# Introduction:

Language generation models have emerged as a cornerstone of natural language processing (NLP), enabling machines to produce human-like text based on given inputs. These models are designed to learn patterns, structures, and semantics from textual data, empowering them to generate coherent and contextually relevant sentences. Applications of language generation models span a wide range, including chatbots, automated content creation, translation systems, and more.

In this project, we focus on developing and training two distinct architectures for language generation: a custom Transformer-based model and a Recurrent Neural Network (RNN) model using a bidirectional Long Short-Term Memory (Bi-LSTM) network. Each architecture offers unique advantages in handling the complexities of language modeling. Transformers, with their attention mechanisms, excel at capturing long-range dependencies in text, while Bi-LSTMs leverage sequential processing to effectively learn contextual information from both past and future states.

The dataset for this project was meticulously curated through web scraping from the Higher Education Commission (HEC) and Government of Pakistan websites. This diverse and domain-specific dataset provides a rich foundation for training models tailored to generate relevant and meaningful text. To further enhance the performance, we incorporated both custom vector embeddings derived from the dataset and pre-trained GloVe-200 embeddings, which offer a robust representation of words by encoding semantic and syntactic relationships.

Evaluation of the trained models was performed using standard metrics, including accuracy and BLEU scores. These metrics provide insights into the models' ability to generate text that aligns closely with the input context and ground truth, ensuring the generated output is both syntactically correct and semantically relevant.

This work not only demonstrates the potential of combining custom data with advanced architectures but also underscores the value of leveraging pre-trained embeddings in achieving superior performance in language generation tasks.

# Dataset:

We have used the dataset which we scrapped in our assignment 1 and made embeddings of it in assignment 2. We used the website

1. Higher education Commission (HEC)
2. Govt of Pakistan

We merged our data sets and made a CSV file of it.

# Part a:

## RNN architecture

We used **Bi-Directional LSTM** for the RNN architecture part.

A Bi-Directional Long Short-Term Memory (Bi-LSTM) is an extension of the standard LSTM architecture, designed to improve the model's ability to capture context in sequential data by processing the input in both forward and backward directions.

