```
!pip install tensorflow
```

```
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/di
Collecting tensorboard~=2.19.0 (from tensorflow)
  Downloading tensorboard-2.19.0-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.1
Collecting tensorflow-io-gcs-filesystem>=0.23.1 (from tensorflow)
  Downloading tensorflow_io_gcs_filesystem-0.37.1-cp311-cp311-manylinux_2_17_x86_6
Collecting wheel<1.0,>=0.23.0 (from astunparse>=1.6.0->tensorflow)
  Downloading wheel-0.45.1-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dis
Requirement already satisfied: markdown>=2.6.8 in /usr/lib/python3/dist-packages (
Collecting tensorboard-data-server<0.8.0,>=0.7.0 (from tensorboard~=2.19.0->tensor
  Downloading tensorboard_data_server-0.7.2-py3-none-manylinux_2_31_x86_64.whl.met
Collecting werkzeug>=1.0.1 (from tensorboard~=2.19.0->tensorflow)
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.1
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packag
Downloading tensorflow-2.19.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 644.9/644.9 MB 592.4 kB/s eta 0:00:00
Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Downloading flatbuffers-25.2.10-py2.py3-none-any.whl (30 kB)
Downloading google_pasta-0.2.0-py3-none-any.whl (57 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.5/57.5 kB 4.1 MB/s eta 0:00:00
Downloading libclang-18.1.1-py2.py3-none-manylinux2010_x86_64.whl (24.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.5/24.5 MB 64.4 MB/s eta 0:00:00
```

```
 # RNN/LSTM for Text Generation
nltk.download('punkt_tab')
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, Model
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, GRU
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
import re
import time
import os
import nltk
from nltk.tokenize import word_tokenize
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.lm import MLE

# Download necessary NLTK data
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# -----------------------------------------
# Data Loading and Preprocessing
# -----------------------------------------

# Download Shakespeare text data
filepath = tf.keras.utils.get_file('shakespeare.txt',
    'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt'

# Read the data
with open(filepath, 'r') as file:
    text = file.read()

# Preprocessing
print(f"Original text length: {len(text)}")
print(f"First 500 characters:\n{text[:500]}")

# Simple text cleaning
text = re.sub(r'[^\w\s]', ' ', text)
text = re.sub(r'\s+', ' ', text)
text = text.lower().strip()

print(f"\nCleaned text length: {len(text)}")
```

```
print(f"\nCleaned text length: {len(text)}")
print(f"First 500 characters after cleaning:\n{text[:500]}")

# Tokenization and sequence creation for RNN
max_sequence_len = 50
step = 3  # How many characters to skip before starting the next sequence

# Create input sequences and labels
input_sequences = []
labels = []

for i in range(0, len(text) - max_sequence_len, step):
    input_sequences.append(text[i:i+max_sequence_len])
    labels.append(text[i+max_sequence_len])

print(f"\nNumber of sequences: {len(input_sequences)}")
print(f"Example sequence: '{input_sequences[0]}' -> '{labels[0]}'")

# Create character-level tokenizer
chars = sorted(list(set(text)))
char_to_idx = {char: idx for idx, char in enumerate(chars)}
idx_to_char = {idx: char for idx, char in enumerate(chars)}

vocab_size = len(chars)
print(f"\nVocabulary size: {vocab_size}")

# Convert sequences and labels to numeric form
X = np.zeros((len(input_sequences), max_sequence_len, vocab_size), dtype=np.bool_
y = np.zeros((len(input_sequences), vocab_size), dtype=np.bool_)

for i, sequence in enumerate(input_sequences):
    for t, char in enumerate(sequence):
        X[i, t, char_to_idx[char]] = 1
    y[i, char_to_idx[labels[i]]] = 1

# Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, random_sta

print(f"Training set shape: {X_train.shape}")
print(f"Validation set shape: {X_val.shape}")

# ------------------------------------------
# Traditional N-gram Language Model
# ------------------------------------------
print("\n------------------------------------------")
print("Traditional N-gram Language Model")
