Convergence of Senses

The goal of this project is to use deep learning models, primarily Transformer-based architectures, to create a system that can translate spoken or typed English input into French. Transformer models are now considered a method for various NLP problems, including machine translation.

Real-time language translation, whether spoken or written, has several useful uses. These include improving interlanguage communication, supporting language learning, and promoting understanding among people from various cultural backgrounds.

To train the translation model, we gathered parallel text datasets in both French and English. To ensure that the model can handle a wide range of input types, this dataset contains a variety of texts.

If the input is in audio format, we should use the below to convert it into text and then give that text as input to the model.

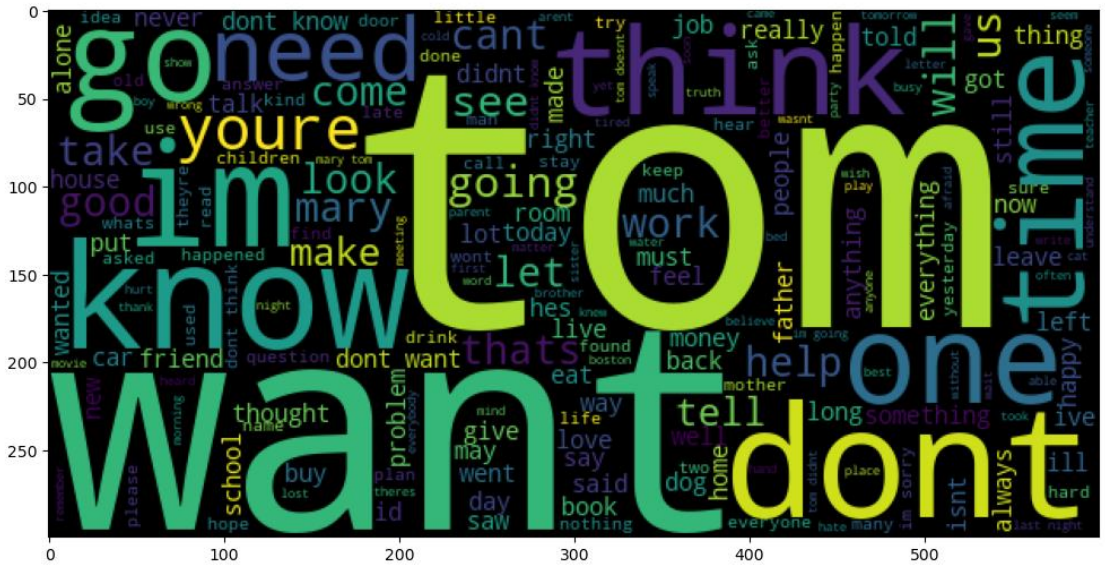
Speech Recognition:

- The audio file is loaded using the AudioFileClip class from the moviepy.editor module. The loaded audio file is stored in the audio_clip variable.
- The loaded audio file is then written to a temporary WAV file named "temp.wav" using the write_audiofile method of the audio_clip object.
- An instance of the Recognizer class from the speech_recognition module is created and assigned to the recognizer variable.
- The temporary WAV file "temp.wav" is opened as an audio source using a with statement.
- The record method of the recognizer instance is called to capture the audio data from the source.
- The captured audio data is passed to the recognize_google method of the recognizer instance for speech recognition.
- The recognized text is stored in the text variable and printed.
- This text is sent as an input to the model in case where the input for translation is given as an audio.

If the input is in text format, we send it directly as an input.

Dataset:

- The distribution and frequency of words in the English text data taken from the English-French translation dataset is displayed. The most frequently occurring words in the dataset and their relative importance are shown in this visualization.

