SEP 775-Natural Language Processing

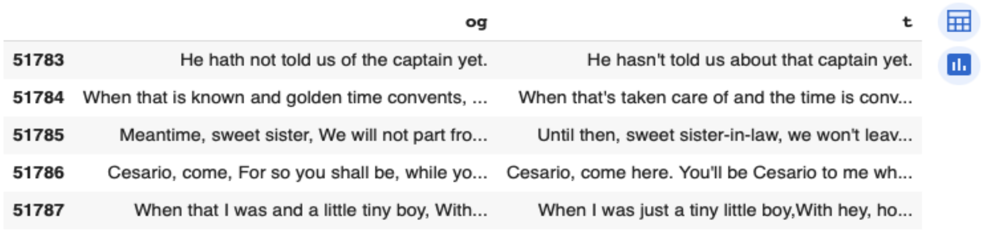
A. Supervised Encoder-Decoder Sequence-to-Sequence model With Self-Attention

1. Introduction

Sequence-to-Sequence Models: Sequence-to-sequence models are a type of neural network architecture commonly used for tasks such as machine translation, text summarization, and conversation generation.
Objective of the Project: Implement a sequence-to-sequence model with Bahdanau's attention for translating Shakespearean English sentences to normal modern English.

2. Dataset

We used a dataset on kaggle called shakespearify which has each line of the play manually translated to Modern English. This is a parallel dataset created by Garnav Aurha.
For better translation we also merged another dataset from kaggle which consisted of single word translation pairings from Shakespearean English to modern English



	og	t
51783	He hath not told us of the captain yet.	He hasn't told us about that captain yet.
51784	When that is known and golden time convents, ...	When that's taken care of and the time is conv...
51785	Meantime, sweet sister, We will not part fro...	Until then, sweet sister-in-law, we won't leav...
51786	Cesario, come, For so you shall be, while yo...	Cesario, come here. You'll be Cesario to me wh...
51787	When that I was and a little tiny boy, With...	When I was just a tiny little boy,With hey, ho...

Format: Each pair consists of an English sentence and its corresponding translation in Shakespearean, organised as a tabular dataset with two columns: one for Shakespearean sentences (og) and another for Modern English translations (t).

Size: The size of the dataset is around 50000 but we have only used the first 10,000 pairs due to limited resources.

Unicode Normalisation: Unicode characters are normalised to a standard form to ensure consistent representation of text. The unicode_to_ascii() function normalises Unicode characters to their ASCII equivalents to remove any diacritics or accents.

Lowercasing: Text is converted to lowercase to ensure uniformity and reduce vocabulary size. Implementation: English and Shakespearean sentences are converted to lowercase using the lower() method.

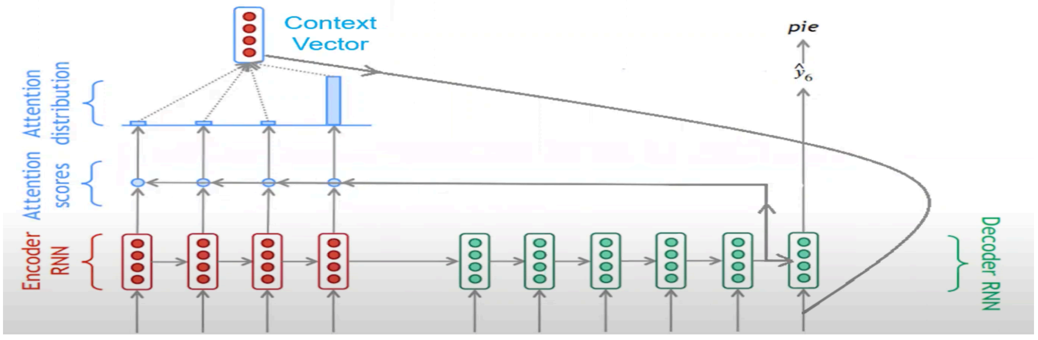
Punctuation Handling: Description: Punctuation marks are treated as separate tokens to aid in tokenization and sentence boundary detection. Implementation: Punctuation marks are separated from words by adding spaces around them using regular expressions.