print("------------------------------------------")

# Prepare data for n-gram model
tokens = word_tokenize(text[:1000000])  # Using first 1M chars to speed up traini
print(f"Number of tokens for n-gram model: {len(tokens)}")

# Create 3-gram model
n = 3
ngram_train, vocab = padded_everygram_pipeline(n, [tokens])
```

```
# Train MLE model
start_time = time.time()
mle_model = MLE(n)
mle_model.fit(ngram_train, vocab)
ngram_time = time.time() - start_time
print(f"N-gram model trained in {ngram_time:.2f} seconds")

# Function to generate text with n-gram model
def generate_text_ngram(model, seed_text, num_words=100):
    current_text = word_tokenize(seed_text.lower())
    generated_words = []

    for _ in range(num_words):
        # Get context for prediction
        context = current_text[-(model.order-1):] if len(current_text) >= model.c

        # Generate next word
        try:
            next_word = model.generate(1, context)[0]
        except:
            next_word = np.random.choice(list(model.vocab))

        generated_words.append(next_word)
        current_text.append(next_word)

    return ' '.join(generated_words)

# Test n-gram model
seed_text = "to be or not"
ngram_generated = generate_text_ngram(mle_model, seed_text)
print(f"\nN-gram generated text from seed '{seed_text}':")
print(ngram_generated)

# -----------------------------------------
# RNN Models for Text Generation
# -----------------------------------------
print("\n-----------------------------------------")
print("RNN Models for Text Generation")
print("-----------------------------------------")

# 1. Simple RNN Model
print("\nBuilding Simple RNN Model...")
simple_rnn_model = Sequential([
    LSTM(128, input_shape=(max_sequence_len, vocab_size), return_sequences=True),
    Dropout(0.3),
    LSTM(128),
    Dense(vocab_size, activation='softmax')
])

simple_rnn_model.compile(
    optimizer=Adam(learning_rate=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
simple_rnn_model.summary()

# 2. Deep LSTM Model
print("\nBuilding Deep LSTM Model...")
deep_lstm_model = Sequential([
    LSTM(256, input_shape=(max_sequence_len, vocab_size), return_sequences=True),
    Dropout(0.3),
    LSTM(256, return_sequences=True),
    Dropout(0.3),
    LSTM(256),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(vocab_size, activation='softmax')
])

deep_lstm_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

deep_lstm_model.summary()

# 3. GRU Model
print("\nBuilding GRU Model...")
gru_model = Sequential([
    GRU(256, input_shape=(max_sequence_len, vocab_size), return_sequences=True),
    Dropout(0.3),
    GRU(256),
    Dense(vocab_size, activation='softmax')
])

gru_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

gru_model.summary()

# Define callbacks
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss'),
    ModelCheckpoint('best_model.h5', save_best_only=True, monitor='val_loss')
]

# Training function
def train_model(model, model_name, epochs=10):
    start_time = time.time()
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=epochs,
        batch_size=128,
        callbacks=callbacks,
```

```
        verbose=1
    )
    training_time = time.time() - start_time

    # Evaluate model
    scores = model.evaluate(X_val, y_val, verbose=0)

    print(f"\n{model_name} Results:")
    print(f"Training time: {training_time:.2f} seconds")
    print(f"Validation accuracy: {scores[1]:.4f}")

    # Plot training history
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(f'{model_name} - Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(f'{model_name} - Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.savefig(f'{model_name.replace(" ", "_").lower()}_training_history.png')
    plt.show()

    return {
        'model': model,
        'name': model_name,
        'training_time': training_time,
        'accuracy': scores[1],
        'history': history.history
    }

# Adjust epochs for demonstration
training_epochs = 3  # Reduced for demonstration purposes

# Train models
print("\nTraining Simple RNN Model...")
simple_rnn_results = train_model(simple_rnn_model, "Simple RNN", epochs=training_

print("\nTraining Deep LSTM Model...")
