Deep Learning Assignment

on

RNN for Text Generation

Submitted by

VIVEK SHARMA (23MSD7013)

Under the Supervision of

Dr. A. Manimaran

In partial fulfillment of the award of the degree of

Master of Science

In

Data Science

Department of Mathematics
School of Advanced Sciences
VIT-AP University

CONTENTS

INTRODUCTION

In recent years, Recurrent Neural Networks (RNNs) have become a cornerstone in text generation tasks, particularly for modeling and generating sequences that require capturing dependencies across words and phrases. This assignment explores the application of RNNs for generating text that emulates the stylistic nuances of William Shakespeare's writings. Using a curated dataset of Shakespeare's plays, an RNN was trained to generate text that aligns with the structure, vocabulary, and expression typical of Shakespeare's language.

Objective: The primary objective of this assignment was to design and train an RNN model that can generate text resembling Shakespeare's literary style. This involved experimenting with different RNN architectures, such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), to evaluate their effectiveness in capturing complex linguistic patterns. Additionally, the assignment sought to analyze the generated output, assessing its coherence, fluency, and resemblance to Shakespearean prose.

DATASET OVERVIEW

The dataset for this assignment consists of the complete works of William Shakespeare, downloaded from an open-source collection. This collection includes plays, sonnets, and other writings, providing a rich corpus with diverse vocabulary and syntactic complexity. This choice of dataset enables the model to capture the unique linguistic style characteristic of Shakespeare's era, including archaic words, rhythmic flow, and structured dialogue.

Key Dataset Attributes:

- **Source**: The dataset was sourced from tensorflow.org/data/shakespeare.txt
- **Size**: The dataset contains approximately 1.1 million characters, which provides a substantial amount of data for training a language model.
- **Sample Text**: A quick examination of the text reveals the distinct language style used by Shakespeare. For instance, the first 250 characters include:

```
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.
```

- **Character Distribution**: The dataset includes a mix of alphabets, punctuation marks, and whitespace, all of which play a role in shaping the model's output.

DATA PREPROCESSING

To prepare the text for training, several preprocessing steps were applied:

- **Tokenization**: Each character in the text was assigned a unique integer, creating a vocabulary of 13,009 tokens. This approach enables character-based text generation, allowing the model to mimic Shakespeare's style at a granular level.
- **Sequence Generation**: The text was split into overlapping sequences, each 100 characters long, with the subsequent character designated as the target for prediction. This length captures enough context without overloading the model's memory.
- **Batching**: The data was divided into batches of size 64, shuffled to ensure the model learns generalized patterns across different parts of the text.

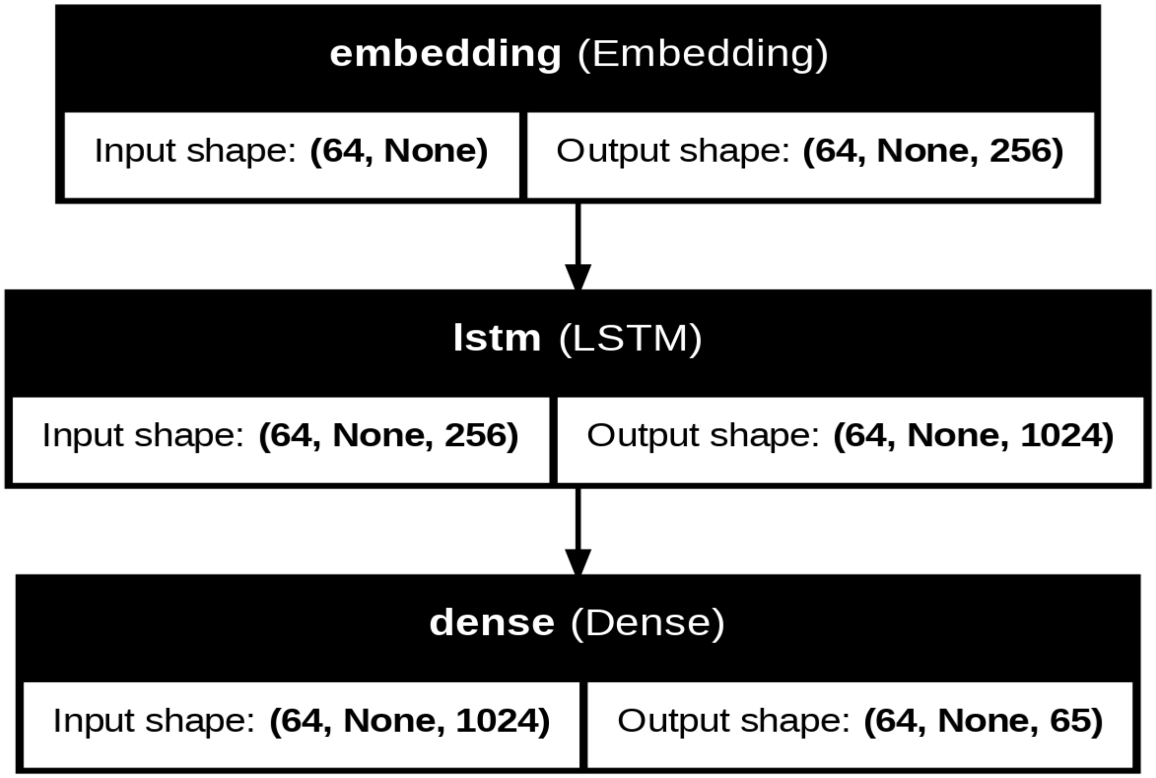
MODEL ARCHITECTURE

Two separate models were built to compare the effectiveness of LSTM and GRU layers in generating Shakespearean text. Both models share a similar architecture, differing only in the type of recurrent layer used (LSTM or GRU):

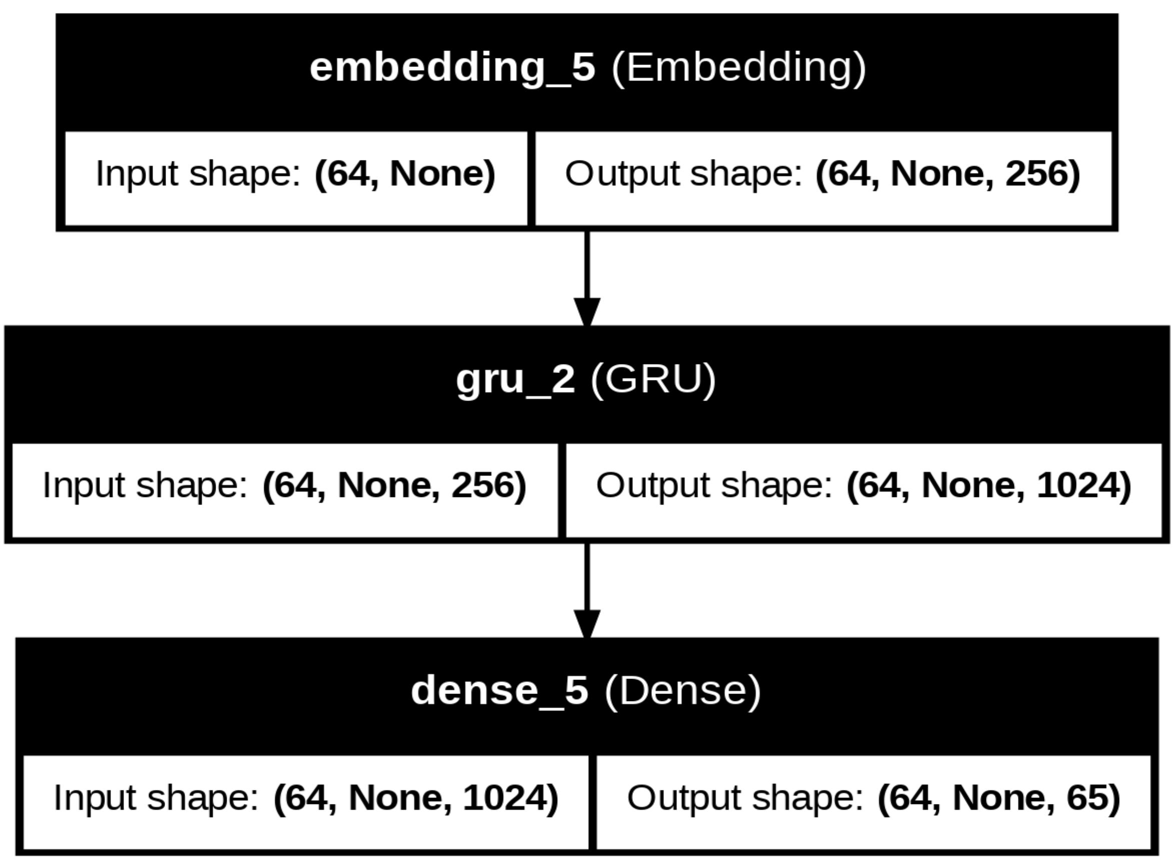
- Embedding Layer: Converts each character to a dense vector, with an output embedding dimension 0f 256, enabling the model to capture relationships between characters in a continuous space.
- Recurrent Layer: The first model uses an LSTM layer with 1024 units, while the second model replaces this layer with a GRU layer. Both layers are designed to capture dependencies over long sequences, but GRU tends to be more computationally efficient.
- Dense Output Layer: A fully connected layer with softmax activation generates probabilities for each character, allowing the model to predict the next character based on learned patterns.

