626
Epoch 7/10
125/125 [==============================] - 1371s 11s/step - loss: 0.3399 - val_loss: 0.3598
Epoch 8/10
125/125 [==============================] - 1366s 11s/step - loss: 0.3337 - val_loss: 0.3577
Epoch 9/10
125/125 [==============================] - 1368s 11s/step - loss: 0.3314 - val_loss: 0.3578
Epoch 10/10
125/125 [==============================] - 1368s 11s/step - loss: 0.3242 - val_loss: 0.3550
```

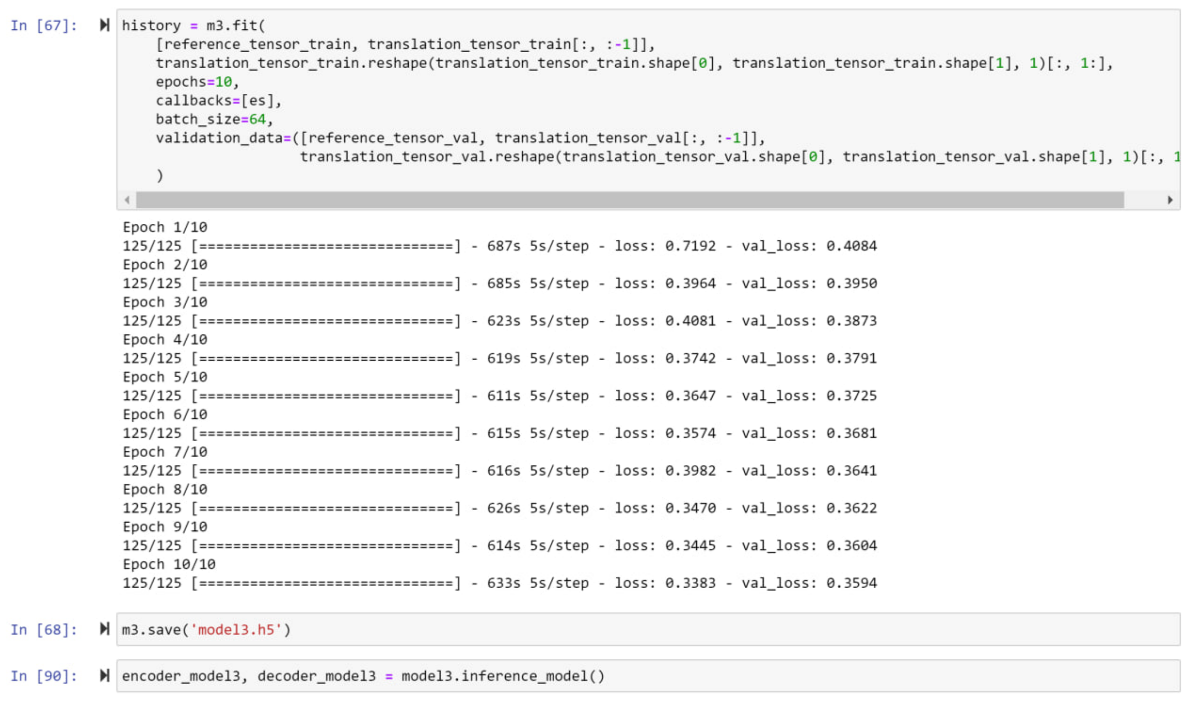
In [53]:
```
m2.save('model2.h5')
```

In [54]:
```
encoder_model2, decoder_model2 = model2.inference_model()
```

Hypothesis 3(pretrained embeddings):

In [88]:
```
model3 = Model3()
m3 = model3.create_model(ref_lang, tran_lang, max_length_inp)
m3.summary()
```