deep_lstm_results = train_model(deep_lstm_model, "Deep LSTM", epochs=training_epc

print("\nTraining GRU Model...")
gru_results = train_model(gru_model, "GRU", epochs=training_epochs)
```

```
# Function to generate text with RNN models
def generate_text_rnn(model, seed_text, max_length=100):
    generated_text = seed_text

    for _ in range(max_length):
        # Convert the generated text to a sequence
        x_pred = np.zeros((1, max_sequence_len, vocab_size))
        for t, char in enumerate(generated_text[-max_sequence_len:]):
            if char in char_to_idx:
                x_pred[0, t, char_to_idx[char]] = 1

        # Predict the next character
        predictions = model.predict(x_pred, verbose=0)[0]
        next_index = np.argmax(predictions)
        next_char = idx_to_char[next_index]

        # Add the next character to the generated text
        generated_text += next_char

    return generated_text

# Generate text with each model
print("\nGenerating text with each model...")
seed_text = "to be or not to be"
print(f"Seed text: '{seed_text}'")

for model_result in [simple_rnn_results, deep_lstm_results, gru_results]:
    model = model_result['model']
    name = model_result['name']

    generated_text = generate_text_rnn(model, seed_text)
    print(f"\n{name} generated text:")
    print(generated_text)

    # Save generated text to file
    with open(f"{name.replace(' ', '_').lower()}_generated_text.txt", 'w') as f:
        f.write(generated_text)

# ----------------------------------------
# Model Comparison
# ----------------------------------------
print("\n----------------------------------------")
print("Model Comparison")
print("----------------------------------------")

# Collect results for comparison
models = [
    {'name': 'N-gram Model', 'training_time': ngram_time, 'accuracy': None},
    simple_rnn_results,
    deep_lstm_results,
    gru_results
]

# Create comparison table
comparison_data = {
```

```
     'Model': [m['name'] for m in models],
    'Training Time (s)': [m['training_time'] for m in models],
    'Validation Accuracy': [m['accuracy'] if 'accuracy' in m else 'N/A' for m in 
}

comparison_df = pd.DataFrame(comparison_data)
print("\nModel Comparison:")
print(comparison_df)

# Plot comparison
plt.figure(figsize=(10, 6))

# Training time comparison
plt.subplot(1, 2, 1)
plt.bar(comparison_df['Model'][1:], comparison_df['Training Time (s)'][1:])
plt.title('Training Time Comparison')
plt.ylabel('Time (seconds)')
plt.xticks(rotation=45)

# Accuracy comparison
plt.subplot(1, 2, 2)
valid_accuracies = comparison_df['Validation Accuracy'][1:]
plt.bar(comparison_df['Model'][1:], valid_accuracies)
plt.title('Validation Accuracy Comparison')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)

plt.tight_layout()
plt.savefig('model_comparison.png')
plt.show()

# ------------------------------------------
# Demonstrate Backpropagation Through Time
# ------------------------------------------
print("\n------------------------------------------")
print("Backpropagation Through Time (BPTT) Visualization")
print("------------------------------------------")

# Create a simple LSTM model for BPTT visualization
bptt_model = Sequential([
    LSTM(64, input_shape=(max_sequence_len, vocab_size), return_sequences=True, r
    LSTM(64, name='lstm_2', return_sequences=False),
    Dense(vocab_size, activation='softmax', name='output')
])

bptt_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Define a custom callback to track gradients
class GradientTracker(tf.keras.callbacks.Callback):
    def __init__(self, validation_data, interval=1):
        super(GradientTracker, self).__init__()
        self.validation_data = validation_data
```

```
        self.validation_data = validation_data
        self.interval = interval
        self.gradients = []

    def on_epoch_end(self, epoch, logs=None):
        if epoch % self.interval == 0:
            inputs, targets = self.validation_data

            with tf.GradientTape() as tape:
                # Forward pass
                pred = self.model(inputs[:1], training=True)
                loss = self.model.loss(targets[:1], pred)

            # Get gradients for each layer
            grads = tape.gradient(loss, self.model.trainable_weights)

            # Compute gradient norms for LSTM layers
            lstm_grads = [g for g, w in zip(grads, self.model.trainable_weights)
                          if 'lstm' in w.name and 'kernel' in w.name]

            # Compute gradient norms
            lstm_grad_norms = [tf.norm(g).numpy() for g in lstm_grads]

            # Store the gradient norms
            self.gradients.append({
                'epoch': epoch,
                'lstm_1': lstm_grad_norms[0] if len(lstm_grad_norms) > 0 else 0,
                'lstm_2': lstm_grad_norms[1] if len(lstm_grad_norms) > 1 else 0,
            })

# Train the BPTT model for demonstration
print("\nTraining model to demonstrate BPTT...")
