Implement a Transformer-based Language Model for text generation

```
# Install required libraries
!pip install tensorflow numpy
```

1. Import Libraries and Sample Data

```
import tensorflow as tf
import numpy as np

# Sample tiny corpus
sentences = [
    "hello world",
    "how are you",
    "hello how are you",
    "hello you",
    "are you there"
]
```

2. Tokenization & Padding

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

tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)

sequences = tokenizer.texts_to_sequences(sentences)
max_len = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post')

vocab_size = len(tokenizer.word_index) + 1
```

3. Prepare Input and Target

```
X = padded_sequences[:, :-1]  # All except last token
y = padded_sequences[:, 1:]   # All except first token

y = tf.keras.utils.to_categorical(y, num_classes=vocab_size)
```

4. Build a Mini Transformer Model

```
from tensorflow.keras.layers import Input, Embedding, Dense, LayerNormalization, Dropout, MultiHeadAttention
from tensorflow.keras.models import Model

embed_dim = 64
num_heads = 2
ff_dim = 128

input_seq = Input(shape=(X.shape[1],))
embedding_layer = Embedding(input_dim=vocab_size, output_dim=embed_dim)(input_seq)

# Multi-Head Attention
attn_output = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)(embedding_layer, embedding_layer)
attn_output = Dropout(0.1)(attn_output)
out1 = LayerNormalization(epsilon=1e-6)(embedding_layer + attn_output)

# Feed-Forward
ffn = Dense(ff_dim, activation='relu')(out1)
ffn = Dense(embed_dim)(ffn)
```

```
ffn_output = Dropout(0.1)(ffn)
out2 = LayerNormalization(epsilon=1e-6)(out1 + ffn_output)

# Output layer
final_output = Dense(vocab_size, activation='softmax')(out2)

model = Model(inputs=input_seq, outputs=final_output)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)         ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer          │ (None, 3)         │          0 │ -                 │
│ (InputLayer)         │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ embedding            │ (None, 3, 64)     │        448 │ input_layer[0][0] │
│ (Embedding)          │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ multi_head_attenti…  │ (None, 3, 64)     │     33,216 │ embedding[0][0],  │
│ (MultiHeadAttentio…  │                   │            │ embedding[0][0]   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_1 (Dropout)  │ (None, 3, 64)     │          0 │ multi_head_atten… │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ add (Add)            │ (None, 3, 64)     │          0 │ embedding[0][0],  │
│                      │                   │            │ dropout_1[0][0]   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ layer_normalization  │ (None, 3, 64)     │        128 │ add[0][0]         │
│ (LayerNormalizatio…  │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense (Dense)        │ (None, 3, 128)    │      8,320 │ layer_normalizat… │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_1 (Dense)      │ (None, 3, 64)     │      8,256 │ dense[0][0]       │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_2 (Dropout)  │ (None, 3, 64)     │          0 │ dense_1[0][0]     │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ add_1 (Add)          │ (None, 3, 64)     │          0 │ layer_normalizat… │
│                      │                   │            │ dropout_2[0][0]   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ layer_normalizatio…  │ (None, 3, 64)     │        128 │ add_1[0][0]       │
│ (LayerNormalizatio…  │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_2 (Dense)      │ (None, 3, 7)      │        455 │ layer_normalizat… │
└──────────────────────┴───────────────────┴────────────┴───────────────────┘
 Total params: 50,951 (199.03 KB)
 Trainable params: 50,951 (199.03 KB)
 Non-trainable params: 0 (0.00 B)
```

5. Train the Model

```
model.fit(X, y, batch_size=2, epochs=100, verbose=0)
print("Model Trained.")
```

```
✅ Model Trained.
```

6. Generate Text (Prediction Function)

```
def generate_text(seed_text, next_words=5):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=X.shape[1], padding='post')
        predicted_probs = model.predict(token_list, verbose=0)
        predicted_id = np.argmax(predicted_probs[0][-1])  # Get next word
        output_word = tokenizer.index_word.get(predicted_id, '')
        seed_text += " " + output_word
    return seed_text

print(generate_text("hello"))
```

```
hello
```

Start coding or generate with AI.