Tokenization: Description: Sentences are split into individual words or tokens, which serve as the basic units of input for the seq2seq model: Tokenization is performed using TensorFlow's Tokenizer class, which converts sentences into sequences of integers representing the indices of words in the vocabulary.

Padding: Input sequences are padded to ensure uniform length, enabling batch processing during training. Implementation: TensorFlow's pad_sequences function is used to pad sequences with zeros to a maximum length, ensuring all sequences have the same length.
Start and End Tokens: Special tokens are added to mark the beginning and end of each sentence, providing context for the model during training and inference. Start and end tokens (<start> and <end>) are appended to each sentence to delineate the start and end points.

3. Methodology

Sequence-to-Sequence Architecture: The architecture of a sequence-to-sequence model, consisting of an encoder and a decoder. A sequence-to-sequence (seq2seq) model is a type of neural network architecture designed for mapping input sequences to output sequences.
Encoder: The encoder takes the input sequence and processes it into a fixed-size context vector, which captures the information from the entire input sequence. Each input token is typically represented as a vector using techniques like word embeddings. The encoder processes the input tokens one by one, updating its internal state at each step.



Implementation: In the code, the encoder is implemented using a recurrent neural network (RNN) with a Gated Recurrent Unit (GRU) cell. The GRU cell processes the input sequence token by token and updates its internal state at each step.

Embedding Layer: Converts input tokens into dense vector representations.
GRU Layer: Processes the embedded input sequence and produces output sequences at each time step. The final output sequence and hidden state of the GRU cell serve as the encoder outputs
Bahdanau Attention Mechanism: Bahdanau attention allows the decoder to focus on different parts of the input sequence dynamically at each time step. It calculates attention weights based on the current decoder state and the encoder outputs. We also experimented with dot product attention but Bahdanau attention mechanism performed well. Bahdanau attention performs well due to its ability to capture fine-grained alignment between input

and output sequences. By allowing the model to focus on relevant parts of the input sequence dynamically, Bahdanau attention facilitates more accurate and contextually relevant translations. This mechanism enhances the model's ability to handle long sequences and capture dependencies effectively, leading to improved translation quality.

Decoder: The decoder takes the context vector produced by the encoder and generates the output sequence token by token. At each time step, the decoder attends to different parts of the input sequence (or context) using an attention mechanism. This allows the model to focus on relevant information when generating each token of the output sequence.
Embedding Layer: Converts target tokens into dense vector representations.
GRU Layer: Similar to the encoder, the decoder uses a GRU cell to process the embedded target sequence. It takes the context vector and the previous decoder hidden state as inputs and produces the next output token and decoder hidden state.
Dense Layer: Maps the decoder output to the vocabulary size, producing logits for each token in the vocabulary.

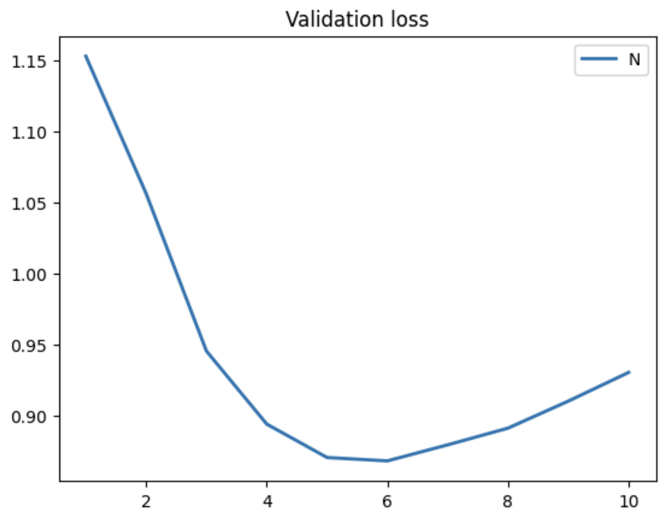
4. Experimental Setup

Usage: The provided code is executed within a Google Colab notebook environment.
GPU Acceleration: The T4 GPU is utilised for accelerating model training and inference, leveraging its parallel processing capabilities and specialised hardware features.
Training Configuration: The model is trained using TensorFlow and Keras APIs, with the training process distributed across the GPU cores for faster convergence.
Data Handling: The dataset and trained model checkpoints may be stored in Google Drive, allowing for easy access and persistence across sessions.