Both models were trained with sparse categorical cross-entropy loss and optimized using the Adam optimizer. This dual approach helps in assessing the comparative performance of LSTM and GRU for text generation.

LSTM Architecture:



GRU Architecture:



Model Summary:

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(64, None, 256)	16,640
lstm (LSTM)	(64, None, 1024)	5,246,976
dense (Dense)	(64, None, 65)	66,625

Total params: 5,330,241 (20.33 MB)

Trainable params: 5,330,241 (20.33 MB)

Non-trainable params: 0 (0.00 B)

Model: "sequential_5"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(64, None, 256)	16,640
gru_2 (GRU)	(64, None, 1024)	3,938,304
dense_5 (Dense)	(64, None, 65)	66,625

Total params: 4,021,569 (15.34 MB)

Trainable params: 4,021,569 (15.34 MB)

Non-trainable params: 0 (0.00 B)

MODEL TRAINING

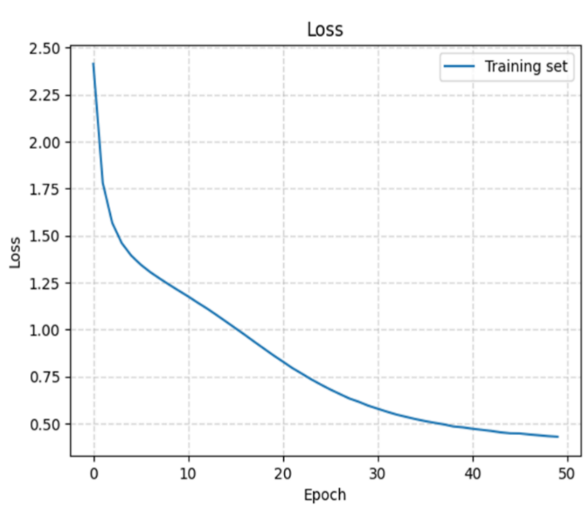
Both the LSTM and GRU models were trained using a sparse categorical cross-entropy loss function, which is suitable for multi-class classification tasks where each character represents a class. The Adam optimizer was applied with a learning rate of 0.001, chosen for its efficiency in handling non-stationary objectives.

Training Configuration:

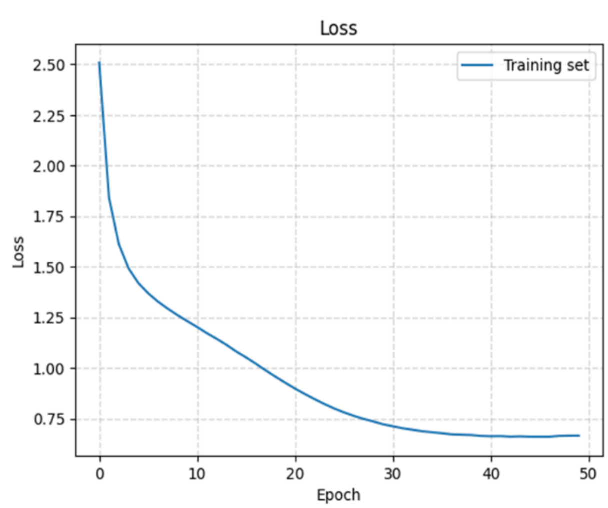
- **Epochs**: Both models were trained for 50 epochs to allow sufficient learning of character sequences.
- **Checkpointing**: Model checkpoints were configured to save the best weights based on loss, enabling easy restoration of the best-performing model for text generation.

After training, the loss was plotted against epochs to visualize convergence, which provided insights into each model's learning progress. The resulting loss curve illustrated the model’s ability to reduce prediction error over time.

LSTM



GRU



RESULTS

After training, text generation was performed using the restored RNN models, each accepting an initial *start string* and generating a specified number of characters based on learned patterns. Using the model's predictions, the text was generated iteratively by feeding each predicted character back into the model to continue the sequence.

The text generation function enables flexible generation with a *temperature* parameter that adjusts the randomness of predictions. Lower temperatures yield more predictable and coherent text, while higher temperatures produce more diverse, creative output.

Sample Outputs:
Default Temperature (1.0): Starting with "ROMEO: ", the generated text showed Shakespearean-style structure, capturing elements like archaic language and dialogue form, though coherence varied.

```
ROMEO: for that I read themselves
Like rags that should endure us broke them more
To swift be itself to be brief wash'd all;
And vouch it to the heavy caused thou strikest me sore wither'd have I in her through they us.

Second Keeper:
But, as it is, Caius Marcius: there my hearts!

TYBALTH:
Which so hang'd up thy friend
Is my poor trade, flesh with the English peers,
That raise his body to the cushion of his mother;
Cry 'Centeracted the king's house, Marcius, whose circums
In he remembering whom we think they take upon
me; I will one nor enemy; you home to crow;
And all comforts are hollow'd friendships.

SOMERSET:
A sixt of all, he's more to purge her forth,
But 'twas the wise for which he play'd it stankedowe a thousand-fold more less;
Therefore die Richard that struck upon thyself?

JULIET:
Farewell! good Pompey. is good night, I would have head
A man well known that we mean to lo;
And he shall scarce call thus, for it good
And be in char he hath shortly of the fire
Of every we to Barthla
```

Higher Temperature (0.8): Using "BRUTUS: " as the start string, this output was more unpredictable, creating unique phrasing while retaining Shakespearean syntax and vocabulary.

```
BRUTUS: O prince, is an earthly modest, some pardon
Are of themselves, that to the palace gall'd in the hour,
For she is spoken of my country's light,
Seldoms, and Romeo did before you go;
And now I fear some ill: Signior Placent in the gove; next, that would have held unto the king.

Second Citizen:
Marry, we will bestrew them, and I hate;
But this alliance may shoot?
O, thou look'st on my journey, and must die with me,
But my true love me well, good follow.

First Senator:
D'd you y, but surely.

Second Servant:
O, these are the music of Time.

SICINIUS:
For the marance and the greater fierce hands no foot,
As if the rest were your ancess.
You are treacher! and he shall turn of you;
And with shall prove false friends; him not am I king!
Edward the man, slow, go with me;
Who now came I him in the best, a beggar.

MENENIUS:
Not to him, and leapthee.

CORIOLANUS:
Cut me not, something that is not the king, and rene
Be satisfied, and beguit home:
Now come too lightnfolk from Pardon for it,
And s
```

Overall, both models successfully emulated Shakespeare’s language style, demonstrating the RNN’s ability to learn character-level dependencies. These results highlight the impact of temperature on text creativity, with lower values yielding structured outputs and higher values providing more diverse but less predictable text.