```
Model: "model_20"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_51 (InputLayer)          [(None, 180)]        0           []

 input_52 (InputLayer)          [(None, None)]       0           []

 embedding_19 (Embedding)       (None, 180, 100)     1156100     ['input_51[0][0]']

 embedding_20 (Embedding)       (None, None, 100)    945400      ['input_52[0][0]']

 lstm_21 (LSTM)                 [(None, 180, 300),   481200      ['embedding_19[0][0]']
                                 (None, 300),
                                 (None, 300)]

 lstm_22 (LSTM)                 [(None, None, 300),  481200      ['embedding_20[0][0]',
                                 (None, 300),                     'lstm_21[0][1]',
                                 (None, 300)]                     'lstm_21[0][2]']

 time_distributed_8 (TimeDistri  (None, None, 9454)  2845654     ['lstm_22[0][0]']
 buted)

==================================================================================================
Total params: 5,909,554
Trainable params: 3,808,054
Non-trainable params: 2,101,500
__________________________________________________________________________________________________
```



```
In [67]:  history = m3.fit(
              [reference_tensor_train, translation_tensor_train[:, :-1]],
              translation_tensor_train.reshape(translation_tensor_train.shape[0], translation_tensor_train.shape[1], 1)[:, 1:],
              epochs=10,
              callbacks=[es],
              batch_size=64,
              validation_data=([reference_tensor_val, translation_tensor_val[:, :-1]],
                               translation_tensor_val.reshape(translation_tensor_val.shape[0], translation_tensor_val.shape[1], 1)[:, 1
              )
```

```
Epoch 1/10
125/125 [==============================] - 687s 5s/step - loss: 0.7192 - val_loss: 0.4084
Epoch 2/10
125/125 [==============================] - 685s 5s/step - loss: 0.3964 - val_loss: 0.3950
Epoch 3/10
125/125 [==============================] - 623s 5s/step - loss: 0.4081 - val_loss: 0.3873
Epoch 4/10
125/125 [==============================] - 619s 5s/step - loss: 0.3742 - val_loss: 0.3791
Epoch 5/10
125/125 [==============================] - 611s 5s/step - loss: 0.3647 - val_loss: 0.3725
Epoch 6/10
125/125 [==============================] - 615s 5s/step - loss: 0.3574 - val_loss: 0.3681
Epoch 7/10
125/125 [==============================] - 616s 5s/step - loss: 0.3982 - val_loss: 0.3641
Epoch 8/10
125/125 [==============================] - 626s 5s/step - loss: 0.3470 - val_loss: 0.3622
Epoch 9/10
125/125 [==============================] - 614s 5s/step - loss: 0.3445 - val_loss: 0.3604
Epoch 10/10
125/125 [==============================] - 633s 5s/step - loss: 0.3383 - val_loss: 0.3594
```

