```
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

# Dataset from the lab document
corpus_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.
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.
"""

# Basic cleaning
corpus_list = [line.lower() for line in corpus_text.split('\n') if line]
```

```
import random
from collections import defaultdict

class NGramModel:
    def __init__(self, n):
        self.n = n
        self.ngrams = defaultdict(list)

    def train(self, corpus):
        # Create n-grams from the corpus
        for sentence in corpus:
            tokens = sentence.split()
            # Pad with start/end tokens if necessary, but skipping for simplicity
            for i in range(len(tokens) - self.n):
                seq = tuple(tokens[i:i+self.n])
                next_word = tokens[i+self.n]
                self.ngrams[seq].append(next_word)

    def generate(self, seed_text, num_words=10):
        current_seq = tuple(seed_text.split()[-self.n:])
        output = list(current_seq)

        for _ in range(num_words):
            if current_seq not in self.ngrams:
```

```
                break
            # Randomly select next word based on frequency (simple sampling)
            next_word = random.choice(self.ngrams[current_seq])
            output.append(next_word)
            current_seq = tuple(output[-self.n:])

        return ' '.join(output)

# Execution
ngram = NGramModel(n=2) # Bigram model
ngram.train(corpus_list)
print("--- N-Gram Output ---")
print(ngram.generate("artificial intelligence"))
```

```
--- N-Gram Output ---
artificial intelligence is transforming modern society.
```

```
# --- Preprocessing [cite: 22, 23, 24] ---

tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus_list)
total_words = len(tokenizer.word_index) + 1

input_sequences = []
for line in corpus_list:
    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)

# Pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# Create predictors and label
X, y = input_sequences[:,:-1], input_sequences[:,-1]
y = to_categorical(y, num_classes=total_words)

# --- Model Design [cite: 25] ---
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

model_rnn = Sequential()
model_rnn.add(Embedding(total_words, 64, input_length=max_sequence_len-1))
# Using LSTM as requested in
model_rnn.add(LSTM(100))
model_rnn.add(Dense(total_words, activation='softmax'))

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

# --- Training [cite: 26] ---
# 100 epochs is usually enough for this tiny dataset
model_rnn.fit(X, y, epochs=100, verbose=0)

# --- Generation Function [cite: 27] ---
def generate_text_rnn(seed_text, next_words, model, max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_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("\n--- LSTM Generated Output ---")
print(generate_text_rnn("artificial intelligence", 5, model_rnn, max_sequence_len))
print(generate_text_rnn("neural networks", 6, model_rnn, max_sequence_len))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is depre
  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

--- LSTM Generated Output ---
artificial intelligence is transforming modern society society
```

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

# --- Transformer Components  ---

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

    def call(self, inputs):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output)
        return self.layernorm2(out1 + ffn_output)

class TokenAndPositionEmbedding(layers.Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super(TokenAndPositionEmbedding, self).__init__()
        self.token_emb = layers.Embedding(input_dim=vocab_size, output_dim=embed_dim)
        self.pos_emb = layers.Embedding(input_dim=maxlen, output_dim=embed_dim)

    def call(self, x):
        maxlen = tf.shape(x)[-1]
        positions = tf.range(start=0, limit=maxlen, delta=1)
        positions = self.pos_emb(positions)
        x = self.token_emb(x)
        return x + positions

# --- Model Architecture  ---
embed_dim = 64  # Embedding size for each token
num_heads = 4   # Number of attention heads
ff_dim = 64     # Hidden layer size in feed forward network inside transformer

inputs = Input(shape=(max_sequence_len-1,))
embedding_layer = TokenAndPositionEmbedding(max_sequence_len-1, total_words, embed_dim)
x = embedding_layer(inputs)
transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
x = transformer_block(x)
x = layers.GlobalAveragePooling1D()(x) # Pooling to flatten for the dense layer
x = layers.Dropout(0.1)(x)
x = layers.Dense(32, activation="relu")(x)
x = layers.Dropout(0.1)(x)
outputs = layers.Dense(total_words, activation="softmax")(x)

model_trans = models.Model(inputs=inputs, outputs=outputs)
model_trans.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
print(model_trans.summary())

# --- Training [cite: 80] ---
# Transformers generally need more data, but we will train on this small set
model_trans.fit(X, y, epochs=150, verbose=0)

# --- Generation [cite: 81] ---
print("\n--- Transformer Generated Output ---")
```

```
# We reuse the generation function as the input/output structure is the same
print(generate_text_rnn("deep learning", 5, model_trans, max_sequence_len))
print(generate_text_rnn("education is", 5, model_trans, max_sequence_len))
```

Model: "functional_2"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 9)	0
token_and_position_embedding (TokenAndPositionEmbedding)	(None, 9, 64)	13,056
transformer_block (TransformerBlock)	(None, 9, 64)	74,944
global_average_pooling1d (GlobalAveragePooling1D)	(None, 64)	0
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

```
--- Transformer Generated Output ---
deep learning uses multi layer neural networks
```