CONCLUSION

In this assignment, we successfully trained RNN models with LSTM and GRU layers to generate text resembling Shakespeare's style. Both models were able to capture character-level dependencies and produce coherent text sequences that mimic Shakespearean language. Experimenting with temperature values in the text generation function allowed for control over the creativity of the output, with lower temperatures producing more structured language and higher temperatures offering unpredictable and varied phrasing.

Overall, this project demonstrates the effectiveness of recurrent architectures for sequence generation tasks, highlighting the balance between model architecture (LSTM vs. GRU) and hyperparameter tuning (e.g., temperature) in generating stylistically accurate text. Future improvements could explore deeper architectures or alternative training methods to enhance the model’s ability to capture long-range dependencies, potentially resulting in even more coherent and contextually accurate outputs.

Shakespearean Text Generation using RNNs

Import dependencies

In [1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import platform
import time
import pathlib
import os
```

Download the dataset

In [2]:
```
cache_dir = './tmp'
dataset_file_name = 'shakespeare.txt'
dataset_file_origin = 'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt'

dataset_file_path = tf.keras.utils.get_file(
    fname=dataset_file_name,
    origin=dataset_file_origin,
    cache_dir=pathlib.Path(cache_dir).absolute()
)

print(dataset_file_path)
```

```
/tmp/.keras/datasets/shakespeare.txt
```

Analyze the dataset

In [3]:
```
# Reading the text file.
text = open(dataset_file_path, mode='r').read()

print(f'Length of text: {len(text)} characters')
```

```
Length of text: 1115394 characters
```

In [4]:
```
# Take a look at the first 250 characters in text.
print(text[:250])
```

```
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.
```

In [5]:
```
# The unique characters in the file
vocab = sorted(set(text))

print(f'{len(vocab)} unique characters')
print('vocab:', vocab)
```

```
65 unique characters
vocab: ['\n', ' ', '!', '$', '&', "'", ',', '-', '.', '3', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'
, 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd',
'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

Process the text data

Vectorize the text

Before feeding the text to our RNN we need to convert the text from a sequence of characters to a sequence of numbers. To do so we will detect all unique characters in the text, form a vocabulary out of it and replace each character with its index in the vocabulary.

In [6]:
```
# Map characters to their indices in vocabulary.
char2index = {char: index for index, char in enumerate(vocab)}
```

In [7]:
```
print(char2index)
```

```
{'\n': 0, ' ': 1, '!': 2, '$': 3, '&': 4, "'": 5, ',': 6, '-': 7, '.': 8, '3': 9, ':': 10, ';': 11, '?': 12, 'A'
: 13, 'B': 14, 'C': 15, 'D': 16, 'E': 17, 'F': 18, 'G': 19, 'H': 20, 'I': 21, 'J': 22, 'K': 23, 'L': 24, 'M': 25
, 'N': 26, 'O': 27, 'P': 28, 'Q': 29, 'R': 30, 'S': 31, 'T': 32, 'U': 33, 'V': 34, 'W': 35, 'X': 36, 'Y': 37, 'Z
': 38, 'a': 39, 'b': 40, 'c': 41, 'd': 42, 'e': 43, 'f': 44, 'g': 45, 'h': 46, 'i': 47, 'j': 48, 'k': 49, 'l': 5
0, 'm': 51, 'n': 52, 'o': 53, 'p': 54, 'q': 55, 'r': 56, 's': 57, 't': 58, 'u': 59, 'v': 60, 'w': 61, 'x': 62, '
y': 63, 'z': 64}
```

In [8]:
```
# Map character indices to characters from vacabulary.
index2char = np.array(vocab)
print(index2char)
```

```
['\n' ' ' '!' '$' '&' "'" ',' '-' '.' '3' ':' ';' '?' 'A' 'B' 'C' 'D' 'E'
 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W'
 'X' 'Y' 'Z' 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o'
 'p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z']
```

In [9]:
```
# Convert chars in text to indices.
text_as_int = np.array([char2index[char] for char in text])
```

In [10]:
```
text_as_int
```

Out[10]:
```
array([18, 47, 56, ..., 45,  8,  0])
```

In [11]:
```
print(f'text_as_int length: {len(text_as_int)}')
```

```
text_as_int length: 1115394
```

In [12]:
```
# Print the first 15 characters of the original text and their integer representation
print(f'{text[:15]} --> {text_as_int[:15]}')
```

```
First Citizen:
 --> [18 47 56 57 58  1 15 47 58 47 64 43 52 10  0]
```

Create training sequences

In [13]:
```
# The maximum length sentence we want for a single input in characters.
sequence_length = 100
examples_per_epoch = len(text) // (sequence_length + 1)

print('examples_per_epoch:', examples_per_epoch)
```

```
examples_per_epoch: 11043
```

In [14]:
```
# Create training dataset.
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

for char in char_dataset.take(5):
    print(index2char[char.numpy()])
```

```
F
i
r
s
t
```

In [15]:
```
# Generate batched sequences from the character dataset
sequences = char_dataset.batch(sequence_length + 1, drop_remainder=True)

# Get the number of sequences, which is the same as examples_per_epoch
sequence_count = len(list(sequences.as_numpy_iterator()))
print(f'Sequences count: {sequence_count}\n')

# Display examples of sequences
for item in sequences.take(5):
    print(''.join(index2char[item.numpy()]))
```

```
Sequences count: 11043

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 k
now Caius Marcius is chief enemy to the people.

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

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

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

Second Citizen:
One word, good citizens.

First Citizen:
We are accounted poor citi
```

In [16]:
```
# sequences shape:
# - 11043 sequences
# - Each sequence of length 101
#
#
#    101      101           101
# [(.....) (.....) ...  (.....)]
#
# <---------- 11043 ----------->
```

For each sequence, duplicate and shift it to form the input and target text. For example, say `sequence_length` is `4` and our text is `Hello`. The input sequence would be `Hell`, and the target sequence `ello`.

In [17]:
```
def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text, target_text
```

In [18]:
```
# Map sequences to input and target text
dataset = sequences.map(split_input_target)

# The dataset size is the same as examples_per_epoch,
# but each element now has a length of `sequence_length`
# and not 'sequence_length + 1'
dataset_size = len(list(dataset.as_numpy_iterator()))
print(f'Dataset size: {dataset_size}')
```

```
Dataset size: 11043
```

In [19]:
```
# Retrieve one example from the dataset
for input_example, target_example in dataset.take(1):
    input_size = len(input_example.numpy())
    target_size = len(target_example.numpy())

    print(f'Input sequence size: {input_size}')
    print(f'Target sequence size: {target_size}\n')

    input_text = repr(''.join(index2char[input_example.numpy()]))
    target_text = repr(''.join(index2char[target_example.numpy()]))

    print(f'Input: {input_text}')
    print(f'Target: {target_text}')
```

```
Input sequence size: 100
Target sequence size: 100

Input: 'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\n
You'
Target: 'irst Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\n
You '
```

In [20]:
```
# dataset shape:
# - 11043 sequences
# - Each sequence is a tuple of 2 sub-sequences of length 100 (input_text and target_text)
#
#
#    100       100           100
# /(.....)\ /(.....)\ ... /(.....)\  <-- input_text
# \(.....)/ \(.....)/     \(.....)/  <-- target_text
#
# <----------- 11043 ------------->
```

Each index of these vectors are processed as one time step. For the input at time step 0, the model receives the index for "F" and trys to predict the index for "i" as the next character. At the next timestep, it does the same thing but the RNN considers the previous step context in addition to the current input character.

