```
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.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
import random
```

```
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.
"""
```

```
words = data.lower().replace("\n"," ").split()

ngram_model = {}

for i in range(len(words)-1):
    w1 = words[i]
    w2 = words[i+1]

    if w1 not in ngram_model:
        ngram_model[w1] = []
        ngram_model[w1].append(w2)

print("Total Keys in Ngram Model:", len(ngram_model))
```

```
Total Keys in Ngram Model: 139
```

```
def generate_ngram_text(seed, n_words=20):
    result = [seed]

    for _ in range(n_words):
        last = result[-1]
        if last in ngram_model:
            next_word = random.choice(ngram_model[last])
            result.append(next_word)
        else:
            break

    return " ".join(result)

print(generate_ngram_text("artificial", 25))
```

```
artificial intelligence is essential in healthcare finance education and gru models predict 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
print("Total Vocabulary:", total_words)
```

```
Total Vocabulary: 134
```

```
input_sequences = []

for line in data.split("\n"):
    token_list = tokenizer.texts_to_sequences([line])[0]

    for i in range(1, len(token_list)):
        n_gram_seq = token_list[:i+1]
        input_sequences.append(n_gram_seq)

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, y.shape)
```

```
(167, 8) (167, 134)
```

```
model = Sequential([
    Embedding(input_dim=total_words,
              output_dim=64,
              input_shape=(max_len-1,)),

    SimpleRNN(128),
    Dense(total_words, activation='softmax')
])

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

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:100: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
  super().__init__(**kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 8, 64)	8,576
simple_rnn (SimpleRNN)	(None, 128)	24,704
dense (Dense)	(None, 134)	17,286

```
 Total params: 50,566 (197.52 KB)
 Trainable params: 50,566 (197.52 KB)
 Non-trainable params: 0 (0.00 B)
```

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

```
Epoch 1/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 226ms/step - accuracy: 0.0171 - loss: 4.8993
Epoch 2/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0754 - loss: 4.8047
Epoch 3/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1129 - loss: 4.7097
Epoch 4/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0848 - loss: 4.6065
Epoch 5/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0830 - loss: 4.4990
Epoch 6/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.0890 - loss: 4.4663
Epoch 7/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1228 - loss: 4.3170
Epoch 8/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1261 - loss: 4.2397
Epoch 9/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1373 - loss: 4.1714
Epoch 10/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.2155 - loss: 4.0643
Epoch 11/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.2377 - loss: 3.9342
Epoch 12/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.2756 - loss: 3.7768
Epoch 13/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.3004 - loss: 3.6640
```

```
Epoch 14/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4010 - loss: 3.4697
Epoch 15/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4063 - loss: 3.3331
Epoch 16/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4435 - loss: 3.2372
Epoch 17/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4991 - loss: 3.0076
Epoch 18/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5413 - loss: 2.8813
Epoch 19/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5555 - loss: 2.7781
Epoch 20/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5276 - loss: 2.5613
Epoch 21/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6206 - loss: 2.3470
Epoch 22/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6372 - loss: 2.2566
Epoch 23/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6314 - loss: 2.1436
Epoch 24/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6625 - loss: 2.0075
Epoch 25/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6797 - loss: 1.9394
Epoch 26/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7252 - loss: 1.7471
Epoch 27/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7393 - loss: 1.6673
Epoch 28/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7442 - loss: 1.6067
Epoch 29/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7557 - loss: 1.5263
```

```
def generate_rnn_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_rnn_text("artificial intelligence", 25))
```

```
artificial intelligence is transforming modern society artificial intelligence predict transforming modern society artificial intelligence is transforming modern societ
```

```
print("\n==============================")
print(" N-GRAM vs RNN COMPARISON")
print("==============================\n")

seed = "artificial intelligence"

print("Seed Text:", seed)
print("\n--- NGRAM OUTPUT ---")
print(generate_ngram_text("artificial", 25))

print("\n--- RNN OUTPUT ---")
print(generate_rnn_text(seed, 25))

print("\n--- OBSERVATION ---")
print("NGRAM:")
print("- Local word prediction only")
print("- May break sentence flow")

print("\nRNN:")
print("- Understands sequence patterns")
print("- More meaningful and coherent text")
```

```
==============================
 N-GRAM vs RNN COMPARISON
==============================

Seed Text: artificial intelligence

--- NGRAM OUTPUT ---
artificial intelligence. intelligent systems. large datasets help models can create stories poems and gru models learn complex patterns. deep learning uses multi layer

--- RNN OUTPUT ---
artificial intelligence is transforming modern society artificial intelligence predict transforming modern society artificial intelligence is transforming modern societ

--- OBSERVATION ---
NGRAM:
- Local word prediction only
- May break sentence flow

RNN:
- Understands sequence patterns
- More meaningful and coherent text
```

```
import gradio as gr

def rnn_ui(seed_text, length):
    return generate_rnn_text(seed_text, int(length))

demo = gr.Interface(
    fn=rnn_ui,
    inputs=[
        gr.Textbox(label="Enter Seed Text", value="artificial intelligence"),
        gr.Slider(5,50,value=20,step=1,label="Generate Words")
    ],
    outputs=gr.Textbox(label="Generated Text"),
    title="RNN Text Generation UI",
    description="GenAI Lab-4 — Text Generation using RNN"
)

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://31ac492c685365b01e.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
```

RNN Text Generation UI

GenAI Lab-4 — Text Generation using RNN

Enter Seed Text

artificial intelligence

Generate Words 20

5 50

Generated Text

Flag

Clear Submit

Use via API · Built with Gradio · Settings

Next steps: Deploy to Cloud Run

```
Start coding or generate with AI.
```