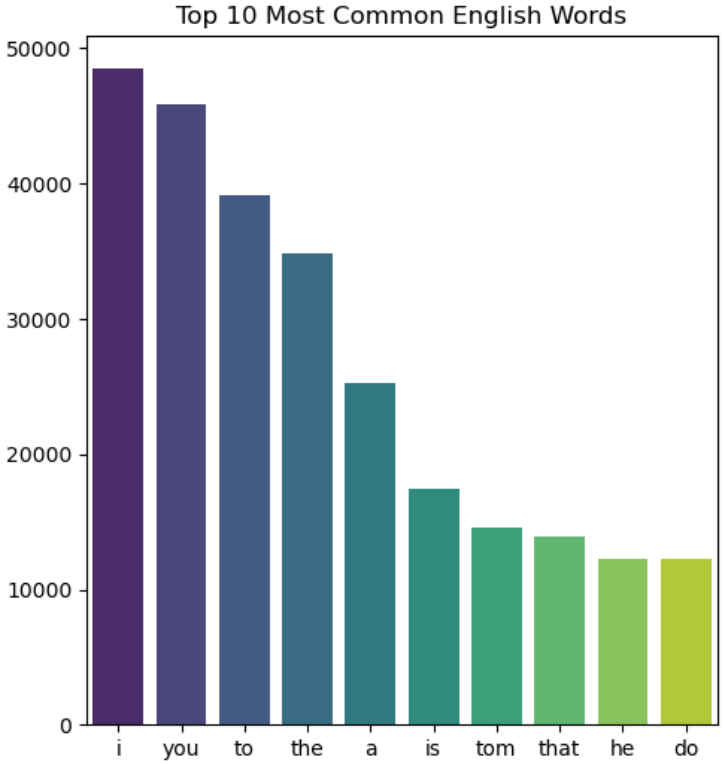


- The distribution and frequency of words in the French text data taken from the English-French translation dataset

- It determines the top 10 most common English terms according to frequency of occurrence as well as the total number of English words and unique English words. Understanding the linguistic properties of the dataset is aided by these statistics, which offer insights into the vocabulary size and word distribution.

```
Total number of English words: 1082094
Number of unique English words: 14637
Top 10 most common English words: ('i', 'you', 'to', 'the', 'a', 'is', 'tom', 'that', 'he', 'do')
```

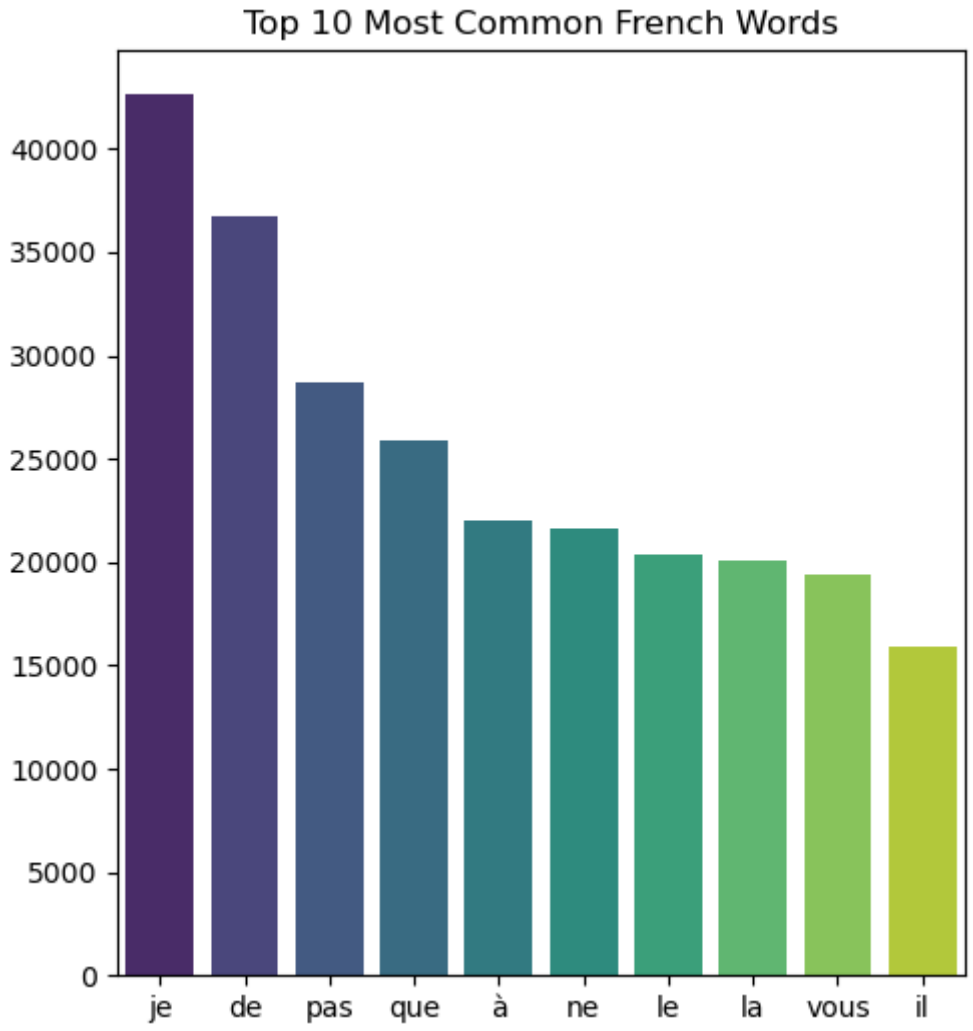
- The top ten most often occurring English words found in the English text data taken from the English-French translation dataset are displayed in this image. An English word's frequency of occurrence is represented by each bar in a horizontal bar plot.