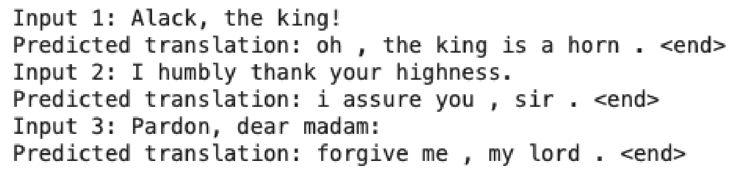
5. Results and Discussion

The training loss decreases steadily over the epochs, indicating effective learning initially. However, towards the end of training, the validation loss begins to increase, suggesting potential overfitting.

```
Text(0.5, 1.0, 'Validation loss')
```

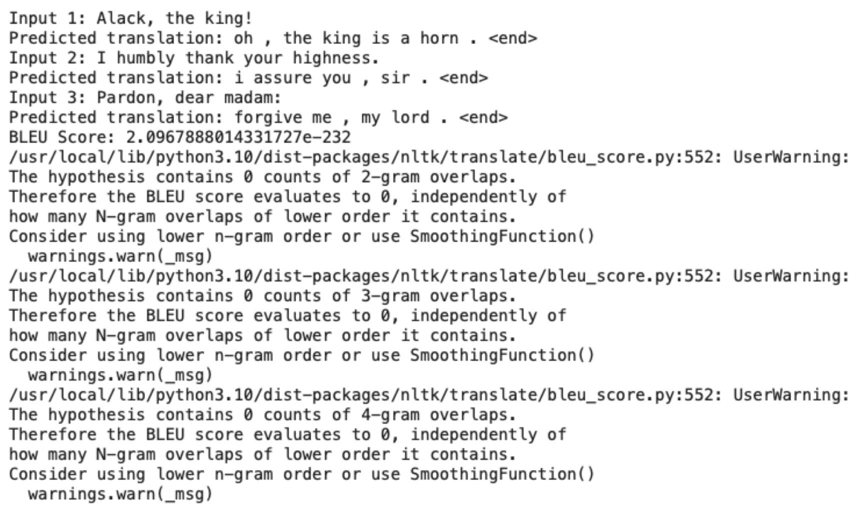


Overfitting occurs when the model learns to memorise the training data rather than generalising well to unseen data.
The quality and quantity of the training data directly impact the model's performance. If the dataset lacks diversity or contains noise, the model may struggle to generalise effectively, leading to overfitting.
Limited computational resources could restrict the model's training time or capacity. Inadequate resources may prevent the model from fully exploring the solution space, affecting its ability to learn complex patterns in the data.



```
Input 1: Alack, the king!
Predicted translation: oh , the king is a horn . <end>
Input 2: I humbly thank your highness.
Predicted translation: i assure you , sir . <end>
Input 3: Pardon, dear madam:
Predicted translation: forgive me , my lord . <end>
```

While the BLEU score provides an objective measure of translation quality, it relies solely on n-gram overlap between predicted and reference translations. It may not fully capture the quality of translations, especially in cases involving paraphrasing or semantic similarity.
Human evaluation is essential to complement automated metrics like BLEU score. In this case, human evaluation resulted in an accuracy of 45 percent, indicating moderate agreement between human judgments and model predictions.



```
Input 1: Alack, the king!
Predicted translation: oh , the king is a horn . <end>
Input 2: I humbly thank your highness.
Predicted translation: i assure you , sir . <end>
Input 3: Pardon, dear madam:
Predicted translation: forgive me , my lord . <end>
BLEU Score: 2.0967888014331727e-232
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
```

6. Conclusion

While the seq2seq model with Bahdanau attention demonstrates effective learning initially, it exhibits signs of overfitting towards the end of training. The quality and quantity of training data, computational resources, and the limitations of automated metrics like BLEU score should be considered when evaluating model performance. Human evaluation complements automated metrics, offering valuable insights into translation quality and ensuring a more comprehensive assessment.

