```
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.utils import to_categorical

raw_lab_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.
however rnn based models are slow for long sequences.
transformer models changed the field of nlp.
they rely on self attention mechanisms.
attention allows the model to focus on relevant context.
transformers process data in parallel.
this makes training faster and more efficient.
modern language models are based on transformers.
education is being improved using artificial intelligence.
intelligent tutoring systems personalize learning.
automated grading saves time for teachers.
online education platforms use recommendation systems.
technology enhances the quality of learning experiences.
ethical considerations are important in artificial intelligence.
fairness transparency and accountability must be ensured.
ai systems should be designed responsibly.
data privacy and security are major concerns.
researchers continue to improve ai safety.
text generation models can create stories poems and articles.
they are used in chatbots virtual assistants and content creation.
generated text should be meaningful and coherent.
evaluation of text generation is challenging.
human judgement is often required.
continuous learning is essential in the field of ai.
research and innovation drive technological progress.
students should build strong foundations in mathematics.
programming skills are important for ai engineers.
```

```
practical experimentation enhances understanding.
"""

# Clean and split data
clean_data = [s.strip().lower() for s in raw_lab_data.split('\n') if
len(s) > 0]

import random

class SimpleNGram:
    def __init__(self, n_val=2):
        self.n_val = n_val
        self.chain = {}

    def fit(self, text_list):
        for line in text_list:
            words = line.split()
            # Loop through words to build the chain
            for i in range(len(words) - self.n_val):
                # Create the key (current state)
                key = tuple(words[i : i + self.n_val])
                target = words[i + self.n_val]

                if key not in self.chain:
                    self.chain[key] = []
                self.chain[key].append(target)

    def predict(self, start_text, length=10):
        # Prepare the initial key
        words = start_text.split()
        current_key = tuple(words[-self.n_val:])
        result_seq = list(current_key)

        for _ in range(length):
            # If the sequence breaks, stop
            if current_key not in self.chain:
                break

            # Pick a possible next word
            options = self.chain[current_key]
            choice = random.choice(options)

            result_seq.append(choice)
            # Update the key for the next iteration
            current_key = tuple(result_seq[-self.n_val:])

        return ' '.join(result_seq)

# Run N-Gram
print("--- N-Gram Results ---")
```

```
ngram_gen = SimpleNGram(n_val=2)
ngram_gen.fit(clean_data)
print(ngram_gen.predict("artificial intelligence"))

--- N-Gram Results ---
artificial intelligence is transforming modern society.

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# --- 1. Processing Data ---
# Initialize tokenizer
main_tokenizer = Tokenizer()
main_tokenizer.fit_on_texts(clean_data)
vocab_size = len(main_tokenizer.word_index) + 1

# Create sequences
sequences = []
for line in clean_data:
    encoded = main_tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(encoded)):
        sequences.append(encoded[:i+1])

# Padding
max_len = max([len(seq) for seq in sequences])
padded_seqs = np.array(pad_sequences(sequences, maxlen=max_len,
padding='pre'))

# Split X and Y
x_vals = padded_seqs[:, :-1]
y_vals = to_categorical(padded_seqs[:, -1], num_classes=vocab_size)

# [cite_start]--- 2. Model Architecture [cite: 25] ---
lstm_net = Sequential([
    Embedding(vocab_size, 64, input_length=max_len-1),
    LSTM(100), # Standard LSTM layer
    Dense(vocab_size, activation='softmax')
])

lstm_net.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
print(lstm_net.summary())

# --- 3. Training ---
print("Training LSTM Network...")
lstm_net.fit(x_vals, y_vals, epochs=100, verbose=0)

# --- 4. Generation Helper ---
def run_generator(seed, count, model_obj, seq_len):
    res_text = seed
```

```
    for _ in range(count):
        # Convert text to sequence
        tokens = main_tokenizer.texts_to_sequences([res_text])[0]
        tokens = pad_sequences([tokens], maxlen=seq_len-1,
padding='pre')

        # Predict class
        pred_idx = np.argmax(model_obj.predict(tokens, verbose=0),
axis=-1)

        # Find word from index
        pred_word = ""
        for word, idx in main_tokenizer.word_index.items():
            if idx == pred_idx:
                pred_word = word
                break
        res_text += " " + pred_word
    return res_text

print("\n--- LSTM Output ---")
print(run_generator("artificial intelligence", 5, lstm_net, max_len))
print(run_generator("neural networks", 6, lstm_net, max_len))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/
embedding.py:97: UserWarning: Argument `input_length` is deprecated.
Just remove it.