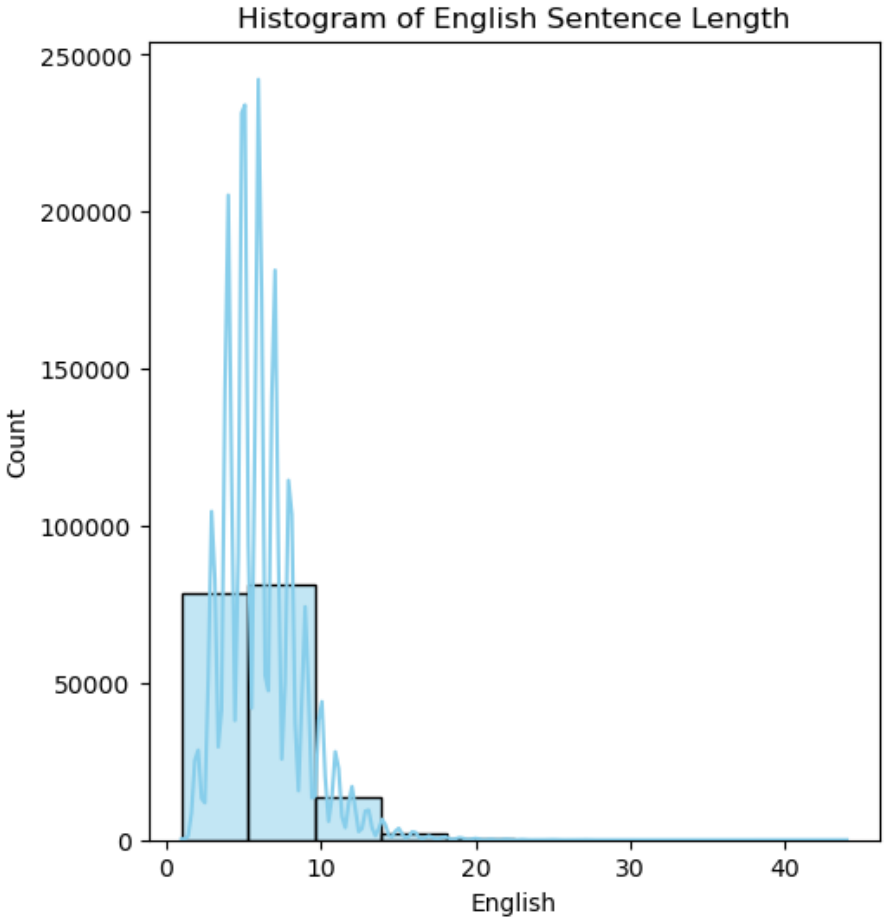
- It determines the top 10 most common French terms according to frequency of occurrence as well as the total number of French words and unique French words. Understanding the linguistic properties of the dataset is aided by these statistics, which offer insights into the vocabulary size and word distribution.

```
Total number of French words: 1082094
Number of unique French words: 14637
Top 10 most common French words: ('je', 'de', 'pas', 'que', 'à', 'ne', 'le', 'la', 'vous', 'il')
```

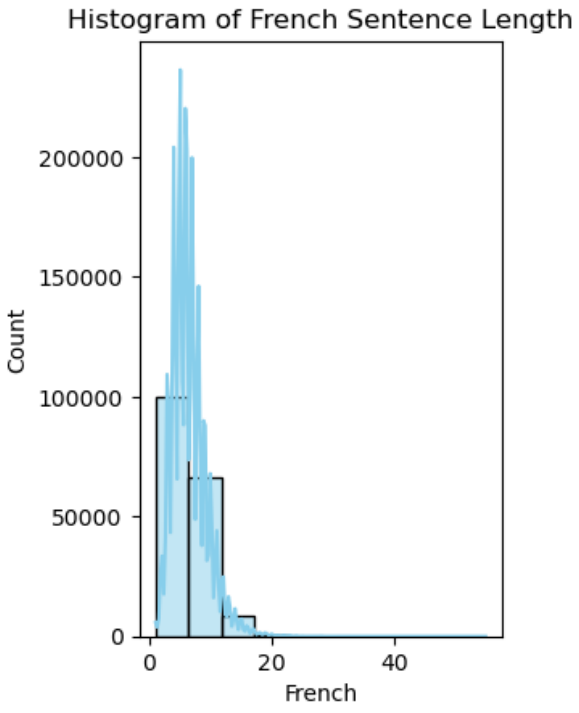
- The top ten most often occurring French words found in the French text data taken from the English-French translation dataset are displayed in this image. A French word's frequency of occurrence is represented by each bar in a horizontal bar plot.