Incorporating unsupervised learning techniques for dataset creation can greatly enhance the effectiveness and robustness of translation models. By augmenting the dataset with diverse examples and leveraging self-supervised learning and adversarial training, the model can improve its generalisation ability and reduce overfitting. When evaluating model performance, it is essential to consider the quality of the dataset and leverage unsupervised learning as a powerful tool for dataset enhancement. This holistic approach, complemented by human evaluation, ensures a comprehensive assessment of translation quality and model performance.

Advantages of Unsupervised Learning for Dataset Enhancement:

Increased Data Diversity: Unsupervised learning techniques can introduce additional variability into the dataset, capturing a wider range of linguistic patterns and phenomena.
Reduced Annotation Costs: Since unsupervised techniques operate on unlabeled data, they can significantly reduce the need for manual annotation, thereby lowering annotation costs and efforts.
Improved Model Generalisation: By augmenting the dataset with diverse examples, unsupervised learning can help the model generalise better to unseen data and reduce the risk of overfitting.
Enhanced Model Robustness: Adversarial training and self-supervised learning can improve the model's robustness to noise and variations in the data, leading to more reliable translations.

B. Unsupervised Encoder-Decoder transformer model with iterative refinement and translation model.

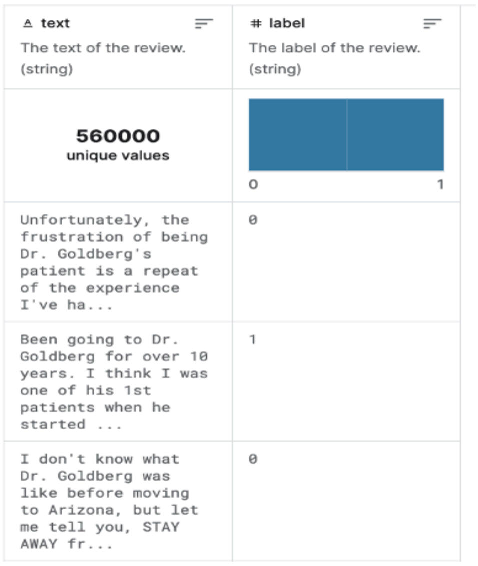
1. Introduction:
The Unsupervised Encoder-DecoderTransformer Model utilises a transformative encoder-decoder architecture, to process and generate text across languages by leveraging monolingual datasets alone, eliminating the need for parallel corpora. The iterative refinement allows the model to enhance the quality of its translations through successive iterations, correcting errors and improving fluency with each step.

Objective of the Project: To convert the positive reviews into corresponding negative reviews. We referred the methodology to address this problem using Unsupervised Text Style Transfer by referring the paper: IMaT: Unsupervised Text Attribute Transfer via Iterative Matching and Translation authored by Zhijing Jin, Di Jin, Jonas Mueller, Nicholas Matthews and Enrico Santus.

2. Dataset:
The Yelp Reviews Polarity dataset was used in the project and is extracted from The Yelp reviews dataset consists of reviews from Yelp about various Business Sources. Credits to Xiang Zhang for constructing the polarity dataset.

Format:
text is the text of the review. label represents the sentiment with 0 and 1 indicating negative and positive reviews respectively.