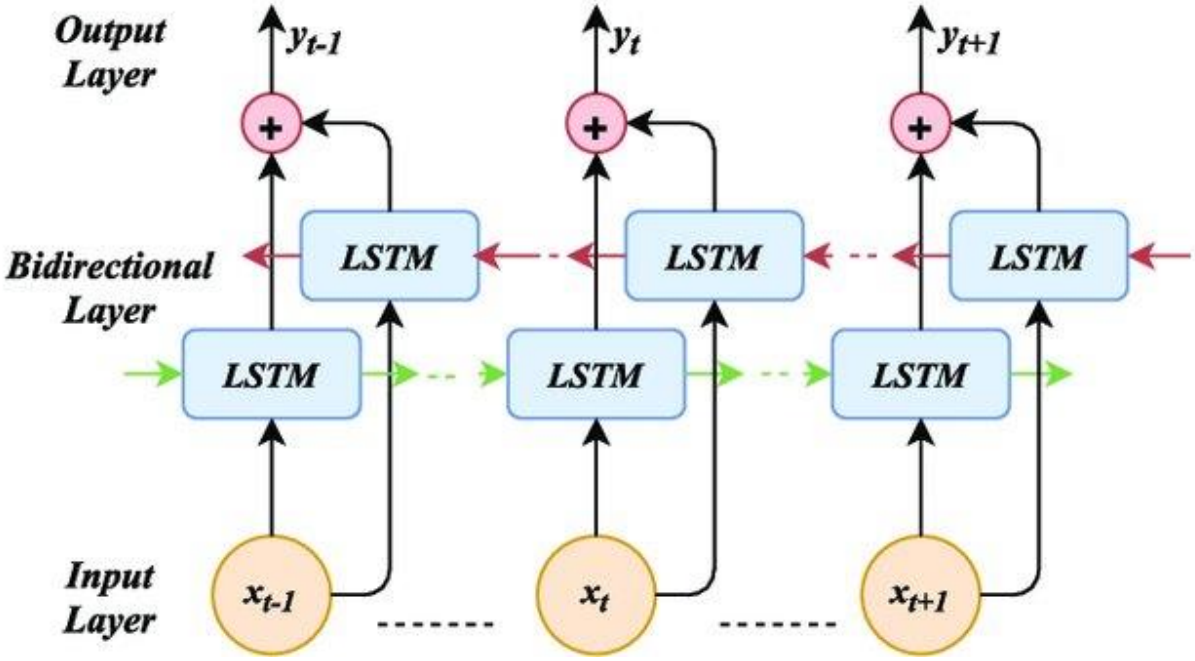



*Figure 1 Bidirectional LSTM model showing the input and output layers. The red arrows represent the backward sequence track and green the forward sequence track.*

### Workflow and Explanation of the Model

## 1. Data Preparation

- A CSV file containing sentences is loaded using Pandas.
- The dataset is inspected for the number of records and fields.
- Sentences are preprocessed to remove non-standard characters (e.g., `\xa0` and `\u200a`).
- Tokenization is applied using Keras' `Tokenizer` with an out-of-vocabulary token (`<oov>`). This converts text into sequences of integer indices.
- The total vocabulary size is computed based on the tokenizer's word index.

## 2. Sequence Generation

- N-gram sequences are generated from tokenized text. Each sequence includes progressively larger portions of the sentence.
- The sequences are padded to ensure uniform length using `pad_sequences`.
- Input features (`xs`) and target labels (`ys`) are created from the padded sequences. Targets are one-hot encoded using `to_categorical`.

## 3. Model Architecture

The model is a Bidirectional LSTM-based sequential model designed for next-word prediction tasks. Below are the details of the architecture:

1. **Embedding Layer**:
    - Maps integer token IDs to dense vectors of dimension 100.
    - Provides a dense representation for each word in the vocabulary.
2. **Bidirectional LSTM**:
    - Utilizes a 150-unit Long Short-Term Memory (LSTM) layer wrapped in a bidirectional configuration.
    - Captures both forward and backward dependencies in the sequence.
3. **Dense Output Layer**:
    - A fully connected layer with a SoftMax activation function predicts the next word's probability distribution across the vocabulary.
4. **Compilation**:
    - The model uses the Adam optimizer with a learning rate of 0.01.
    - The loss function is categorical cross-entropy, and accuracy is used as the evaluation metric.

## 4. Model Training

- The model is trained on the prepared input and target data (`xs` and `ys`) for **25 epochs.**
- Training history is visualized using Matplotlib to monitor loss and accuracy trends over epochs.