- The distribution of English sentence lengths within the English text data taken from the English-French translation dataset is shown graphically in this visualization as a histogram. Each bar in the histogram shows how frequently sentences with a given length fall within a particular range.



- The distribution of French sentence lengths within the French text data taken from the English-French translation dataset is shown graphically in this visualization as a histogram. Each bar in the histogram shows how frequently sentences with a given length fall within a particular range.



Data Cleaning:

The performance of machine learning models is greatly impacted by the cleanliness and quality of the dataset, particularly when it comes to natural language processing tasks like machine translation. The methods used to clean and preprocess the English-French parallel dataset to achieve the best possible model training and performance are described in this project.

- Special characters have been eliminated from the English text. Punctuation, symbols, and other non-alphanumeric characters are examples of special characters that might generate noise and impede the model's learning process.
- We then concentrated on erasing the numbers from the French text. We removed digits from the French sentences because they are irrelevant for translation and could confuse the model.
- Variations in the amount of whitespace used can break up the text's consistency and have an impact on the model's performance. As a result, we normalized whitespace by adding a single space in place of several consecutive whitespace characters.
- By eliminating extra special characters unique to the French language, more cleaning was done. Symbols, punctuation, and other non-alphanumeric characters were among them.

Tokenization:

For preprocessing and later model training, it is crucial to comprehend the vocabulary and tokenization structure of the French and English languages in our dataset. The method of creating the French and English vocabularies and tokenizing French sentences from the given dataset is described below.

- Every statement in both French and English is tokenized into its component words, resulting in a list of tokens unique to each sentence. Tokens that are unique to the French text and their matching index are initialized into an empty dictionary.
- A dictionary with distinct tokens from the text data and their matching indices is effectively produced by the vocabulary construction method. The number of distinct tokens in the text is indicated by the size of the vocabulary.

A crucial stage in tasks involving natural language processing is vocabulary creation, which makes it possible to transform textual data into numerical representations appropriate for machine learning models.

Tokenizing sentences and building a vocabulary with distinct tokens and matching indices has allowed us to produce a basic building block that is required for later phases of data preparation and model training.

Padding:

By padding the sentences to guarantee that they are all the same length and indexing each token according to the previously created vocabulary, we hope to prepare the text data for input into the translation model.

- Every sentence is represented by a series of tokens that are created through vocabulary creation and tokenization.
- Every token in the sequence is padded with zeros to fit the maximum length if its length is less than the maximum sentence length. This guarantees that every sentence is the same length, which makes batch processing easier when training the model.

To train the translation model and ultimately enable accurate and efficient English-French translation, the padded and indexed English data and the matching French data are used.

Data Preparation:

- To make the process of training, evaluating, and validating the model easier, we separated the preprocessed English and French data into sets for testing, validation, and training. By dividing up the dataset, we make sure that the model is trained on one part, validated on another, and tested on a different part to evaluate how well it generalizes.
- For training, validation, and testing, three TensorDataset objects are produced, each comprising paired input and target data.
- TensorDataset objects are used to create data loaders, which allow for effective batch processing for model training, validation, and assessment.

Model 1:

The model has a transformer-based architecture and is intended for English-to-French translation. This model learns the mappings between English and French sentences by utilizing a feedforward neural network, many layers of encoders and decoders, and embeddings.