Example:

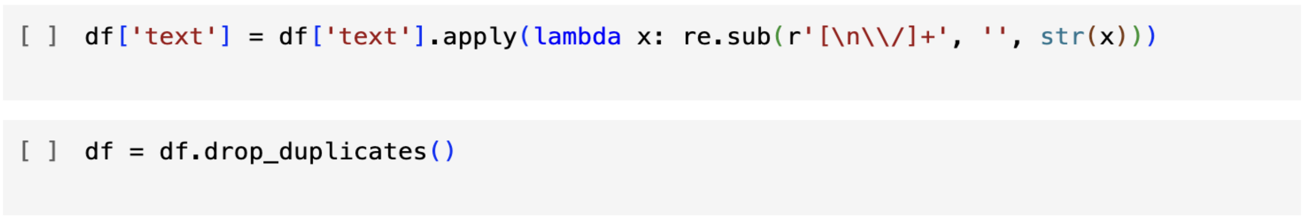


text The text of the review. (string)	label The label of the review. (string)
560000 unique values	0 1
Unfortunately, the frustration of being Dr. Goldberg's patient is a repeat of the experience I've ha...	0
Been going to Dr. Goldberg for over 10 years. I think I was one of his 1st patients when he started ...	1
I don't know what Dr. Goldberg was like before moving to Arizona, but let me tell you, STAY AWAY fr...	0

Size: There are 560000 rows in the dataset. After initial preprocessing and matching steps to create the base parallel corpora, 893 pairs were retained.

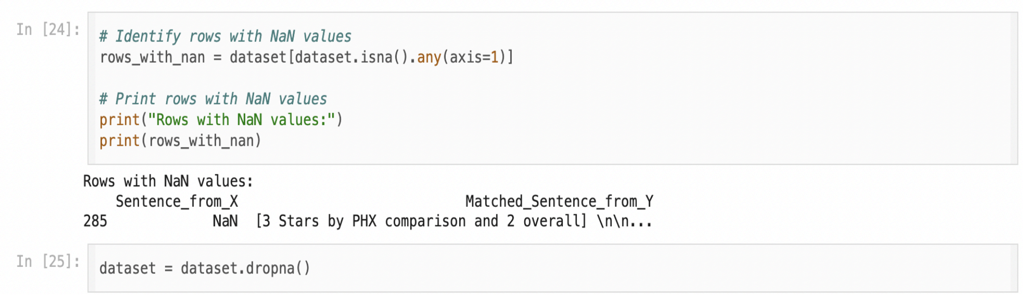
Preprocessing:

● The text had NAN characters and back slashes as well as other special characters that were removed.



```
[ ]  df['text'] = df['text'].apply(lambda x: re.sub(r'[\n\\/]+', '', str(x)))
```

```
[ ]  df = df.drop_duplicates()
```



```
In [24]: # Identify rows with NaN values
         rows_with_nan = dataset[dataset.isna().any(axis=1)]

         # Print rows with NaN values
         print("Rows with NaN values:")
         print(rows_with_nan)

Rows with NaN values:
    Sentence_from_X                            Matched_Sentence_from_Y
285             NaN  [3 Stars by PHX comparison and 2 overall] \n\n...

In [25]: dataset = dataset.dropna()
```

● The text was lowercase and standardised allowing us to effectively utilize the GloVe model for computing the Word Mover's Distance between sentences, offering a measure of semantic shift crucial for our iterative refinement process.

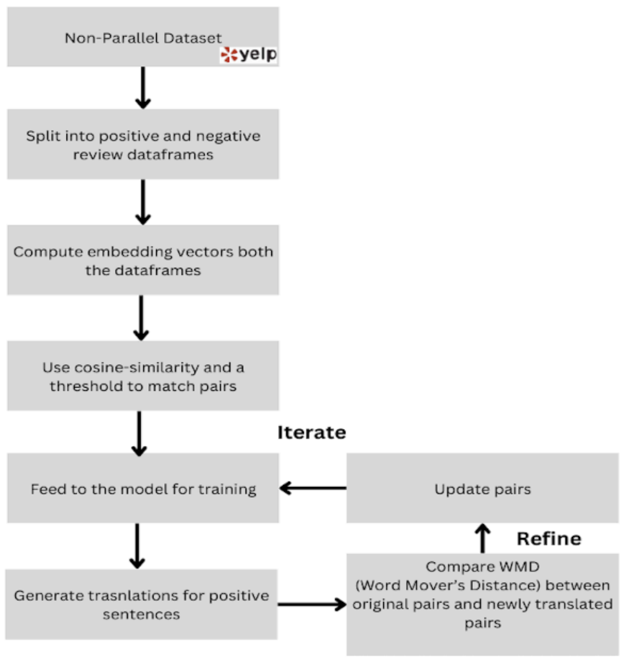
```
from gensim.models import KeyedVectors
from gensim.parsing.preprocessing import preprocess_string

def preprocess(text):
    # Simple preprocessing: tokenize, lower, alpha characters
    result= preprocess_string(text)
    print(result)
    return result
```

Tokenization: Tokenization is performed using T5Tokenizer for the "t5-small" variant using Hugging Face's Transformers library, enabling the conversion of text into a numerical format that the T5 model can understand.