In [21]:
```
# Iterate through the first five elements of input and target examples
for i, (input_idx, target_idx) in enumerate(zip(input_example[:5], target_example[:5])):
    input_char = repr(index2char[input_idx])
    target_char =repr( index2char[target_idx])

    print(f'Step {i:2d}')
    print(f'  input: {input_idx} ({input_char})')
    print(f'  expected output: {target_idx} ({target_char})')
```

```
Step  0
  input: 18 ('F')
  expected output: 47 ('i')
Step  1
  input: 47 ('i')
  expected output: 56 ('r')
Step  2
  input: 56 ('r')
  expected output: 57 ('s')
Step  3
  input: 57 ('s')
  expected output: 58 ('t')
Step  4
  input: 58 ('t')
  expected output: 1 (' ')
```

Split training sequences into batches

We used `tf.data` to split the text into manageable sequences. But before feeding this data into the model, we need to shuffle the data and pack it into batches.

In [22]:
```
# Batch size.
BATCH_SIZE = 64

# Set the buffer size for shuffling the dataset.
# TensorFlow's data pipeline is designed for potentially infinite sequences,
# so it uses a buffer to shuffle elements rather than shuffling the entire dataset in memory.
BUFFER_SIZE = 10000

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)

dataset
```

Out[22]:
```
<_BatchDataset element_spec=(TensorSpec(shape=(64, 100), dtype=tf.int64, name=None), TensorSpec(shape=(64, 100)
, dtype=tf.int64, name=None))>
```

In [23]:
```
# Print the size of the batched dataset
batched_dataset_size = len(list(dataset.as_numpy_iterator()))
print(f'Batched dataset size: {batched_dataset_size}')
```

```
Batched dataset size: 172
```

In [24]:
```
for input_text, target_text in dataset.take(1):
    print('1st batch: input_text:', input_text)
    print()
    print('1st batch: target_text:', target_text)
```

```
1st batch: input_text: tf.Tensor(
[[20 13 25 ...  0 13 14]
 [58 43 50 ... 59 50  6]
 [44 47 58 ... 40 43 46]
 ...
 [18 18 10 ... 42 53 59]
 [52  1 13 ... 59 58 63]
 [ 1 52 53 ... 59 52 39]], shape=(64, 100), dtype=int64)

1st batch: target_text: tf.Tensor(
[[13 25 10 ... 13 14 30]
 [43 50 50 ... 50  6  0]
 [47 58 39 ... 43 46 43]
 ...
 [18 10  0 ... 53 59 40]
 [ 1 13 54 ... 58 63  1]
 [52 53 58 ... 52 39 42]], shape=(64, 100), dtype=int64)
```

In [25]:
```python
# dataset shape:
# - 172 batches
# - 64 sequences per batch
# - Each sequence is a tuple of 2 sub-sequences of length 100 (input_text and target_text)
#
#
#     100       100           100             100       100           100
# |/(.....)\ /(.....)\ ... /(.....)\| ... |/(.....)\ /(.....)\ ... /(.....)\|  <-- input_text
# |\(.....)/ \(.....)/     \(.....)/| ... |\(.....)/ \(.....)/     \(.....)/|  <-- target_text
#
# <------------- 64 ---------------->     <------------- 64 ---------------->
#
# <--------------------------------- 172 ----------------------------------->
```

Model Building

Hyperparameters

In [26]:
```python
# Length of the vocabulary in characters.
vocab_size = len(vocab)

# Dimension of the embedding layer
embedding_dim = 256

# Number of RNN units
rnn_units = 1024

# Number of samples per gradient update
# batch_size = 64
```

In [27]:
```python
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    # Define a Sequential model
    model = tf.keras.Sequential()

    # Set the input layer with fixed batch size
    model.add(tf.keras.layers.Input(batch_shape=(batch_size, None)))

    # Embedding layer
    model.add(tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=embedding_dim))

    # LSTM layer with stateful=True and a GlorotNormal initializer
    model.add(tf.keras.layers.LSTM(
        units=rnn_units,
        return_sequences=True,
        stateful=True,
        recurrent_initializer=tf.keras.initializers.GlorotNormal()
    ))

    # Dense layer with output units equal to vocab size
    model.add(tf.keras.layers.Dense(vocab_size))

    return model
```

In [28]:
```python
model = build_model(vocab_size, embedding_dim, rnn_units, BATCH_SIZE)
```

In [29]:
```python
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(64, None, 256)	16,640
lstm (LSTM)	(64, None, 1024)	5,246,976
dense (Dense)	(64, None, 65)	66,625

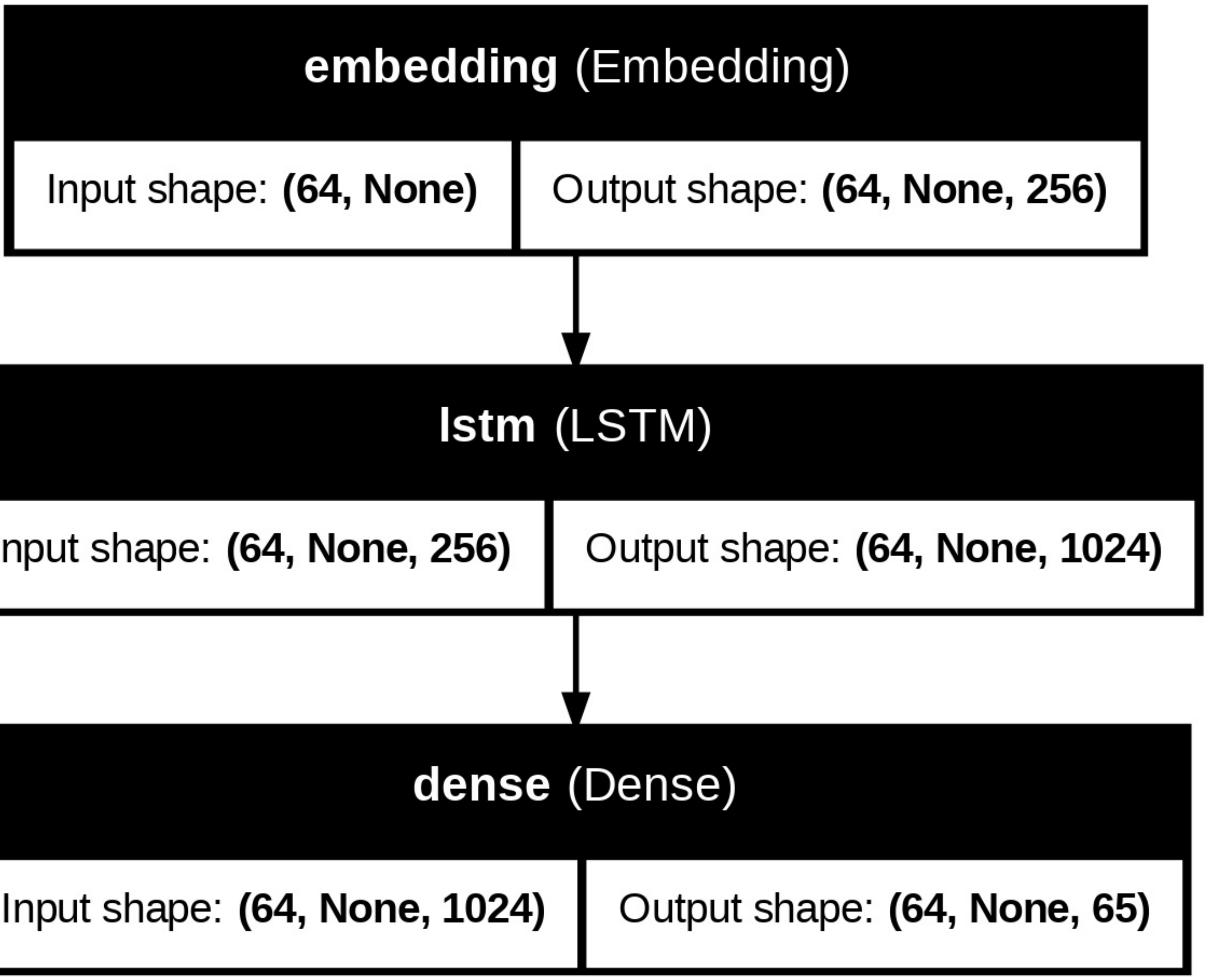
Total params: 5,330,241 (20.33 MB)

Trainable params: 5,330,241 (20.33 MB)

Non-trainable params: 0 (0.00 B)

In [30]:
```
tf.keras.utils.plot_model(
    model,
    show_shapes=True,
    show_layer_names=True,
)
```

Out[30]:



Try the model

In [31]:
```
for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print(example_batch_predictions.shape, "# (batch_size, sequence_length, vocab_size)")
```