- First, the input tokens in French and English are routed through embedding layers. The model can learn meaningful representations of the input tokens because of the conversion of the token indices into dense vectors of a fixed dimension by embedding layers.
- There are numerous levels of encoders and decoders in the Transformer design. While the decoder creates the output French tokens using the encoded English representation, the encoder examines the input English tokens to extract contextual information.
- Self-attention mechanisms are present in both the encoder and decoder layers, allowing the model to concurrently focus on distinct segments of the input and output sequences.
- The input embeddings are implicitly handled by positional encoding, which enables the model to comprehend the tokens' sequential order inside the input sequences. To capture the sequential dependencies in the incoming data, this positional information is essential.
- Based on the previously created tokens and the encoded representation of the input sequence, the transformer decoder predicts the next token in the output sequence.
- The transformer model's ultimate output is subjected to a linear layer to forecast the probability distribution over the French vocabulary. The probability distribution, which shows the chance of each French token in the vocabulary, is usually obtained by applying the SoftMax function to the output.

```
Eng_Fr_Translator(
  (src_embedding): Embedding(14638, 128)
  (tgt_embedding): Embedding(29411, 128)
  (transformer): Transformer(
    (encoder): TransformerEncoder(
      (layers): ModuleList(
        (0): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=128, out_features=128, bias=True)
          )
          (linear1): Linear(in_features=128, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=128, bias=True)
          (norm1): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
      )
      (norm): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
    )
    (decoder): TransformerDecoder(
      (layers): ModuleList(
        (0): TransformerDecoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=128, out_features=128, bias=True)
          )
          (multihead_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=128, out_features=128, bias=True)
          )
          (linear1): Linear(in_features=128, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=128, bias=True)
          (norm1): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
          (norm3): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
          (dropout3): Dropout(p=0.1, inplace=False)
        )
      )
      (norm): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
    )
  )
  (fc): Linear(in_features=128, out_features=29411, bias=True)
)
```

Performance Metrics

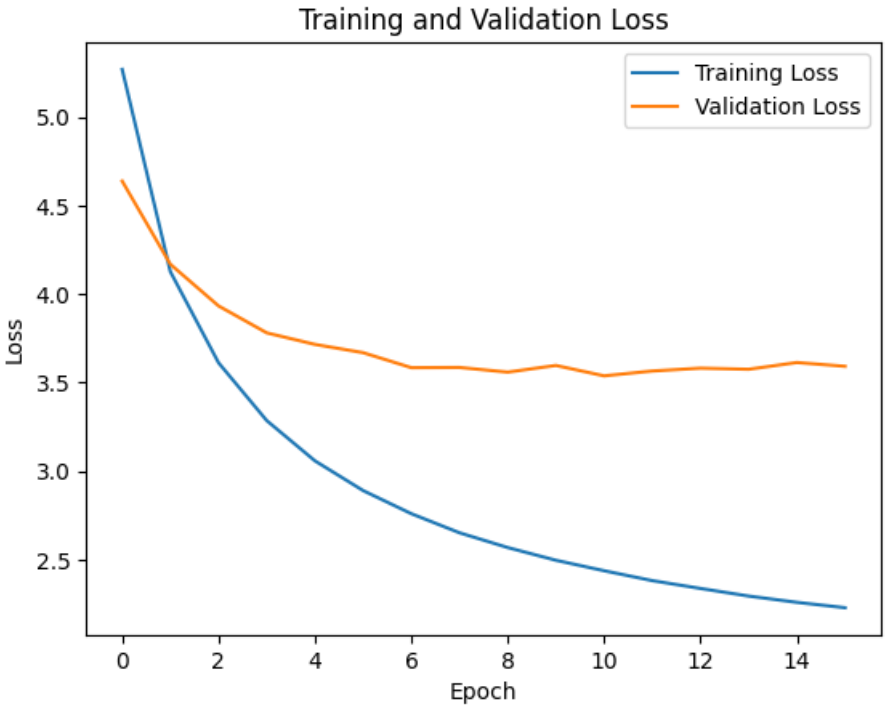
we have trained my model with 30 epochs to get the best results.

Model 1 was trained with a dropout of 0.5 and n_head set to 4. The training process comprised 30 epochs. The loss decreased gradually over epochs, with the best validation loss achieved at epoch 16 (3.5913). However, the model did not exhibit further improvement in validation loss after epoch 16, leading to early stopping. The best overall loss attained was 3.6.

```
Epoch [1], Train Loss: 5.2687, Val Loss: 4.6376
Epoch [2], Train Loss: 4.1258, Val Loss: 4.1677
Epoch [3], Train Loss: 3.6112, Val Loss: 3.9324
Epoch [4], Train Loss: 3.2846, Val Loss: 3.7802
Epoch [5], Train Loss: 3.0572, Val Loss: 3.7150
Epoch [6], Train Loss: 2.8892, Val Loss: 3.6683
Epoch [7], Train Loss: 2.7587, Val Loss: 3.5836
Epoch [8], Train Loss: 2.6508, Val Loss: 3.5847
Epoch [9], Train Loss: 2.5675, Val Loss: 3.5584
Epoch [10], Train Loss: 2.4956, Val Loss: 3.5961
Epoch [11], Train Loss: 2.4365, Val Loss: 3.5374
Epoch [12], Train Loss: 2.3806, Val Loss: 3.5647
Epoch [13], Train Loss: 2.3364, Val Loss: 3.5807
Epoch [14], Train Loss: 2.2934, Val Loss: 3.5745
Epoch [15], Train Loss: 2.2579, Val Loss: 3.6127
Epoch [16], Train Loss: 2.2276, Val Loss: 3.5913
Validation loss has not improved for 5 epochs. Early stopping...
```