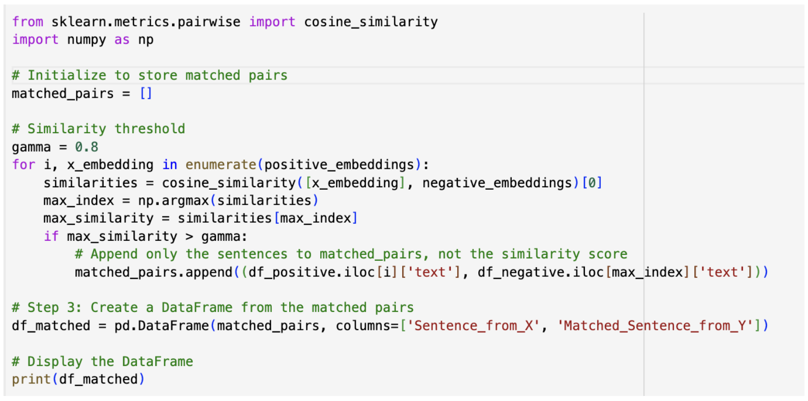
Padding: The tokenizer applies padding to ensure all sequences are of the same length (max_length), truncates any input longer than max_length, and converts the tokens into PyTorch tensors (return_tensors='pt').

3. Methodology

Architecture:

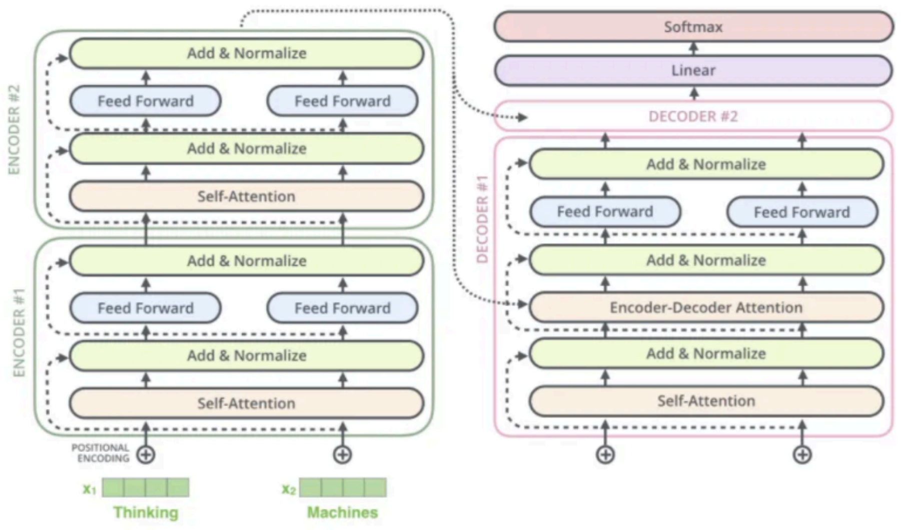

Non-Parallel Dataset
yelp
Split into positive and negative review dataframes
Compute embedding vectors both the dataframes
Use cosine-similarity and a threshold to match pairs
Iterate
Feed to the model for training
Update pairs
Refine
Generate trasnlations for positive sentences
Compare WMD (Word Mover's Distance) between original pairs and newly translated pairs