```
(64, 100, 65) # (batch_size, sequence_length, vocab_size)
```

To get actual predictions from the model we need to sample from the output distribution, to get actual character indices. This distribution is defined by the logits over the character vocabulary.

In [32]:
```
print('Prediction for the 1st letter of the batch 1st sequense:')
print(example_batch_predictions[0, 0])
```

```
Prediction for the 1st letter of the batch 1st sequense:
tf.Tensor(
[-3.3136769e-03 -1.8216955e-03 -4.4506965e-03 -9.6264260e-04
  1.3516686e-03 -4.3478291e-03 -3.3204367e-03 -4.8723798e-03
 -8.0455130e-04  3.3995772e-03  3.0999419e-03 -8.9938578e-04
 -9.3025733e-03  2.6494870e-03  6.0682697e-03  1.8063136e-03
  2.8438068e-03 -4.9730563e-03  7.7216048e-04  3.5291386e-03
 -2.8528078e-04  5.8763768e-03 -1.5338142e-03  5.6031938e-03
  1.7908477e-03 -5.6365877e-03 -5.4445714e-03 -4.1985004e-03
  2.2727300e-03  1.0246054e-03 -4.3345070e-03 -3.0270455e-04
  2.6373642e-03 -1.2825216e-03  2.0898411e-03 -3.6836811e-04
 -8.3221169e-03  1.6993647e-03  4.4226772e-03  4.4572103e-06
  2.7378099e-03  3.7711624e-03 -3.8224093e-03  2.8337573e-03
 -3.7130071e-03  6.7760440e-04 -3.1746884e-03 -1.1740917e-03
  6.2958623e-04 -2.1350794e-03  4.1272305e-03 -2.7814067e-03
  3.4220580e-03 -9.5357513e-03 -4.1006296e-04  6.8000290e-03
  3.9745895e-03  7.5441715e-04  2.1849668e-03 -5.4619354e-03
  5.6177314e-04 -5.7808380e-03 -3.0489831e-04  2.7330685e-04
  2.7779185e-03], shape=(65,), dtype=float32)
```

In [33]:
```python
sampled_indices = tf.random.categorical(
    logits=example_batch_predictions[0],
    num_samples=1
)

sampled_indices.shape
```

Out[33]:
```
TensorShape([100, 1])
```

In [34]:
```python
sampled_indices = tf.squeeze(
    input=sampled_indices,
    axis=-1
).numpy()

sampled_indices.shape
```

Out[34]:
```
(100,)
```

In [35]:
```python
sampled_indices
```

Out[35]:
```
array([46, 19,  2, 42, 46, 36,  6, 10, 55, 33, 13, 11, 58, 48, 31, 39, 10,
       23, 29, 53, 41, 58,  1, 63,  5, 50, 39, 61, 40, 20, 16, 56, 11, 45,
       26, 17, 13, 10,  8,  0, 15,  8, 59, 12, 58, 46, 18, 33, 46, 29, 48,
       47, 28, 14, 56, 37, 49, 57, 32, 14, 30, 11, 24, 41, 19, 16, 12, 13,
       55, 15, 37, 50, 63, 44, 55, 60,  9, 40, 25, 28,  4, 25, 32,  3, 58,
        8, 18, 53, 38, 54, 36,  4,  8, 16, 39, 39, 41, 57, 10,  3])
```

In [36]:
```python
print('Input:\n', repr(''.join(index2char[input_example_batch[0]])))
print()
print('Next char prediction:\n', repr(''.join(index2char[sampled_indices])))
```

```
Input:
 "lphos, and from thence have brought\nThe seal'd-up oracle, by the hand deliver'd\nOf great Apollo's pr"

Next char prediction:
 "hG!dhX,:qUA;tjSa:KQoct y'lawbHDr;gNEA:.\nC.u?thFUhQjiPBrYksTBR;LcGD?AqCYlyfqv3bMP&MT$t.FoZpX&.Daacs:$"
```

In [37]:
```python
# Display predictions for the first 5 samples
for i, (input_idx, sample_idx) in enumerate(zip(input_example_batch[0][:5], sampled_indices[:5])):
    input_char = index2char[input_idx]  # Get the character from input index
    predicted_char = index2char[sample_idx]  # Get the predicted character

    print(f'Prediction {i:2d}')
    print(f'  input: {input_idx} ({repr(input_char)})')
    print(f'  next predicted: {sample_idx} ({repr(predicted_char)})')
```

```
Prediction  0
  input: 50 ('l')
  next predicted: 46 ('h')
Prediction  1
  input: 54 ('p')
  next predicted: 19 ('G')
Prediction  2
  input: 46 ('h')
  next predicted: 2 ('!')
Prediction  3
  input: 53 ('o')
  next predicted: 42 ('d')
Prediction  4
  input: 57 ('s')
  next predicted: 46 ('h')
```

Model Training

In [38]:
```
# An objective function.
# The function is any callable with the signature scalar_loss = fn(y_true, y_pred).
def loss(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(y_true=labels, y_pred=logits, from_logits=True)
```

In [39]:
```
example_batch_loss = loss(target_example_batch, example_batch_predictions)

print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size, sequence_length, vocab_size)")
print("scalar_loss:      ", example_batch_loss.numpy().mean())
```