Loss: The best loss we got was 3.6

Training	Validation	Testing
2.22	3.59	3.6



The performance metrics indicate that the model achieved a training loss of 2.22 and a validation loss of 3.59, with a testing loss of 3.6. While the model demonstrated reasonable performance, there may be room for improvement, particularly in stabilizing the validation loss and potentially enhancing the testing loss.

Model 2

n_head=4 and dropout = 0.5

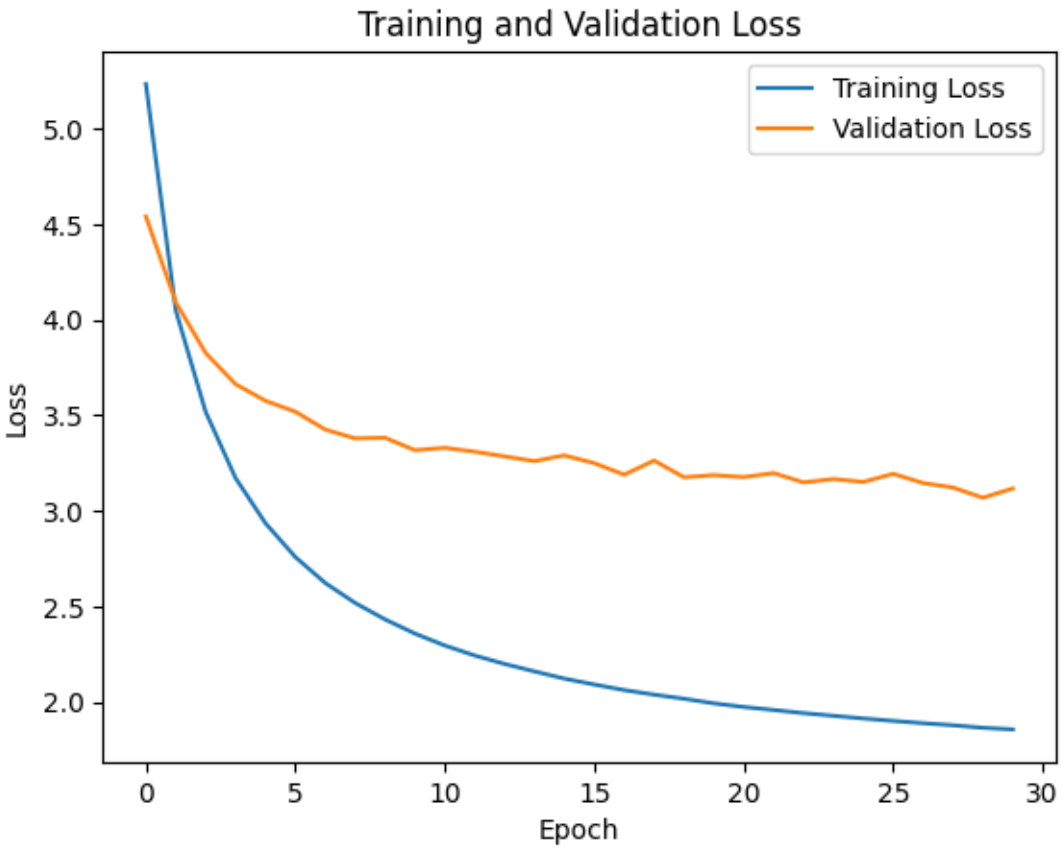
Model 2 utilized a dropout of 0.5 and n_head set to 4. It underwent training for 30 epochs as well. The loss decreased consistently throughout training, with the best validation loss observed at epoch 30 (3.2493). The model exhibited a gradual reduction in loss over epochs, indicating effective learning.