  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)

None
Training LSTM Network...

--- LSTM Output ---
artificial intelligence is transforming modern society society
neural networks are inspired by biological neurons sequences
```

```python
from tensorflow.keras import layers, models, Input

# --- Custom Layers ---

class EncoderLayer(layers.Layer):
    def __init__(self, d_model, heads, d_ff, dropout_rate=0.1):
        super(EncoderLayer, self).__init__()
        self.attention = layers.MultiHeadAttention(num_heads=heads,
key_dim=d_model)
        self.feed_forward = models.Sequential([
            layers.Dense(d_ff, activation="relu"),
            layers.Dense(d_model)
        ])
        self.norm1 = layers.LayerNormalization(epsilon=1e-6)
        self.norm2 = layers.LayerNormalization(epsilon=1e-6)
        self.drop1 = layers.Dropout(dropout_rate)
        self.drop2 = layers.Dropout(dropout_rate)

    def call(self, x):
        # Attention Sub-layer
        attn = self.attention(x, x)
        attn = self.drop1(attn)
        res1 = self.norm1(x + attn)

        # Feed Forward Sub-layer
        ff_out = self.feed_forward(res1)
        ff_out = self.drop2(ff_out)
        return self.norm2(res1 + ff_out)

class PositionalEmbedding(layers.Layer):
    def __init__(self, seq_len, vocab_s, embed_s):
        super(PositionalEmbedding, self).__init__()
        self.token_embeddings = layers.Embedding(input_dim=vocab_s,
output_dim=embed_s)
        self.pos_embeddings = layers.Embedding(input_dim=seq_len,
output_dim=embed_s)

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
```

```
        embedded_pos = self.pos_embeddings(positions)
        embedded_tokens = self.token_embeddings(inputs)
        return embedded_tokens + embedded_pos

# [cite_start]--- Transformer Setup [cite: 79] ---
embed_size = 64
head_count = 4
feed_forward_dim = 64

# Input Layer
input_tensor = Input(shape=(max_len-1,))

# Embedding Block
emb_layer = PositionalEmbedding(max_len-1, vocab_size, embed_size)
(input_tensor)

# Encoder Block
trans_enc = EncoderLayer(embed_size, head_count, feed_forward_dim)
(emb_layer)

# Output Head
pooled = layers.GlobalAveragePooling1D()(trans_enc)
dropped = layers.Dropout(0.1)(pooled)
hidden = layers.Dense(32, activation="relu")(dropped)
final_drop = layers.Dropout(0.1)(hidden)
final_out = layers.Dense(vocab_size, activation="softmax")(final_drop)

# Compilation
trans_model = models.Model(inputs=input_tensor, outputs=final_out)
trans_model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])
print(trans_model.summary())

# --- Training ---
print("Training Transformer...")
trans_model.fit(x_vals, y_vals, epochs=150, verbose=0)

# --- Results ---
print("\n--- Transformer Output ---")
# Reusing the generator helper from the LSTM section
print(run_generator("deep learning", 5, trans_model, max_len))
print(run_generator("education is", 5, trans_model, max_len))
```

```
Model: "functional_2"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
```

```
━━━━━━┫
┃ input_layer_1 (InputLayer)      ┃ (None, 9)              ┃
0 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━
━━━━━┫
┃ positional_embedding            ┃ (None, 9, 64)          ┃
13,056 ┃
┃ (PositionalEmbedding)           ┃                        ┃
┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━
━━━━━┫
┃ encoder_layer (EncoderLayer)    ┃ (None, 9, 64)          ┃
74,944 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━
━━━━━┫
┃ global_average_pooling1d        ┃ (None, 64)             ┃
0 ┃
┃ (GlobalAveragePooling1D)        ┃                        ┃
┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━
━━━━━┫
┃ dropout_3 (Dropout)             ┃ (None, 64)             ┃
0 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━
━━━━━┫
┃ dense_3 (Dense)                 ┃ (None, 32)             ┃
2,080 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━
━━━━━┫
┃ dropout_4 (Dropout)             ┃ (None, 32)             ┃
0 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━
━━━━━┫
┃ dense_4 (Dense)                 ┃ (None, 195)            ┃
6,435 ┃
┗━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┻━━━━━━━━━━━━━━━━━━━━━━━━┻━━━━━━━━━━
━━━━━┛

 Total params: 96,515 (377.01 KB)

 Trainable params: 96,515 (377.01 KB)

 Non-trainable params: 0 (0.00 B)

None
Training Transformer...

--- Transformer Output ---
deep learning uses multi layer neural networks
education is being improved using artificial intelligence
```