```
Prediction shape:  (64, 100, 65)  # (batch_size, sequence_length, vocab_size)
scalar_loss:       4.173642
```

In [40]:
```
# Compile the model
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=adam_optimizer, loss=loss)
```

Configure checkpoints

In [41]:
```
# define the checkpoint
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.keras"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode=
callbacks_list = [checkpoint]
```

Execute the training

In [42]:
```
EPOCHS = 50
```

In [43]:
```
history = model.fit(x=dataset, epochs=EPOCHS, callbacks=callbacks_list)
```

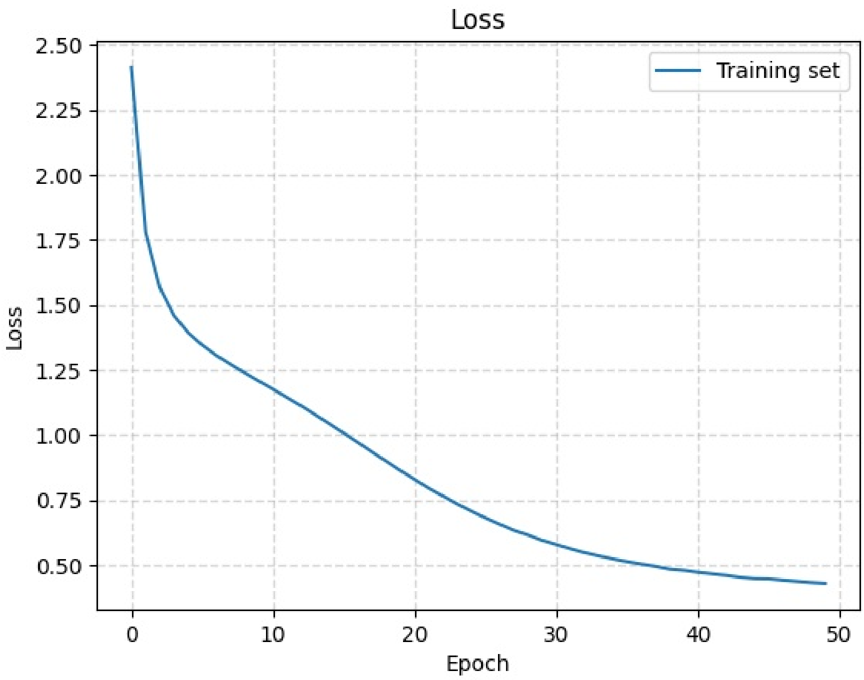
```
Epoch 1/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 2.8661
Epoch 1: loss improved from inf to 2.41399, saving model to weights-improvement-01-2.4140.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 17s 72ms/step - loss: 2.8634
Epoch 2/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 1.8563
Epoch 2: loss improved from 2.41399 to 1.78012, saving model to weights-improvement-02-1.7801.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 14s 73ms/step - loss: 1.8558
Epoch 3/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 1.6016
Epoch 3: loss improved from 1.78012 to 1.56745, saving model to weights-improvement-03-1.5674.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 73ms/step - loss: 1.6014
Epoch 4/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 1.4751
Epoch 4: loss improved from 1.56745 to 1.46019, saving model to weights-improvement-04-1.4602.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 73ms/step - loss: 1.4751
Epoch 5/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 1.4016
Epoch 5: loss improved from 1.46019 to 1.39373, saving model to weights-improvement-05-1.3937.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 76ms/step - loss: 1.4016
Epoch 6/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 1.3454
Epoch 6: loss improved from 1.39373 to 1.34564, saving model to weights-improvement-06-1.3456.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 75ms/step - loss: 1.3454
Epoch 7/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 1.3035
Epoch 7: loss improved from 1.34564 to 1.30594, saving model to weights-improvement-07-1.3059.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 22s 75ms/step - loss: 1.3035
Epoch 8/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 1.2654
Epoch 8: loss improved from 1.30594 to 1.27173, saving model to weights-improvement-08-1.2717.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 19s 75ms/step - loss: 1.2654
Epoch 9/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 1.2318
Epoch 9: loss improved from 1.27173 to 1.23903, saving model to weights-improvement-09-1.2390.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 1.2318
Epoch 10/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 1.1990
Epoch 10: loss improved from 1.23903 to 1.20748, saving model to weights-improvement-10-1.2075.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 1.1991
Epoch 11/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 1.1663
Epoch 11: loss improved from 1.20748 to 1.17667, saving model to weights-improvement-11-1.1767.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 22s 75ms/step - loss: 1.1664
Epoch 12/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 1.1324
Epoch 12: loss improved from 1.17667 to 1.14355, saving model to weights-improvement-12-1.1436.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 76ms/step - loss: 1.1324
```

```
Epoch 13/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 1.0991
Epoch 13: loss improved from 1.14355 to 1.11189, saving model to weights-improvement-13-1.1119.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 75ms/step - loss: 1.0992
Epoch 14/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 1.0649
Epoch 14: loss improved from 1.11189 to 1.07772, saving model to weights-improvement-14-1.0777.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 1.0650
Epoch 15/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 1.0263
Epoch 15: loss improved from 1.07772 to 1.04256, saving model to weights-improvement-15-1.0426.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 1.0264
Epoch 16/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.9914
Epoch 16: loss improved from 1.04256 to 1.00714, saving model to weights-improvement-16-1.0071.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 0.9915
Epoch 17/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.9546
Epoch 17: loss improved from 1.00714 to 0.97165, saving model to weights-improvement-17-0.9716.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 0.9547
Epoch 18/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.9186
Epoch 18: loss improved from 0.97165 to 0.93480, saving model to weights-improvement-18-0.9348.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 0.9187
Epoch 19/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.8819
Epoch 19: loss improved from 0.93480 to 0.89905, saving model to weights-improvement-19-0.8990.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 0.8820
Epoch 20/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.8459
Epoch 20: loss improved from 0.89905 to 0.86327, saving model to weights-improvement-20-0.8633.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 19s 74ms/step - loss: 0.8460
Epoch 21/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.8100
Epoch 21: loss improved from 0.86327 to 0.82965, saving model to weights-improvement-21-0.8296.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 75ms/step - loss: 0.8101
Epoch 22/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.7763
Epoch 22: loss improved from 0.82965 to 0.79541, saving model to weights-improvement-22-0.7954.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 76ms/step - loss: 0.7764
Epoch 23/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.7476
Epoch 23: loss improved from 0.79541 to 0.76573, saving model to weights-improvement-23-0.7657.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 75ms/step - loss: 0.7477
Epoch 24/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.7159
Epoch 24: loss improved from 0.76573 to 0.73524, saving model to weights-improvement-24-0.7352.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 0.7160
Epoch 25/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.6882
Epoch 25: loss improved from 0.73524 to 0.70761, saving model to weights-improvement-25-0.7076.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 75ms/step - loss: 0.6883
Epoch 26/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.6632
Epoch 26: loss improved from 0.70761 to 0.68135, saving model to weights-improvement-26-0.6814.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 76ms/step - loss: 0.6633
Epoch 27/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.6398
Epoch 27: loss improved from 0.68135 to 0.65738, saving model to weights-improvement-27-0.6574.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 0.6399
Epoch 28/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.6140
Epoch 28: loss improved from 0.65738 to 0.63377, saving model to weights-improvement-28-0.6338.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 75ms/step - loss: 0.6141
Epoch 29/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.5971
Epoch 29: loss improved from 0.63377 to 0.61592, saving model to weights-improvement-29-0.6159.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 19s 74ms/step - loss: 0.5972
Epoch 30/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.5781
Epoch 30: loss improved from 0.61592 to 0.59538, saving model to weights-improvement-30-0.5954.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 76ms/step - loss: 0.5782
Epoch 31/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.5618
Epoch 31: loss improved from 0.59538 to 0.57911, saving model to weights-improvement-31-0.5791.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 76ms/step - loss: 0.5619
Epoch 32/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.5475
Epoch 32: loss improved from 0.57911 to 0.56270, saving model to weights-improvement-32-0.5627.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 75ms/step - loss: 0.5476
Epoch 33/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.5313
Epoch 33: loss improved from 0.56270 to 0.54780, saving model to weights-improvement-33-0.5478.keras
```

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 0.5314
Epoch 34/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.5182
Epoch 34: loss improved from 0.54780 to 0.53554, saving model to weights-improvement-34-0.5355.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 76ms/step - loss: 0.5183
Epoch 35/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.5092
Epoch 35: loss improved from 0.53554 to 0.52363, saving model to weights-improvement-35-0.5236.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 0.5093
Epoch 36/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.4957
Epoch 36: loss improved from 0.52363 to 0.51308, saving model to weights-improvement-36-0.5131.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 75ms/step - loss: 0.4958
Epoch 37/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.4878
Epoch 37: loss improved from 0.51308 to 0.50397, saving model to weights-improvement-37-0.5040.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 76ms/step - loss: 0.4879
Epoch 38/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.4810
Epoch 38: loss improved from 0.50397 to 0.49538, saving model to weights-improvement-38-0.4954.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 74ms/step - loss: 0.4811
Epoch 39/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.4694
Epoch 39: loss improved from 0.49538 to 0.48435, saving model to weights-improvement-39-0.4844.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 0.4695
Epoch 40/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - loss: 0.4654
Epoch 40: loss improved from 0.48435 to 0.47975, saving model to weights-improvement-40-0.4797.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 17s 85ms/step - loss: 0.4654
Epoch 41/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.4570
Epoch 41: loss improved from 0.47975 to 0.47244, saving model to weights-improvement-41-0.4724.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 17s 76ms/step - loss: 0.4571
Epoch 42/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.4524
Epoch 42: loss improved from 0.47244 to 0.46634, saving model to weights-improvement-42-0.4663.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 18s 75ms/step - loss: 0.4525
Epoch 43/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.4449
Epoch 43: loss improved from 0.46634 to 0.46048, saving model to weights-improvement-43-0.4605.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 0.4450
Epoch 44/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.4392
Epoch 44: loss improved from 0.46048 to 0.45290, saving model to weights-improvement-44-0.4529.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 22s 76ms/step - loss: 0.4393
Epoch 45/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.4339
Epoch 45: loss improved from 0.45290 to 0.44795, saving model to weights-improvement-45-0.4479.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 76ms/step - loss: 0.4340
Epoch 46/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.4346
Epoch 46: loss improved from 0.44795 to 0.44715, saving model to weights-improvement-46-0.4471.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 74ms/step - loss: 0.4346
Epoch 47/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.4280
Epoch 47: loss improved from 0.44715 to 0.44188, saving model to weights-improvement-47-0.4419.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 22s 74ms/step - loss: 0.4281
Epoch 48/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.4252
Epoch 48: loss improved from 0.44188 to 0.43746, saving model to weights-improvement-48-0.4375.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 76ms/step - loss: 0.4252
Epoch 49/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.4195
Epoch 49: loss improved from 0.43746 to 0.43293, saving model to weights-improvement-49-0.4329.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 15s 75ms/step - loss: 0.4195
Epoch 50/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.4182
Epoch 50: loss improved from 0.43293 to 0.42973, saving model to weights-improvement-50-0.4297.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 73ms/step - loss: 0.4183
```