Performance metrics:

```
Epoch [1], Train Loss: 5.2343, Val Loss: 4.5412
Epoch [2], Train Loss: 4.0481, Val Loss: 4.0911
Epoch [3], Train Loss: 3.5176, Val Loss: 3.8257
Epoch [4], Train Loss: 3.1712, Val Loss: 3.6627
Epoch [5], Train Loss: 2.9349, Val Loss: 3.5756
Epoch [6], Train Loss: 2.7579, Val Loss: 3.5188
Epoch [7], Train Loss: 2.6220, Val Loss: 3.4252
Epoch [8], Train Loss: 2.5174, Val Loss: 3.3790
Epoch [9], Train Loss: 2.4314, Val Loss: 3.3824
Epoch [10], Train Loss: 2.3574, Val Loss: 3.3172
Epoch [11], Train Loss: 2.2946, Val Loss: 3.3297
Epoch [12], Train Loss: 2.2423, Val Loss: 3.3093
Epoch [13], Train Loss: 2.1982, Val Loss: 3.2847
Epoch [14], Train Loss: 2.1591, Val Loss: 3.2601
Epoch [15], Train Loss: 2.1214, Val Loss: 3.2899
Epoch [16], Train Loss: 2.0907, Val Loss: 3.2493
Epoch [17], Train Loss: 2.0614, Val Loss: 3.1889
Epoch [18], Train Loss: 2.0376, Val Loss: 3.2626
Epoch [19], Train Loss: 2.0160, Val Loss: 3.1755
Epoch [20], Train Loss: 1.9918, Val Loss: 3.1866
Epoch [21], Train Loss: 1.9729, Val Loss: 3.1762
Epoch [22], Train Loss: 1.9572, Val Loss: 3.1973
Epoch [23], Train Loss: 1.9407, Val Loss: 3.1486
Epoch [24], Train Loss: 1.9268, Val Loss: 3.1662
Epoch [25], Train Loss: 1.9128, Val Loss: 3.1511
Epoch [26], Train Loss: 1.9001, Val Loss: 3.1934
Epoch [27], Train Loss: 1.8880, Val Loss: 3.1452
Epoch [28], Train Loss: 1.8772, Val Loss: 3.1218
Epoch [29], Train Loss: 1.8639, Val Loss: 3.0682
Epoch [30], Train Loss: 1.8557, Val Loss: 3.1170
```

Loss: The best loss we got was 3.1391

Training	Validation	Testing
1.855	3.1170	3.1391



The performance metrics for Model 2 reveal a training loss of 1.855, a validation loss of 3.1170, and a testing loss of 3.1391. Notably, Model 2 demonstrated improved performance compared to Model 1, with lower losses across all datasets. This suggests that the adjustments made to the model architecture and parameters positively impacted its learning capability.

