```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Layer, Embedding, Dense, LayerNormalization, Dropout
from tensorflow.keras.models import Model
```

```
data = """
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.

transformer models changed the field of nlp.
they rely on self attention mechanisms.
attention allows the model to focus on relevant context.

education is being improved using artificial intelligence.
intelligent tutoring systems personalize learning.

ethical considerations are important in artificial intelligence.
ai systems should be designed responsibly.

text generation models can create stories poems and articles.
generated text should be meaningful and coherent.

continuous learning is essential in the field of ai.
programming skills are important for ai engineers.
"""
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([data])

total_words = len(tokenizer.word_index) + 1

input_sequences = []

for line in data.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])
```

```python
max_len = max(len(x) for x in input_sequences)

input_sequences = pad_sequences(input_sequences, maxlen=max_len, padding='pre')

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

y = tf.keras.utils.to_categorical(y, num_classes=total_words)

print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (167, 8)
y shape: (167, 134)
```

```python
class PositionalEncoding(Layer):
    def __init__(self, max_len, d_model):
        super().__init__()
        pos = np.arange(max_len)[:, np.newaxis]
        i = np.arange(d_model)[np.newaxis, :]
        angle_rates = 1 / np.power(10000, (2*(i//2))/np.float32(d_model))
        angle_rads = pos * angle_rates

        angle_rads[:,0::2] = np.sin(angle_rads[:,0::2])
        angle_rads[:,1::2] = np.cos(angle_rads[:,1::2])

        self.pos_encoding = tf.cast(angle_rads[np.newaxis,...], dtype=tf.float32)

    def call(self, x):
        return x + self.pos_encoding[:,:tf.shape(x)[1],:]
```

```python
class TransformerBlock(Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super().__init__()
        self.att = tf.keras.layers.MultiHeadAttention(num_heads=num_heads,
                                                      key_dim=embed_dim)
        self.ffn = tf.keras.Sequential([
            Dense(ff_dim, activation="relu"),
            Dense(embed_dim),
        ])
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)

    def call(self, inputs, training=False):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)

        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

```
embed_dim = 64
num_heads = 2
ff_dim = 128

inputs = tf.keras.Input(shape=(max_len-1,))

x = Embedding(total_words, embed_dim)(inputs)
x = PositionalEncoding(max_len, embed_dim)(x)

x = TransformerBlock(embed_dim, num_heads, ff_dim)(x)

x = tf.keras.layers.GlobalAveragePooling1D()(x)

outputs = Dense(total_words, activation="softmax")(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])

model.summary()
```

```
Model: "functional_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)        │ (None, 8)              │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ embedding (Embedding)           │ (None, 8, 64)          │         8,576 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ positional_encoding             │ (None, 8, 64)          │             0 │
│ (PositionalEncoding)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ transformer_block               │ (None, 8, 64)          │        50,048 │
│ (TransformerBlock)              │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling1d        │ (None, 64)             │             0 │
│ (GlobalAveragePooling1D)        │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 134)            │         8,710 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 67,334 (263.02 KB)
 Trainable params: 67,334 (263.02 KB)
```

```
history = model.fit(X, y, epochs=100, verbose=1)
```

```
Epoch 78/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9911 - loss: 0.2970
Epoch 79/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9911 - loss: 0.2633
Epoch 80/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9855 - loss: 0.2776
Epoch 81/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9829 - loss: 0.2566
Epoch 82/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9888 - loss: 0.2456
Epoch 83/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9922 - loss: 0.2475
Epoch 84/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9900 - loss: 0.2374
Epoch 85/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9948 - loss: 0.2191
Epoch 86/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9762 - loss: 0.2313
Epoch 87/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9900 - loss: 0.1992
Epoch 88/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9870 - loss: 0.1963
Epoch 89/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9888 - loss: 0.1915
Epoch 90/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9974 - loss: 0.1920
Epoch 91/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9974 - loss: 0.1759
Epoch 92/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9911 - loss: 0.1718
Epoch 93/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9818 - loss: 0.1665
Epoch 94/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9963 - loss: 0.1535
Epoch 95/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9908 - loss: 0.1657
Epoch 96/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9874 - loss: 0.1896
Epoch 97/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9948 - loss: 0.1720
Epoch 98/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9911 - loss: 0.1623
Epoch 99/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9874 - loss: 0.1575
Epoch 100/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9888 - loss: 0.1292
```

```python
def generate_transformer_text(seed_text, next_words=20):

    for _ in range(next_words):

        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_len-1, padding='pre')

        predicted = model.predict(token_list, verbose=0)
        predicted_word = tokenizer.index_word[np.argmax(predicted)]

        seed_text += " " + predicted_word

    return seed_text
```

```
print(generate_transformer_text("artificial intelligence", 25))
```

```
artificial intelligence is transforming modern society used healthcare finance education and transportation transportation transportation long term dependency problems
```

```
import gradio as gr

def transformer_ui(seed_text, length):
    return generate_transformer_text(seed_text, int(length))

demo = gr.Interface(
    fn=transformer_ui,
    inputs=[
        gr.Textbox(
            label="Enter Seed Text",
            value="artificial intelligence",
            max_lines=5
        ),
        gr.Slider(
            minimum=5,
            maximum=50,
            value=20,
            step=1,
            label="Number of Words"
        )
    ],
    outputs=gr.Textbox(
        label="Generated Text",
        lines=15,
        max_lines=25,
        show_copy_button=True
    ),
    title="Transformer Text Generation UI",
    description="GenAI Lab-4 Component-II – Transformer Based Text Generation"
)

demo.launch(debug=True)
```

```
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (you can turn this off by setting `sh

Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug=False in launch().
* Running on public URL: https://7967ee96637b184e43.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face
```

Transformer Text Generation UI

GenAI Lab-4 Component-II — Transformer Based Text Generation

Enter Seed Text

artificial intelligence

Number of Words 6

5 50

Clear Submit

Generated Text

artificial intelligence is transforming modern society used healthcare

Next steps: Deploy to Cloud Run

Start coding or generate with AI.