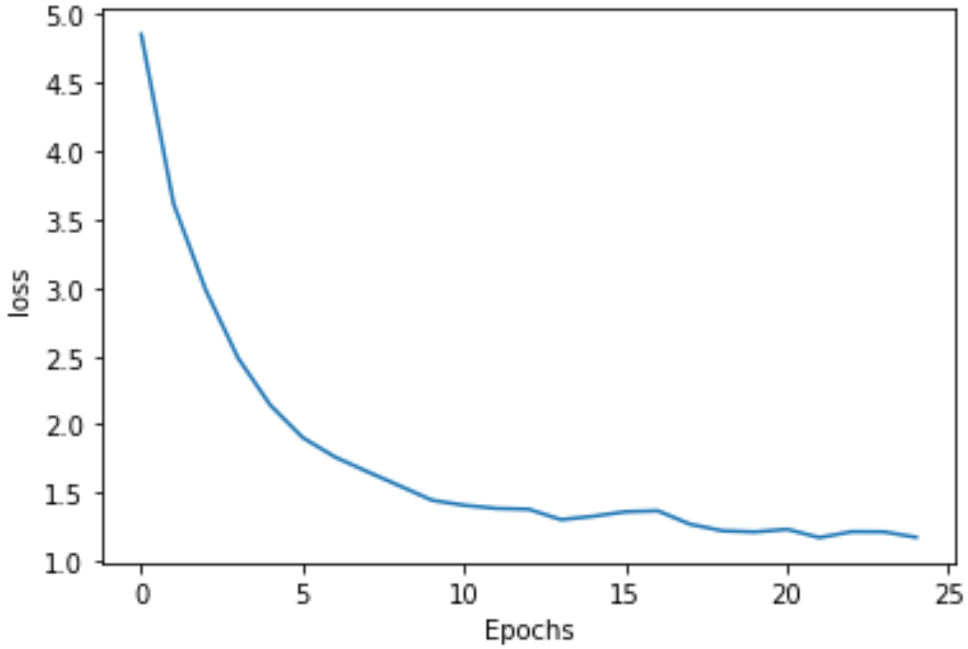



*Figure 2 Loss over the epochs*

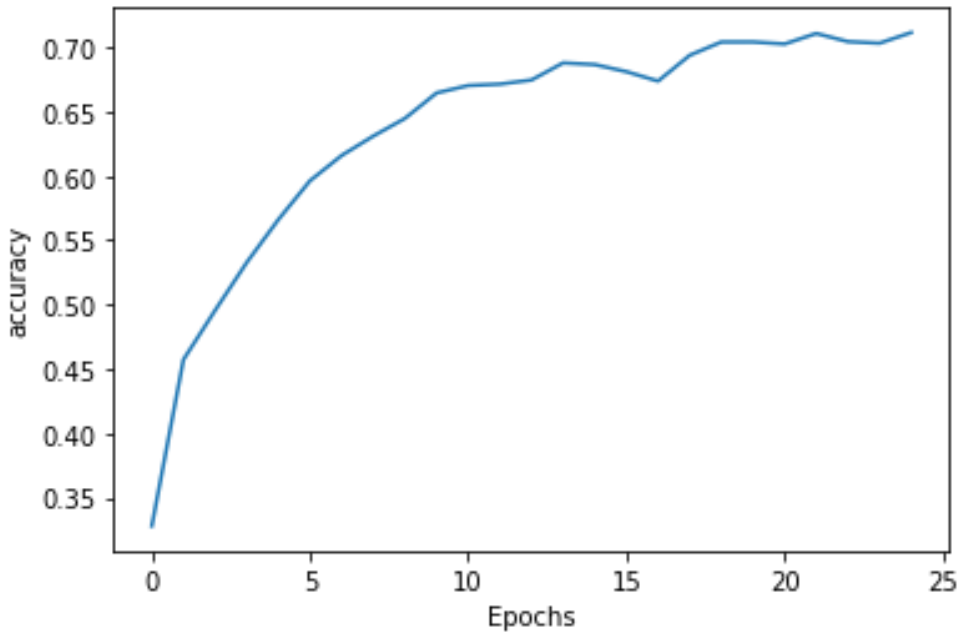



*Figure 3 Accuracy over the epochs*

## 5. Text Generation

- A seed text is provided as input to the model for generating new words.
- The model predicts the next word iteratively, appending it to the seed text to produce a generated sentence.

## 6. Evaluation Metrics

- **BLEU Score**:
  - Measures the similarity between the generated text and a reference text using n-gram overlap.
- **ROUGE Scores**:
  - Calculates precision, recall, and F1 scores for unigrams (ROUGE-1), bigrams (ROUGE-2), and longest common subsequences (ROUGE-L).

