```
#COMPONENT 1--->RNN / LSTM Based Text Generation (Character-Level)
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
```

```
text = """
artificial intelligence is transforming modern society.
it is used in healthcare finance education and transportation.
machine learning allows systems to improve automatically with experience.
data plays a critical role in training intelligent systems.
large datasets help models learn complex patterns.
deep learning uses multi layer neural networks.
neural networks are inspired by biological neurons.
each neuron processes input and produces an output.
training a neural network requires optimization techniques.
gradient descent minimizes the loss function.

natural language processing helps computers understand human language.
text generation is a key task in nlp.
language models predict the next word or character.
recurrent neural networks handle sequential data.
lstm and gru models address long term dependency problems.
however rnn based models are slow for long sequences.
"""
```

```
chars = sorted(list(set(text)))
char_to_idx = {c:i for i,c in enumerate(chars)}
idx_to_char = {i:c for i,c in enumerate(chars)}

vocab_size = len(chars)
print("Vocabulary size:", vocab_size)
```

```
Vocabulary size: 28
```

```
seq_length = 40
X = []
y = []

for i in range(len(text) - seq_length):
    X.append([char_to_idx[c] for c in text[i:i+seq_length]])
    y.append(char_to_idx[text[i+seq_length]])

X = np.array(X)
y = np.array(y)
```

```
model = Sequential([
    Embedding(vocab_size, 64, input_length=seq_length),
    LSTM(128),
    Dense(vocab_size, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam')

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_lengt
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
model.fit(X, y, epochs=20, batch_size=64)
```

```
Epoch 1/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - loss: 3.3015
Epoch 2/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 3.0306
Epoch 3/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 98ms/step - loss: 2.9970
Epoch 4/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 113ms/step - loss: 2.9437
Epoch 5/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.9422
Epoch 6/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.9247
Epoch 7/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.9169
Epoch 8/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.8922
Epoch 9/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.8426
Epoch 10/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 2.8158
Epoch 11/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - loss: 2.7246
Epoch 12/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - loss: 2.6726
Epoch 13/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 2.5698
Epoch 14/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.5497
Epoch 15/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 104ms/step - loss: 2.4611
Epoch 16/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 109ms/step - loss: 2.4489
Epoch 17/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - loss: 2.4333
Epoch 18/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.3531
Epoch 19/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 2.2766
Epoch 20/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 2.2246
<keras.src.callbacks.history.History at 0x7c01de173fb0>
```

```
def generate_text(seed, length=300):
    result = seed
    for _ in range(length):
        input_seq = np.array([[char_to_idx[c] for c in seed]])
        prediction = model.predict(input_seq, verbose=0)
        next_char = idx_to_char[np.argmax(prediction)]
        result += next_char
        seed = seed[1:] + next_char
    return result

print(generate_text("artificial intelligence is t"))
```

```
artificial intelligence is toron ine lers ane neres neres ne neres neres ne neres neres ne neres neres ne neres n
```

```
#COMPONENT 2---->Transformer Based Text Generation (Word-Level)
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1
```

```
input_sequences = []

for line in text.split("\n"):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        input_sequences.append(token_list[:i+1])

max_len = max(len(seq) for seq in input_sequences)
input_sequences = pad_sequences(input_sequences, maxlen=max_len)

X = input_sequences[:, :-1]
```

```
y = input_sequences[:, -1]
```

```
from tensorflow.keras.layers import Layer, MultiHeadAttention, LayerNormalization, Dropout
```

```
class TransformerBlock(Layer):
    def __init__(self, embed_dim, num_heads, ff_dim):
        super().__init__()
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = Sequential([
            Dense(ff_dim, activation="relu"),
            Dense(embed_dim)
        ])
        self.norm1 = LayerNormalization()
        self.norm2 = LayerNormalization()

    def call(self, inputs):
        attn_output = self.att(inputs, inputs)
        out1 = self.norm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        return self.norm2(out1 + ffn_output)
```

```
embed_dim = 64

inputs = tf.keras.Input(shape=(max_len-1,))
x = Embedding(total_words, embed_dim)(inputs)
x = TransformerBlock(embed_dim, num_heads=2, ff_dim=128)(x)
x = tf.keras.layers.GlobalAveragePooling1D()(x)
outputs = Dense(total_words, activation="softmax")(x)

transformer_model = tf.keras.Model(inputs, outputs)
transformer_model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="adam"
)

transformer_model.summary()
```

Model: "functional_2"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 8)	0
embedding_1 (Embedding)	(None, 8, 64)	6,336
transformer_block (TransformerBlock)	(None, 8, 64)	50,048
global_average_pooling1d (GlobalAveragePooling1D)	(None, 64)	0
dense_3 (Dense)	(None, 99)	6,435

Total params: 62,819 (245.39 KB)
Trainable params: 62,819 (245.39 KB)
Non-trainable params: 0 (0.00 B)

```
transformer_model.fit(X, y, epochs=30, batch_size=32)
```

```
Epoch 1/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 4.7818
Epoch 2/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 4.3700
Epoch 3/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 4.2650
Epoch 4/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 4.0577
Epoch 5/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 3.9308
Epoch 6/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 3.8427
Epoch 7/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 3.7212
Epoch 8/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 3.6222
Epoch 9/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 3.5471
Epoch 10/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 3.3735
```

```
Epoch 11/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 3.2775
Epoch 12/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 3.2339
Epoch 13/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 3.1062
Epoch 14/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 3.0169
Epoch 15/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 2.8770
Epoch 16/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 2.8020
Epoch 17/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2.6815
Epoch 18/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 2.5713
Epoch 19/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 2.4307
Epoch 20/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2.2933
Epoch 21/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2.1985
Epoch 22/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2.0743
Epoch 23/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 1.9682
Epoch 24/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 1.8663
Epoch 25/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 1.7194
Epoch 26/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 1.6840
Epoch 27/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 1.6317
Epoch 28/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 1.5649
Epoch 29/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 1.5055
```