grad_tracker = GradientTracker((X_val[:10], y_val[:10]))

bptt_history = bptt_model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=3,  # Just a few epochs for demonstration
    batch_size=128,
    callbacks=[grad_tracker],
    verbose=1
)

# Plot gradient norms to visualize BPTT
plt.figure(figsize=(10, 6))
epochs = [g['epoch'] for g in grad_tracker.gradients]
lstm1_grads = [g['lstm_1'] for g in grad_tracker.gradients]
lstm2_grads = [g['lstm_2'] for g in grad_tracker.gradients]

plt.plot(epochs, lstm1_grads, 'o-', label='LSTM Layer 1')
plt.plot(epochs, lstm2_grads, 'o-', label='LSTM Layer 2')
plt.title('Gradient Norms during Backpropagation Through Time')
plt.xlabel('Epoch')
plt.ylabel('Gradient Norm')
plt.legend()
plt.savefig('bptt_visualization.png')
```

```
plt.show()

# Summary
print("\nSummary of Text Generation Models:")
print("1. Traditional N-gram models are fast to train but have limited capacity t
print("2. RNN and LSTM models capture long-term dependencies, resulting in more c
print("3. Deeper LSTM architectures generally perform better but take longer to t
print("4. GRU models provide a good balance between performance and training effi
print("\nBackpropagation Through Time (BPTT) is crucial for training recurrent mo

print("\nExperiment completed. Results, models, and generated text have been save
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Original text length: 1115394
First 500 characters:
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?

All:
No more talking on't; let it be done: away, away!

Second Citizen:
One word, good citizens.

First Citizen:
We are accounted poor

Cleaned text length: 1059634
First 500 characters after cleaning:
first citizen before we proceed any further hear me speak all speak speak first ci

Number of sequences: 353195
Example sequence: 'first citizen before we proceed any further hear m' -> 'e'

Vocabulary size: 28
Training set shape: (317875, 50, 28)
Validation set shape: (35320, 50, 28)

----------------------------------------
Traditional N-gram Language Model
----------------------------------------
Number of tokens for n-gram model: 196996
N-gram model trained in 2.60 seconds

N-gram generated text from seed 'to be or not':
r l s o c w y s p b s r l s t a l s o c w a w s s m s l s f h c w n u m b t h s t

----------------------------------------
RNN Models for Text Generation
----------------------------------------

Building Simple RNN Model...
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarni
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50, 128)	80,384
dropout (Dropout)	(None, 50, 128)	0
lstm_1 (LSTM)	(None, 128)	131,584
dense (Dense)	(None, 28)	3,612

Total params: 215,580 (842.11 KB)
Trainable params: 215,580 (842.11 KB)
Non-trainable params: 0 (0.00 B)

Building Deep LSTM Model...
Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 50, 256)	291,840
dropout_1 (Dropout)	(None, 50, 256)	0
lstm_3 (LSTM)	(None, 50, 256)	525,312
dropout_2 (Dropout)	(None, 50, 256)	0
lstm_4 (LSTM)	(None, 256)	525,312
dense_1 (Dense)	(None, 128)	32,896
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 28)	3,612