### Results of Model 1

1. BLEU Score: 0.5072
2. ROUGE Scores:

ROUGE1:
- Precision: 0.6875
- Recall: 1.0
- F1 Score: 0.8148

ROUGE2:
- Precision: 0.6667
- Recall: 1.0
- F1 Score: 0.8

ROUGEL:
- Precision: 0.6875
- Recall: 1.0
- F1 Score: 0.8148

### Results of Model 2

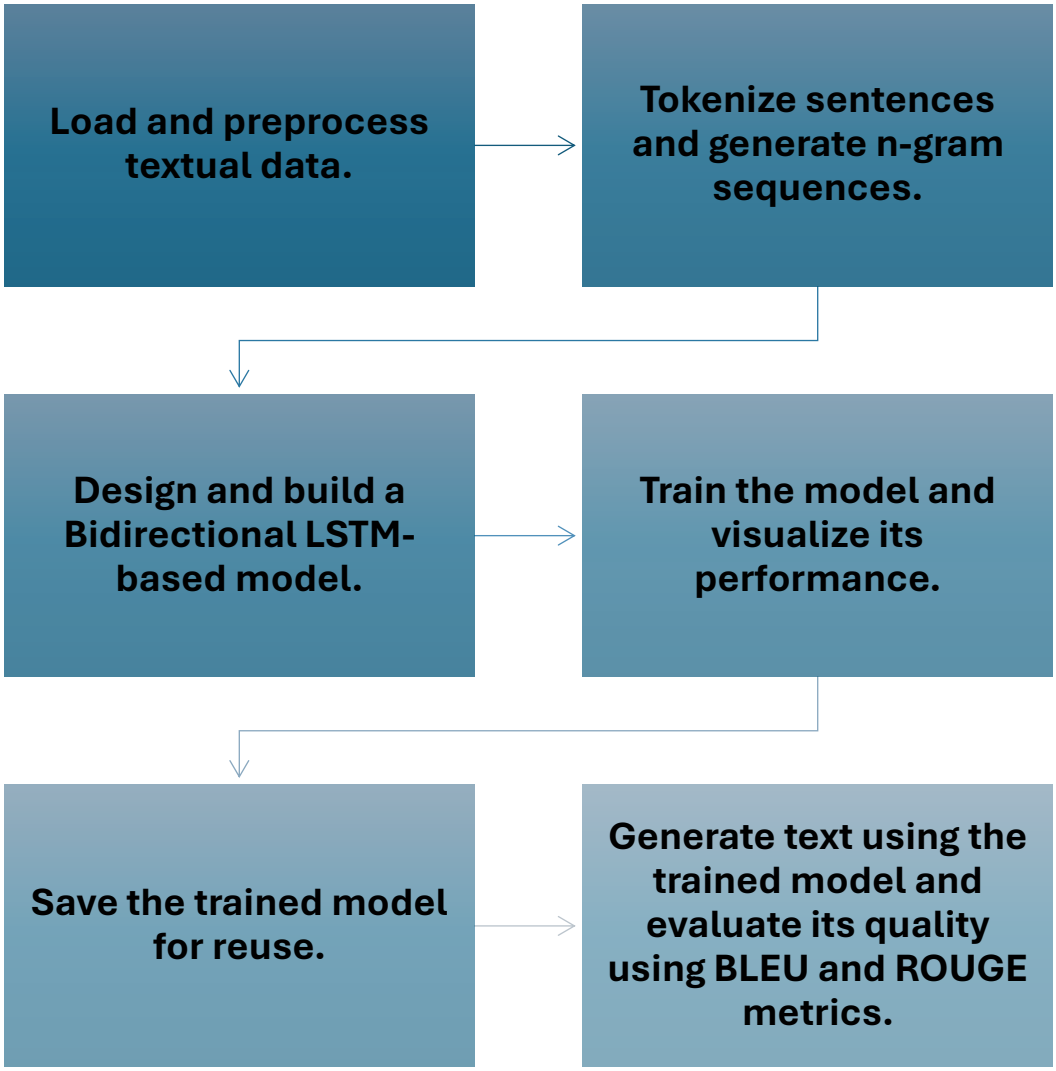
1. BLEU Score: 0.3389
2. ROUGE Scores:

ROUGE1:
- Precision: 0.6
- Recall: 0.8182
- F1 Score: 0.6923

ROUGE2:
- Precision: 0.5714
- Recall: 0.8
- F1 Score: 0.6667

ROUGEL:
- Precision: 0.6
- Recall: 0.8182
- F1 Score: 0.6923

8. Summary of the Workflow:



# Part b

# Transformer architecture

## Workflow and Explanation of the Model

## 1. Data Preparation

- The dataset is loaded using Pandas from a CSV file.
- Text data is extracted and previewed to ensure proper loading.
- The `TextVectorization` layer is configured to tokenize and vectorize the text data, converting it into a sequence of integer tokens.
- A TensorFlow dataset is created and passed to the vectorizer for adaptation.
- A helper function, `create_sequences`, is defined to generate input and target sequences from the tokenized text for sequence modeling.
- Input (`X`) and target (`Y`) sequences are created, validated for non-emptiness, and converted to TensorFlow tensors.