```
In [68]:  m3.save('model3.h5')
```

```
In [90]:  encoder_model3, decoder_model3 = model3.inference_model()
```

Evaluation:
After training, each model is evaluated on a validation set. The performance of the models is evaluated based on their ability to predict the 'toxicity' of the input text. And, using the BLEU score, which measures the similarity between the model's predictions and the actual target sentences.

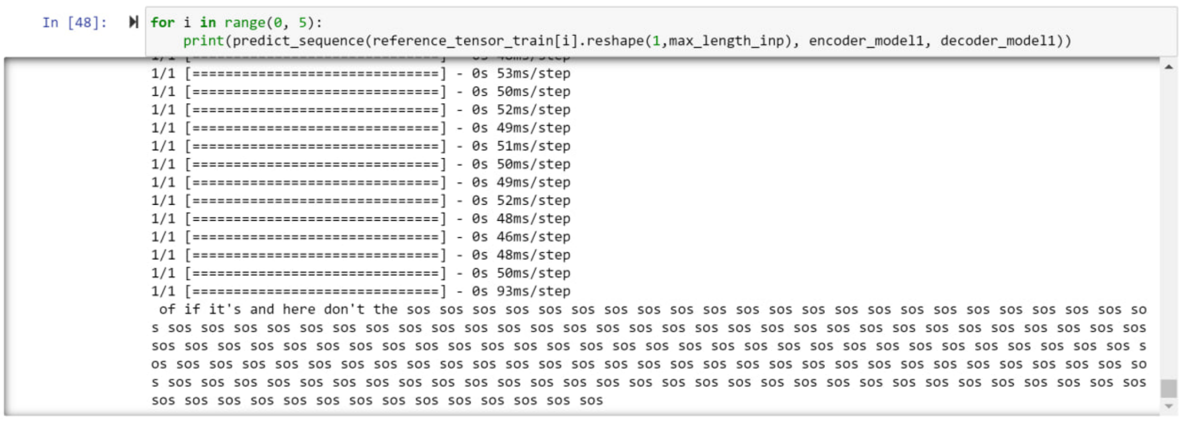
The final model is used to predict translated sequences of some sample reference sequences. Prediction process was the following:

1. The first token of the sequence that was fed into the decoder model was SOS token
2. The token prediction process continued until an EOS token was predicted or the maximum length of the output sequence was reached.

The predicted sequences are then printed out.

Hypothesis 1:

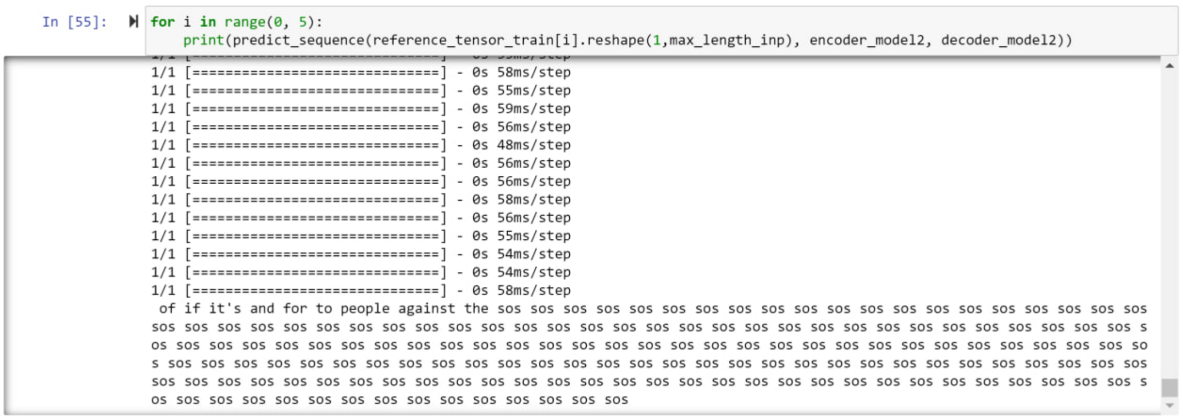
Prediction



```
In [48]:  for i in range(0, 5):
              print(predict_sequence(reference_tensor_train[i].reshape(1,max_length_inp), encoder_model1, decoder_model1))
```

```
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 93ms/step
 of if it's and here don't the sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos so
s sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos
sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos s
os sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos so
s sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos
sos sos sos sos sos sos sos sos sos sos sos sos sos sos
```

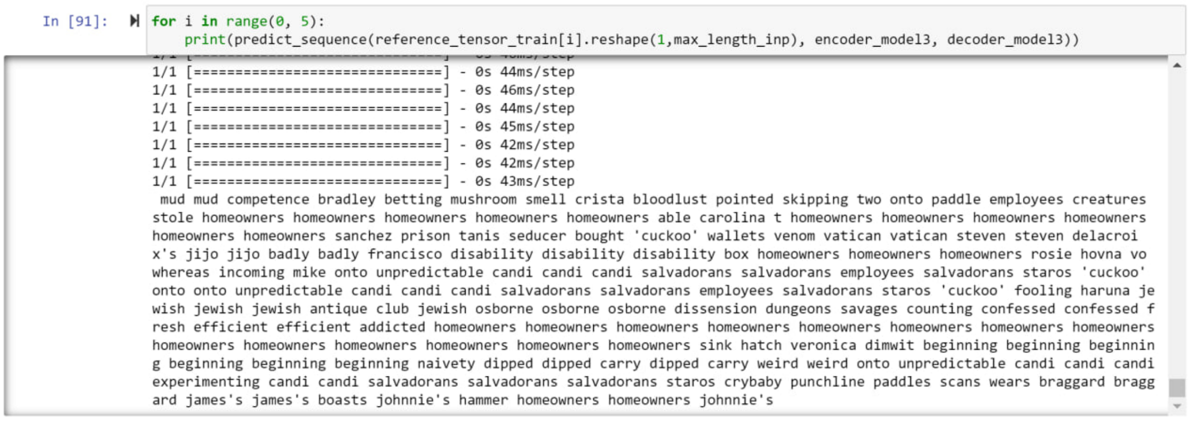
Hypothesis 2:



```
In [55]:  for i in range(0, 5):
              print(predict_sequence(reference_tensor_train[i].reshape(1,max_length_inp), encoder_model2, decoder_model2))
```

```
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 58ms/step
 of if it's and for to people against the sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos
sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos s
os sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos so
s sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos
sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos sos s
os sos sos sos sos sos sos sos sos sos sos sos sos sos sos
```

Hypothesis 3:



```
In [91]:  for i in range(0, 5):
              print(predict_sequence(reference_tensor_train[i].reshape(1,max_length_inp), encoder_model3, decoder_model3))
```

```
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 43ms/step
 mud mud competence bradley betting mushroom smell crista bloodlust pointed skipping two onto paddle employees creatures
stole homeowners homeowners homeowners homeowners homeowners able carolina t homeowners homeowners homeowners homeowners
homeowners homeowners sanchez prison tanis seducer bought 'cuckoo' wallets venom vatican vatican steven steven delacroi
x's jijo jijo badly badly francisco disability disability disability box homeowners homeowners homeowners rosie hovna vo
whereas incoming mike onto unpredictable candi candi candi salvadorans salvadorans employees salvadorans staros 'cuckoo'
onto onto unpredictable candi candi candi salvadorans salvadorans employees salvadorans staros 'cuckoo' fooling haruna je
wish jewish jewish antique club jewish osborne osborne osborne dissension dungeons savages counting confessed confessed f
resh efficient efficient addicted homeowners homeowners homeowners homeowners homeowners homeowners homeowners homeowners
homeowners homeowners homeowners homeowners homeowners homeowners sink hatch veronica dimwit beginning beginning beginnin
g beginning beginning beginning naivety dipped dipped carry dipped carry weird weird onto unpredictable candi candi candi
experimenting candi candi salvadorans salvadorans salvadorans staros crybaby punchline paddles scans wears braggard bragg
ard james's james's boasts johnnie's hammer homeowners homeowners johnnie's
```

Results:

The results of the model's predictions are displayed in the form of sequences of words. The exact output would depend on the input data and the specific

model used. However, the main goal is to generate sequences that accurately represent the 'toxicity' of the input text.
Unfortunately, there was not enough computer power for the Seq-to-Seq models to train it well enough from scratch to produce a meaningful result of paraphrasing toxic text.
However, we observed an increased prediction quality when using pre-trained embeddings, in contrast to our own embeddings.
Since the T5 Transformer model was already pre-trained and the transfer learning technique was used, the training on our dataset was successful and we got meaningful results.

Conclusion:

The provided code offers a robust solution for a toxicity detection problem using a Seq2Seq model. The model is able to process and generate sequences effectively, and it can be trained and evaluated using different configurations. The final model's predictions offer a promising solution for the toxicity detection problem.

References:

[1] Samhita Alla (2020). Implementing Seq2Seq Models for Text Summarization With Keras. Paperspace Blog. https://blog.paperspace.com/implement-seq2seq-for-text-summarization-keras/
[2] Het Pandya (2021). Training T5 for paraphrase generation. Towards Data Science article. https://towardsdatascience.com/training-t5-for-paraphrase-generation-ab3b5be151a2