Step1 : Creating initial pseudo-parallel corpus: The process begins with constructing a pseudo-parallel corpus by aligning semantically similar sentences from source and target corpora by using cosine-similarity.Only those similar pairs are retained whose similarity index is greater than the threshold.



```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Initialize to store matched pairs
matched_pairs = []

# Similarity threshold
gamma = 0.8
for i, x_embedding in enumerate(positive_embeddings):
    similarities = cosine_similarity([x_embedding], negative_embeddings)[0]
    max_index = np.argmax(similarities)
    max_similarity = similarities[max_index]
    if max_similarity > gamma:
        # Append only the sentences to matched_pairs, not the similarity score
        matched_pairs.append((df_positive.iloc[i]['text'], df_negative.iloc[max_index]['text']))

# Step 3: Create a DataFrame from the matched pairs
df_matched = pd.DataFrame(matched_pairs, columns=['Sentence_from_X', 'Matched_Sentence_from_Y'])

# Display the DataFrame
print(df_matched)
```

Step 2: Model Training:



A T5 (Text-to-Text Transfer Transformer), a model built on the Transformer architecture that adopts a text-to-text framework is used.The process involves feeding textual input into the model and training it to generate a specified target text as output. The model underwent training for a duration of 30 epochs, utilising the Adam optimizer along with a regimen of 100 warm-up steps to gradually adapt the learning rate. An early stopping mechanism was also implemented to optimise model performance. This precautionary measure terminates the training process if there is an observed increase in evaluation loss, thereby ensuring that the optimal model iteration is selected and preserved.

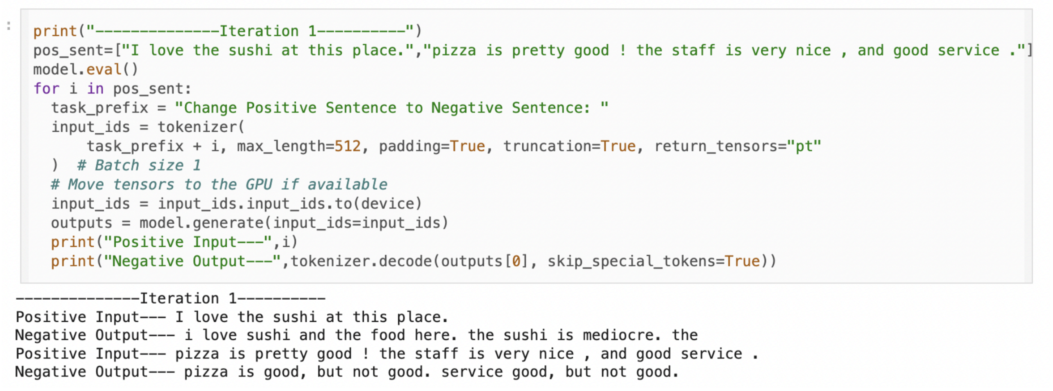
Step 3: Iterative Refinement: After the initial training, the model's outputs (translated sentences) are used to refine the alignment in the pseudo-parallel corpus.WMD scores are used for selecting training pairs for re-training the model. Pairs with lower WMD scores (indicating better content preservation) can be prioritized to guide the model towards maintaining semantic content while transferring text attributes. These newly generated sentences are then compared with the existing ones in the pseudo-parallel corpus, and alignments are updated to reflect sentences with closer semantic meanings and more accurate attribute transfers. Sentence pairs are then passed through the model to generate new target attribute sentences.

4. Experimental Setup

Usage: The provided code is executed within a Google Colab notebook environment.
GPU Acceleration: The T4 GPU is utilised for accelerating model training and inference, leveraging its parallel processing capabilities and specialised hardware features.
Training Configuration: The model is trained using TensorFlow and Keras APIs, with the training process distributed across the GPU cores for faster convergence.
Data Handling: The dataset and trained model checkpoints may be stored in Google Drive, allowing for easy access and persistence across sessions.