In [44]:
```python
def render_training_history(training_history):
    loss = training_history.history['loss']
    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(loss, label='Training set')
    plt.legend()
    plt.grid(linestyle='--', linewidth=1, alpha=0.5)
    plt.show()
```

In [45]:
```python
render_training_history(history)
```



In [ ]:

Building and Training a GRU model

In [71]:
```
# define the checkpoint
filepath2 = "weights-improvement-gru-{epoch:02d}-{loss:.4f}.keras"
checkpoint2 = tf.keras.callbacks.ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode=
callbacks_list2 = [checkpoint2]
```

In [72]:
```
def build_gru_model(vocab_size, embedding_dim, rnn_units, batch_size):
    # Define a Sequential model
    model = tf.keras.Sequential()

    # Set the input layer with fixed batch size
    model.add(tf.keras.layers.Input(batch_shape=(batch_size, None)))

    # Embedding layer
    model.add(tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=embedding_dim))

    # GRU layer with stateful=True and a GlorotNormal initializer
    model.add(tf.keras.layers.GRU(
        units=rnn_units,
        return_sequences=True,
        stateful=True,
        recurrent_initializer=tf.keras.initializers.GlorotNormal()
    ))

    # Dense layer with output units equal to vocab size
    model.add(tf.keras.layers.Dense(vocab_size))

    return model
```

In [73]:
```
# Instantiate the GRU model
gru_model = build_gru_model(vocab_size, embedding_dim, rnn_units, BATCH_SIZE)
```