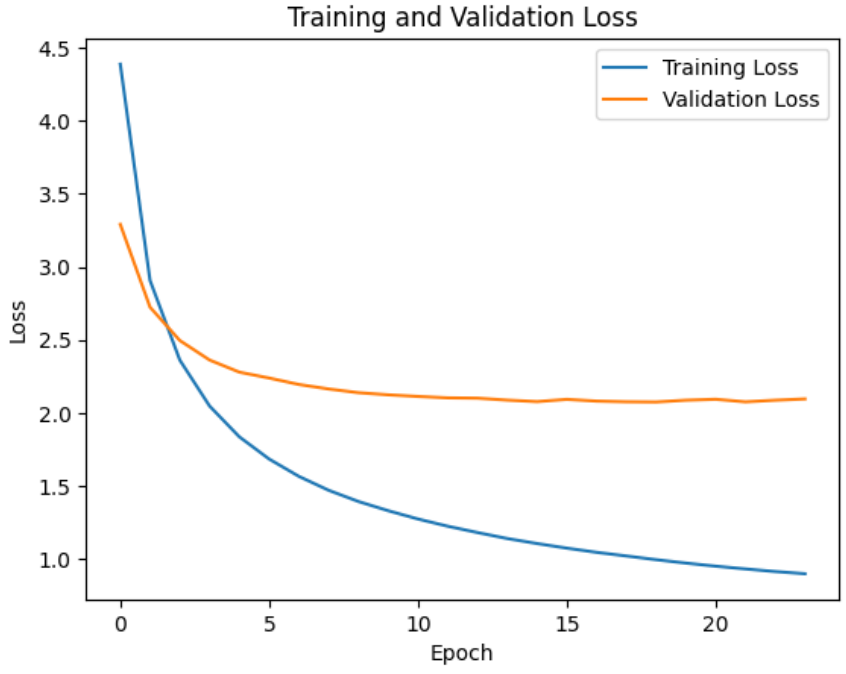
Model 3:
Dropout = 0.1
n_head = 8
Model 3 was trained with a dropout of 0.1 and n_head set to 8. The training process encompassed 24 epochs. The loss exhibited a steady decrease during training, with the best validation loss achieved at epoch 24 (2.0955). Similar to Model 1, early stopping was triggered due to lack of improvement in validation loss.

```
Epoch [1], Train Loss: 4.3875, Val Loss: 3.2910
Epoch [2], Train Loss: 2.9065, Val Loss: 2.7237
Epoch [3], Train Loss: 2.3615, Val Loss: 2.4956
Epoch [4], Train Loss: 2.0469, Val Loss: 2.3619
Epoch [5], Train Loss: 1.8360, Val Loss: 2.2790
Epoch [6], Train Loss: 1.6842, Val Loss: 2.2387
Epoch [7], Train Loss: 1.5654, Val Loss: 2.1946
Epoch [8], Train Loss: 1.4701, Val Loss: 2.1637
Epoch [9], Train Loss: 1.3934, Val Loss: 2.1384
Epoch [10], Train Loss: 1.3304, Val Loss: 2.1238
Epoch [11], Train Loss: 1.2735, Val Loss: 2.1126
Epoch [12], Train Loss: 1.2241, Val Loss: 2.1030
Epoch [13], Train Loss: 1.1808, Val Loss: 2.1008
Epoch [14], Train Loss: 1.1398, Val Loss: 2.0874
Epoch [15], Train Loss: 1.1056, Val Loss: 2.0772
Epoch [16], Train Loss: 1.0740, Val Loss: 2.0927
Epoch [17], Train Loss: 1.0453, Val Loss: 2.0807
Epoch [18], Train Loss: 1.0204, Val Loss: 2.0761
Epoch [19], Train Loss: 0.9950, Val Loss: 2.0746
Epoch [20], Train Loss: 0.9717, Val Loss: 2.0869
Epoch [21], Train Loss: 0.9505, Val Loss: 2.0933
Epoch [22], Train Loss: 0.9317, Val Loss: 2.0756
Epoch [23], Train Loss: 0.9138, Val Loss: 2.0868
Epoch [24], Train Loss: 0.8989, Val Loss: 2.0955
Validation loss has not improved for 5 epochs. Early stopping...
```

Loss: The best loss we got was 3.1115

Training	Validation	Testing
0.8989	2.0955	3.1115

The performance metrics indicate that Model 3 attained a training loss of 0.8989, a validation loss of 2.0955, and a testing loss of 3.1115. Model 3 outperformed both Model 1 and Model 2 in terms of validation loss, demonstrating superior learning efficiency and generalization capability.



In summary, Model 3 showcased the most promising performance among the three models, achieving the lowest validation loss and demonstrating effective learning. Further analysis and fine-tuning may be warranted to optimize the model's performance for real-world applications.

Real-world deep learning application:

- By using this deep learning model for speech recognition tasks, people with disabilities or speech impairments can communicate more successfully. Spoken words can be translated into text using these models, which can then be utilized for other purposes.
- Natural language processing (NLP) deep learning models can help with real-time language translation, making it easier for people who speak different languages to communicate with one another. This is especially helpful for people with impairments who might find it challenging to communicate with conventional methods.

Novelty

- By including several input modalities, like speech recognition and keyboard input, our product may be distinguished and serve a wide spectrum of users, including those with disabilities. Draw attention to how this multi-modal approach improves accessibility and user experience in comparison to current systems that might only use one input modality.

Contribution

Team member	Project part	Contribution
Revathi Gollapudi	**Speech recognition**	**50%**
Sumana Madhireddy	**Speech recognition**	**50%**
Revathi Gollapudi	**Data preprocessing**	**50%**
Sumana Madhireddy	**Data preprocessing**	**50%**
Revathi Gollapudi	**Model building**	**50%**
Sumana Madhireddy	**Model building**	**50%**
Revathi Gollapudi	**Training the model**	**50%**
Sumana Madhireddy	**Training the model**	**50%**

References:

- **https://pytorch.org/hub/huggingface_pytorch-transformers/**
- **https://pytorch.org/tutorials/beginner/transformer_tutorial.html**
- Project is based on the CSE 676 deep learning Assignment 1 by Sumana Madhireddy and Revathi Gollapudi.
- Lecture notes.