## 2. Model Architecture

The model is a small Transformer-based sequence model designed for text prediction tasks. Below are the details of the architecture:

1. **Embedding Layer**:
   - Maps integer token IDs to dense vectors of a specified embedding dimension (`embed_dim`).
2. **Positional Encoding**:
   - A custom layer adds positional information to embeddings. It uses sine and cosine functions to encode positions, enabling the model to consider word order.
3. **Transformer Blocks**:
   - Composed of the following components:
     - **Multi-Head Attention**: Captures relationships between words in the sequence.
     - **Dropout**: Adds regularization to prevent overfitting.
     - **Layer Normalization**: Stabilizes the training process.
     - **Feedforward Neural Network (FFN)**: Applies dense layers for feature transformation.
4. **Final Output Layer**:
   - A dense layer with a SoftMax activation function outputs probabilities over the vocabulary for the next token prediction.

## 3. Hyperparameters

| | |
|---|---|
| **Embedding Dimension:** | **128** |
| **Number of Attention Heads:** | 2 |
| **Feedforward Dimension:** | 128 |
| **Number of Transformer Layers:** | 2 |
| **Dropout Rate:** | 0.1 |

## 4. Model Training

- The dataset is split into training and validation sets using Scikit-learn's `train_test_split`.
- Early stopping is implemented to halt training if the loss does not improve for a specified number of epochs.
- The model is compiled with:
  - Optimizer: Adam
  - Loss: Sparse categorical cross-entropy
  - Metric: Accuracy
  - Training is performed for up to 20 epochs with a batch size of 32.