5. Results and Discussions:

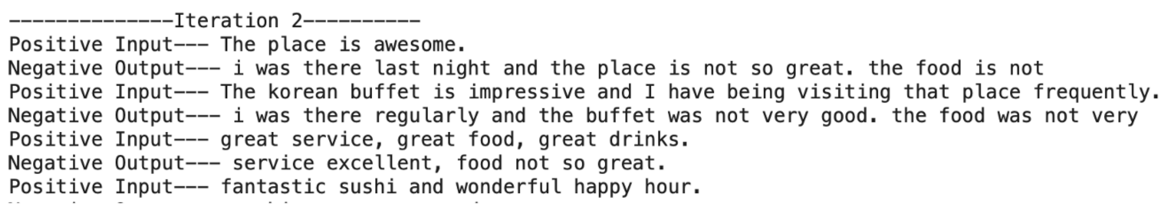
We trained the model by iteratively refining the dataset by using WMD distance for 2 iterations.



```
print("--------------Iteration 1----------")
pos_sent=["I love the sushi at this place.","pizza is pretty good ! the staff is very nice , and good service ."]
model.eval()
for i in pos_sent:
  task_prefix = "Change Positive Sentence to Negative Sentence: "
  input_ids = tokenizer(
      task_prefix + i, max_length=512, padding=True, truncation=True, return_tensors="pt"
  )  # Batch size 1
  # Move tensors to the GPU if available
  input_ids = input_ids.input_ids.to(device)
  outputs = model.generate(input_ids=input_ids)
  print("Positive Input---",i)
  print("Negative Output---",tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
--------------Iteration 1----------
Positive Input--- I love the sushi at this place.
Negative Output--- i love sushi and the food here. the sushi is mediocre. the
Positive Input--- pizza is pretty good ! the staff is very nice , and good service .
Negative Output--- pizza is good, but not good. service good, but not good.
```

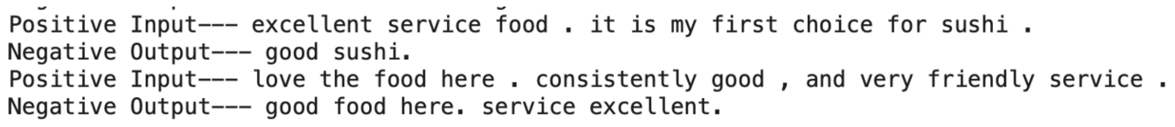
After the first iteration, we found that there were 154 rows updated with the new data pairs.



```
--------------Iteration 2----------
Positive Input--- The place is awesome.
Negative Output--- i was there last night and the place is not so great. the food is not
Positive Input--- The korean buffet is impressive and I have being visiting that place frequently.
Negative Output--- i was there regularly and the buffet was not very good. the food was not very
Positive Input--- great service, great food, great drinks.
Negative Output--- service excellent, food not so great.
Positive Input--- fantastic sushi and wonderful happy hour.
```

We can see significant improvement in the results after the first iteration. If we train the model for more iterations, we will be able to generate even better results.

However, in some cases, the model isn't able to output negative sentiments in the target sentence.



```
Positive Input--- excellent service food . it is my first choice for sushi .
Negative Output--- good sushi.
Positive Input--- love the food here . consistently good , and very friendly service .
Negative Output--- good food here. service excellent.
```

6. Conclusion and Future Scope:

We started with a non-parallel corpus adopting an unsupervised approach. For the initial pseudo-parallel-corpus, we used cosine similarity and used a Unsupervised Encoder-Decoder Transformer Model to train on it. Subsequently, we continued to iteratively refine the dataset by using WMD distance for 2 iterations.

This methodology proves advantageous when working with non-parallel datsets, although it does require a certain degree of interdependence between the two data varieties to establish the initial matches. Successive iterations enhance the model's precision, a trend confirmed through human evaluations. While working with a smaller dataset may result in modest BLEU scores, there is still a discernible upward trend. Opting for an independently trained transformer over one that is pre-trained could potentially yield more effective results.

From the results,we see certain cases where the model is unable to generate a negative output. We suggest 2 approaches to address this:

1. We can add a classification layer in between to classify whether the nature of a sentence is positive or negative.
2. We can use a contrastive loss function. The main goal of the contrastive loss is to ensure that similar pairs (or "positive" pairs) are closer together in the learned representation space, while dissimilar pairs (or "negative" pairs) are further apart.