🔹 Introduction (Medium Summary)

Transformer models use Self-Attention mechanism instead of recurrence.

Unlike RNN/LSTM:

No sequential processing

Processes words in parallel

Captures long-range dependencies efficiently

It uses:

Embedding

Positional Encoding

Multi-Head Attention

Feed Forward Network

Transformers are the foundation of modern models like GPT and BERT.

```
# ========================================
# TRANSFORMER BASED TEXT GENERATION
# ========================================

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

# -------------------------
# 1. Load Dataset
# -------------------------

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
"""

# -------------------------
# 2. Tokenization
# -------------------------

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)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

max_seq_len = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_seq_len, padding='pre')

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

# -------------------------
# 3. Positional Encoding
# -------------------------

def positional_encoding(length, depth):
    depth = depth / 2
    positions = np.arange(length)[:, np.newaxis]
```

```
    depths = np.arange(depth)[np.newaxis, :] / depth
    angle_rates = 1 / (10000**depths)
    angle_rads = positions * angle_rates
    pos_encoding = np.concatenate(
        [np.sin(angle_rads), np.cos(angle_rads)],
        axis=-1)
    return tf.cast(pos_encoding, dtype=tf.float32)

# --------------------------
# 4. Build Transformer Model
# --------------------------

embed_dim = 64
num_heads = 2
ff_dim = 128

inputs = Input(shape=(max_seq_len-1,))
embedding_layer = Embedding(total_words, embed_dim)(inputs)

pos_encoding = positional_encoding(max_seq_len-1, embed_dim)
x = embedding_layer + pos_encoding

# Multi-Head Attention
attention_output = MultiHeadAttention(
    num_heads=num_heads,
    key_dim=embed_dim
)(x, x)

x = LayerNormalization(epsilon=1e-6)(x + attention_output)

# Feed Forward Network
ffn = Dense(ff_dim, activation="relu")(x)
ffn = Dense(embed_dim)(ffn)

x = LayerNormalization(epsilon=1e-6)(x + ffn)

# Global pooling (take last token)
x = x[:, -1, :]

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

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

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

model.summary()

# --------------------------
# 5. Train Model
# --------------------------

model.fit(X, y, epochs=65, verbose=1)

# --------------------------
# 6. Text Generation
# --------------------------

def generate_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_seq_len-1, padding='pre')

        predicted = np.argmax(model.predict(token_list, verbose=0), axis=-1)

        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break

        seed_text += " " + output_word

    return seed_text

print("\nGenerated Text:\n")
print(generate_text("artificial intelligence", 20))
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 8)	0	-
embedding_1 (Embedding)	(None, 8, 64)	4,224	input_layer_1[0]…
add_3 (Add)	(None, 8, 64)	0	embedding_1[0][0]
multi_head_attenti… (MultiHeadAttentio…	(None, 8, 64)	33,216	add_3[0][0], add_3[0][0]
add_4 (Add)	(None, 8, 64)	0	add_3[0][0], multi_head_atten…
layer_normalizatio… (LayerNormalizatio…	(None, 8, 64)	128	add_4[0][0]
dense_3 (Dense)	(None, 8, 128)	8,320	layer_normalizat…
dense_4 (Dense)	(None, 8, 64)	8,256	dense_3[0][0]
add_5 (Add)	(None, 8, 64)	0	layer_normalizat… dense_4[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 8, 64)	128	add_5[0][0]
get_item_1 (GetItem)	(None, 64)	0	layer_normalizat…
dense_5 (Dense)	(None, 66)	4,290	get_item_1[0][0]

```
 Total params: 58,562 (228.76 KB)
 Trainable params: 58,562 (228.76 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.0000e+00 - loss: 4.5214
Epoch 2/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.0194 - loss: 4.2896
Epoch 3/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0310 - loss: 4.1935
Epoch 4/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0232 - loss: 4.2511
Epoch 5/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0504 - loss: 4.1800
Epoch 6/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0310 - loss: 4.1484
Epoch 7/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0310 - loss: 4.1165
Epoch 8/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.0310 - loss: 4.0948
Epoch 9/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0310 - loss: 4.0877
Epoch 10/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.1008 - loss: 4.0233
Epoch 11/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0620 - loss: 4.0455
Epoch 12/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0620 - loss: 4.0382
Epoch 13/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0930 - loss: 4.0064
Epoch 14/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.1124 - loss: 3.9917
Epoch 15/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0193 - loss: 4.0360
Epoch 16/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0388 - loss: 4.0432
Epoch 17/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0310 - loss: 4.0968
Epoch 18/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0388 - loss: 3.9983
Epoch 19/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0388 - loss: 3.9616
Epoch 20/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0426 - loss: 3.9444
Epoch 21/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0116 - loss: 3.9474
Epoch 22/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0426 - loss: 3.9454
Epoch 23/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0310 - loss: 3.8705
Epoch 24/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0310 - loss: 3.8508
Epoch 25/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0504 - loss: 3.8137
Epoch 26/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0736 - loss: 3.8010
```

- Limitations
  - ❌ Requires more computation
  - ❌ Needs larger data for best performance
  - ❌ Complex architecture
  - ❌ Memory heavy

```
Epoch 27/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.0504 - loss: 3.8077
Epoch 28/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.1008 - loss: 3.8078
Epoch 29/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0971 - loss: 3.8473
Epoch 30/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0310 - loss: 3.8707
Epoch 31/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.0426 - loss: 3.8484
Epoch 32/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0387 - loss: 3.8594
Epoch 33/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0504 - loss: 3.7810
Epoch 34/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0194 - loss: 3.7511
Epoch 35/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0116 - loss: 3.6341
Epoch 36/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0930 - loss: 3.5345
Epoch 37/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.0851 - loss: 3.5255
Epoch 38/65
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.1240 - loss: 3.4825
```