## 5. Visualization of Training

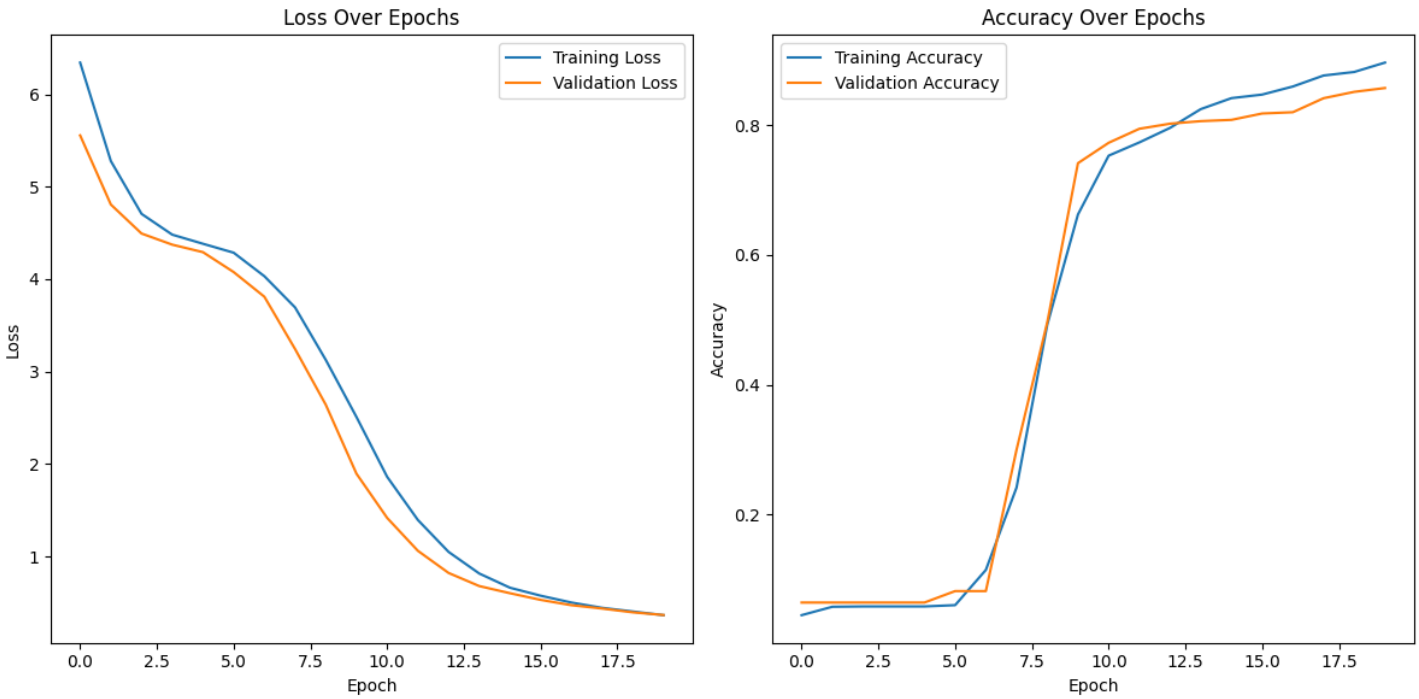



*Figure 4 Training and Validation loss and accuracy of Model using Assignment 2 embeddings*

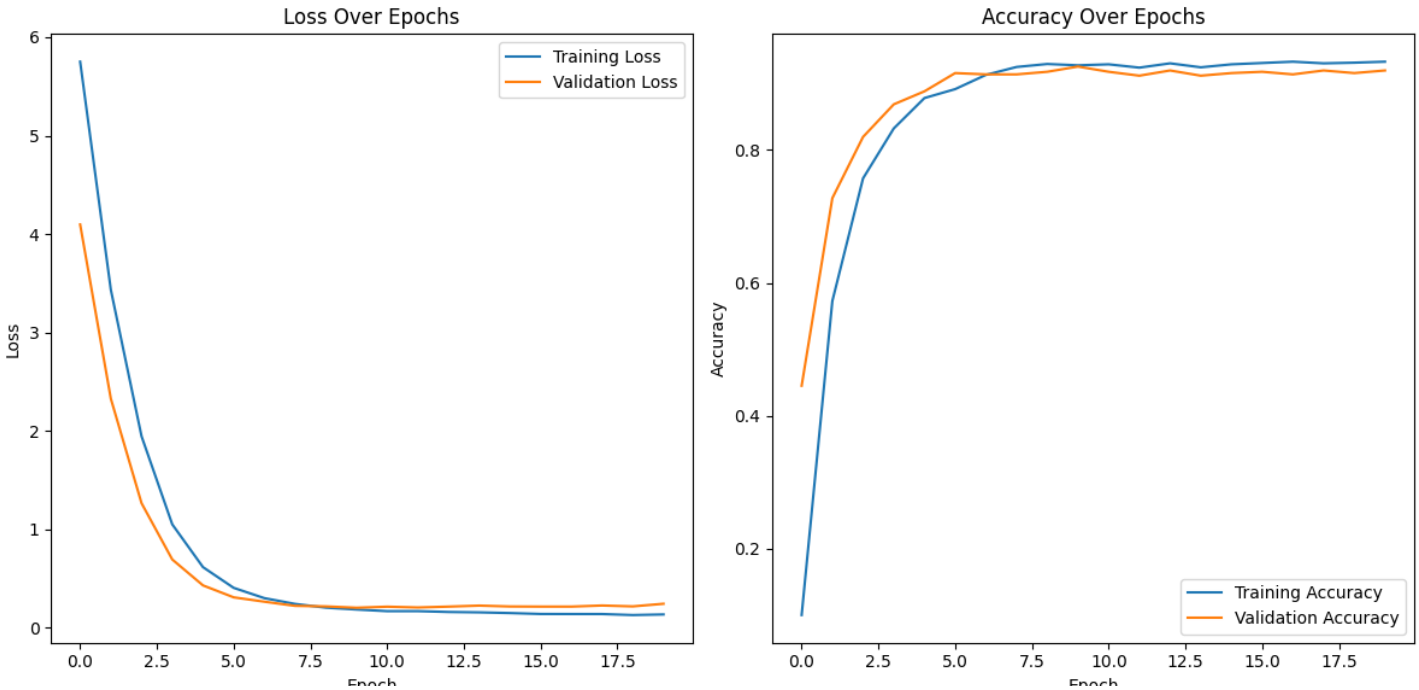
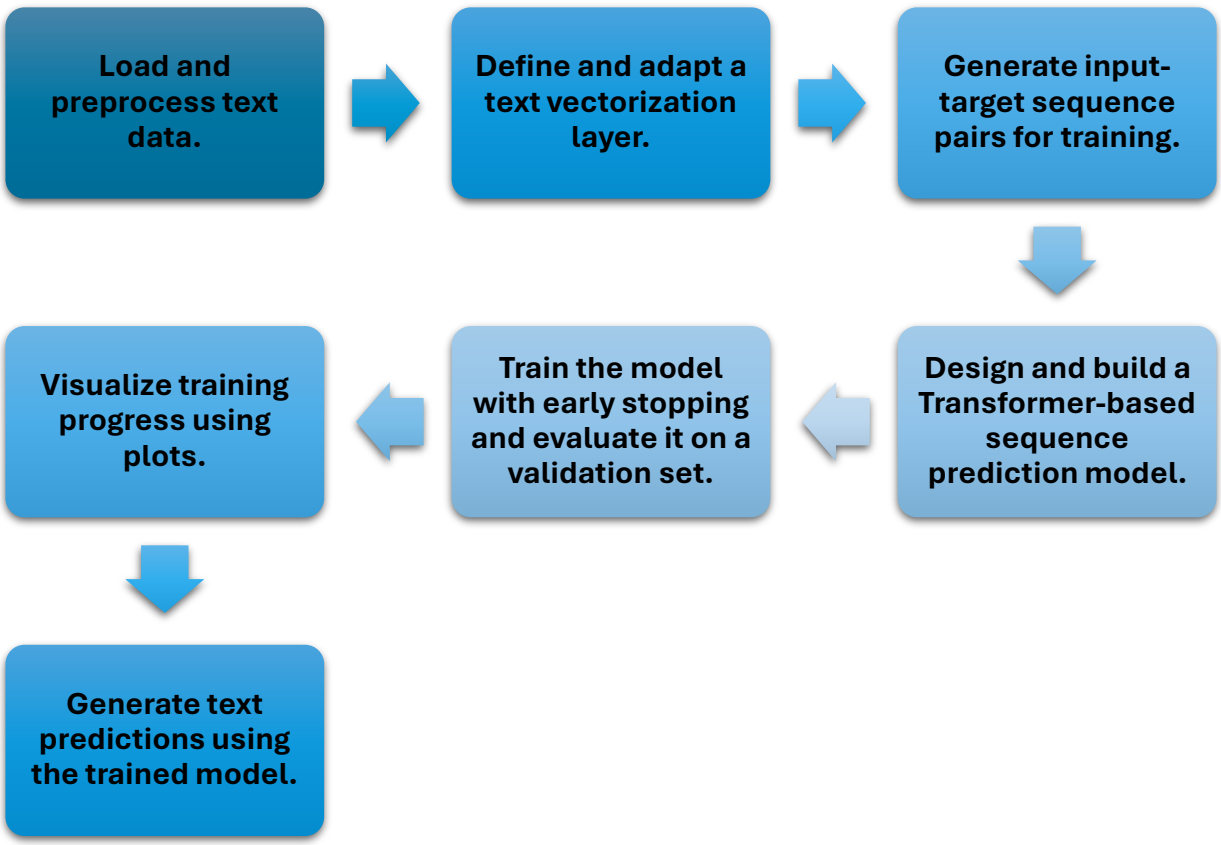



*Figure 5 Training and Validation loss and accuracy of Model using Word2Vec 2 embeddings*

## 6. Text Generation

- A function, `generate_text`, is implemented to generate text based on the trained model.
- The function:
    - Take a seed string and tokenizes it.
    - Iteratively predicts the next token using the model.
    - Applies a temperature parameter to control randomness in token sampling.
    - Outputs the generated text as a string.

## Summary of the Workflow:



## Difficulties:

- Making the dataset that we can give to our Models
- Training in the LSTM Part was very difficult.

## References:

Ihianle, I. K., Nwajana, A. O., Ebenuwa, S. H., Otuka, R. I., Owa, K., & Orisatoki, M. O. (2020). A deep learning approach for human activities recognition from multimodal sensing devices. *IEEE Access*, *8*, 179028-179038.

Vaswani, A. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.