In [74]:
```
gru_model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(64, None, 256)	16,640
gru_2 (GRU)	(64, None, 1024)	3,938,304
dense_5 (Dense)	(64, None, 65)	66,625

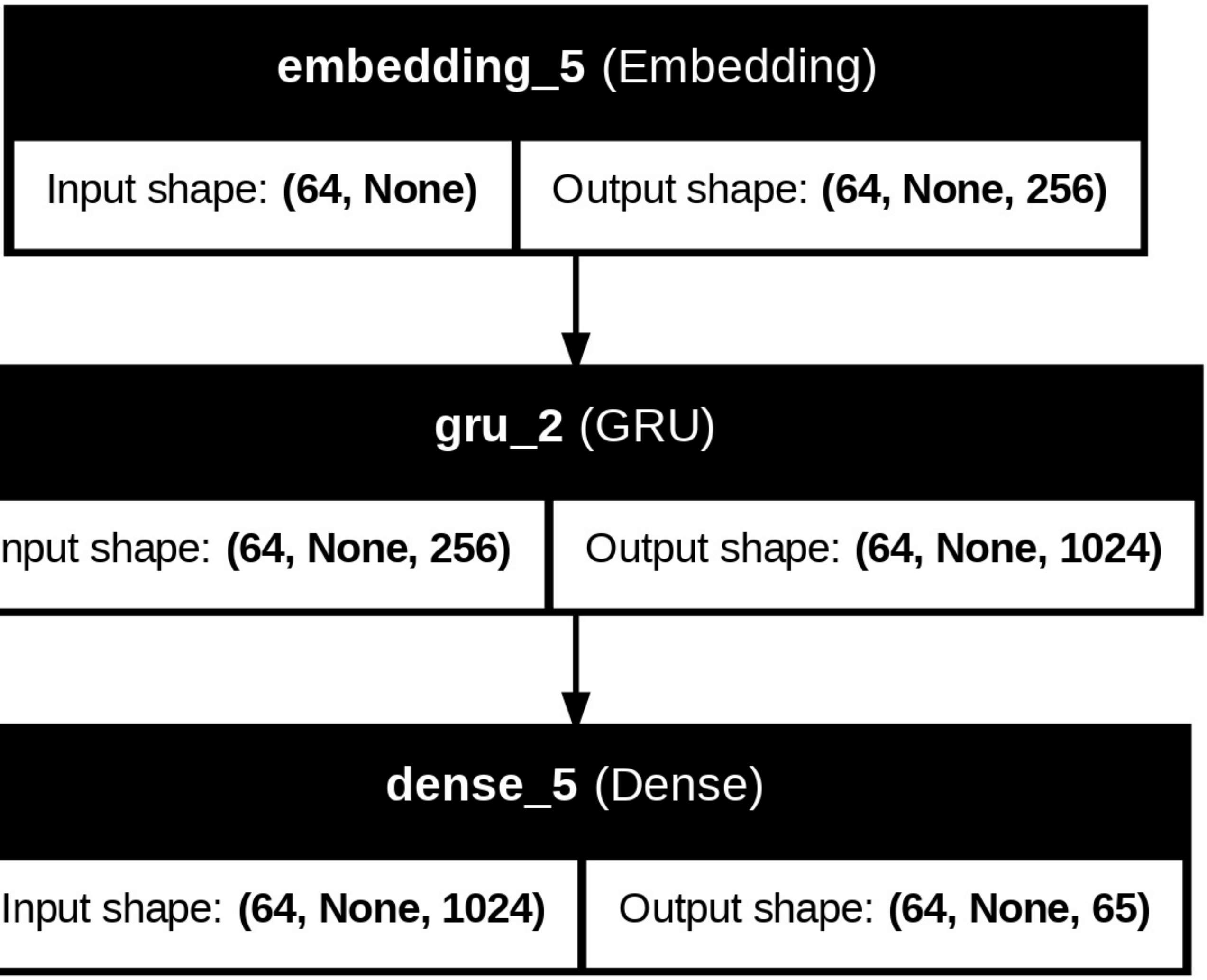
Total params: 4,021,569 (15.34 MB)

Trainable params: 4,021,569 (15.34 MB)

Non-trainable params: 0 (0.00 B)

In [78]:
```python
tf.keras.utils.plot_model(
    gru_model,
    show_shapes=True,
    show_layer_names=True,
)
```

Out[78]:



In [75]:
```python
# An objective function.
# The function is any callable with the signature scalar_loss = fn(y_true, y_pred).
def loss2(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(y_true=labels, y_pred=logits, from_logits=True)
```

In [76]:
```python
# Compile the GRU model
adam_optimizer2 = tf.keras.optimizers.Adam(learning_rate=0.001)
gru_model.compile(optimizer=adam_optimizer2, loss=loss2)
```

In [77]:
```python
# Train the GRU model
history_gru = gru_model.fit(x=dataset, epochs=EPOCHS, callbacks=callbacks_list2, verbose=1)
```

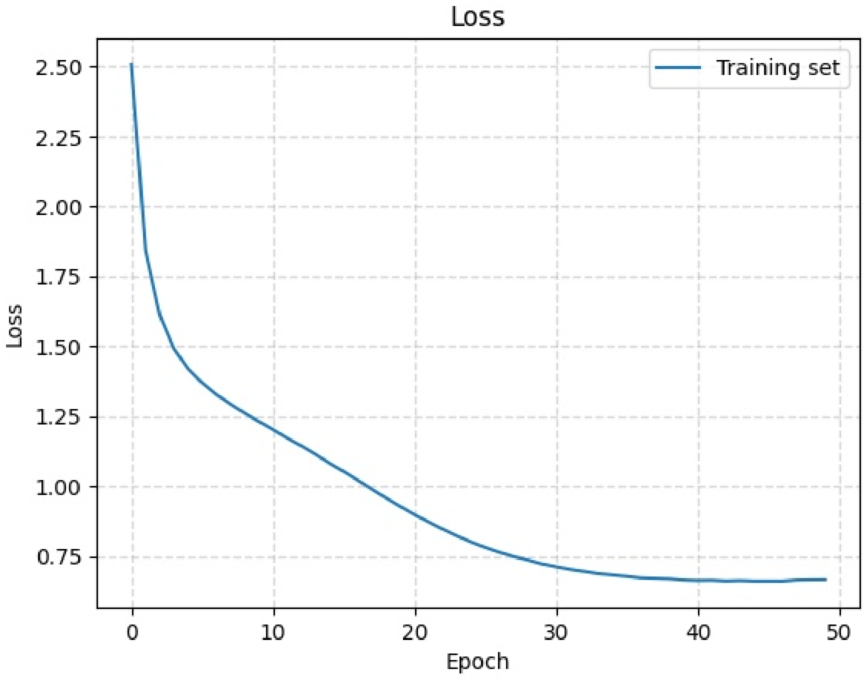
```
Epoch 1/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 3.0951
Epoch 1: loss improved from inf to 2.50778, saving model to weights-improvement-01-2.5078.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 57ms/step - loss: 3.0917
Epoch 2/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 1.9178
Epoch 2: loss improved from 2.50778 to 1.84165, saving model to weights-improvement-02-1.8417.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 58ms/step - loss: 1.9173
Epoch 3/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 1.6505
Epoch 3: loss improved from 1.84165 to 1.61228, saving model to weights-improvement-03-1.6123.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 59ms/step - loss: 1.6503
Epoch 4/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 1.5092
Epoch 4: loss improved from 1.61228 to 1.49205, saving model to weights-improvement-04-1.4920.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 59ms/step - loss: 1.5091
Epoch 5/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 1.4267
Epoch 5: loss improved from 1.49205 to 1.41953, saving model to weights-improvement-05-1.4195.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 60ms/step - loss: 1.4267
Epoch 6/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 1.3692
Epoch 6: loss improved from 1.41953 to 1.36920, saving model to weights-improvement-06-1.3692.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 60ms/step - loss: 1.3692
```

```
Epoch 7/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 1.3234
Epoch 7: loss improved from 1.36920 to 1.32797, saving model to weights-improvement-07-1.3280.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 60ms/step - loss: 1.3234
Epoch 8/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 1.2870
Epoch 8: loss improved from 1.32797 to 1.29322, saving model to weights-improvement-08-1.2932.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 62ms/step - loss: 1.2870
Epoch 9/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 1.2563
Epoch 9: loss improved from 1.29322 to 1.26112, saving model to weights-improvement-09-1.2611.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 62ms/step - loss: 1.2564
Epoch 10/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 1.2216
Epoch 10: loss improved from 1.26112 to 1.23139, saving model to weights-improvement-10-1.2314.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 60ms/step - loss: 1.2216
Epoch 11/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 1.1932
Epoch 11: loss improved from 1.23139 to 1.20285, saving model to weights-improvement-11-1.2028.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - loss: 1.1933
Epoch 12/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 1.1579
Epoch 12: loss improved from 1.20285 to 1.17193, saving model to weights-improvement-12-1.1719.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 62ms/step - loss: 1.1580
Epoch 13/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 1.1296
Epoch 13: loss improved from 1.17193 to 1.14431, saving model to weights-improvement-13-1.1443.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 61ms/step - loss: 1.1297
Epoch 14/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 1.0991
Epoch 14: loss improved from 1.14431 to 1.11458, saving model to weights-improvement-14-1.1146.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 62ms/step - loss: 1.0991
Epoch 15/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 1.0658
Epoch 15: loss improved from 1.11458 to 1.08164, saving model to weights-improvement-15-1.0816.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 62ms/step - loss: 1.0659
Epoch 16/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 1.0354
Epoch 16: loss improved from 1.08164 to 1.05252, saving model to weights-improvement-16-1.0525.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 60ms/step - loss: 1.0355
Epoch 17/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 1.0039
Epoch 17: loss improved from 1.05252 to 1.02147, saving model to weights-improvement-17-1.0215.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - loss: 1.0040
Epoch 18/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.9724
Epoch 18: loss improved from 1.02147 to 0.98877, saving model to weights-improvement-18-0.9888.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 62ms/step - loss: 0.9725
Epoch 19/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.9383
Epoch 19: loss improved from 0.98877 to 0.95752, saving model to weights-improvement-19-0.9575.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 62ms/step - loss: 0.9384
Epoch 20/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.9072
Epoch 20: loss improved from 0.95752 to 0.92765, saving model to weights-improvement-20-0.9277.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 60ms/step - loss: 0.9073
Epoch 21/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.8776
Epoch 21: loss improved from 0.92765 to 0.89865, saving model to weights-improvement-21-0.8986.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 61ms/step - loss: 0.8777
Epoch 22/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.8487
Epoch 22: loss improved from 0.89865 to 0.87192, saving model to weights-improvement-22-0.8719.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 60ms/step - loss: 0.8488
Epoch 23/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.8238
Epoch 23: loss improved from 0.87192 to 0.84649, saving model to weights-improvement-23-0.8465.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - loss: 0.8239
Epoch 24/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.7995
Epoch 24: loss improved from 0.84649 to 0.82250, saving model to weights-improvement-24-0.8225.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 61ms/step - loss: 0.7997
Epoch 25/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.7774
Epoch 25: loss improved from 0.82250 to 0.80046, saving model to weights-improvement-25-0.8005.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - loss: 0.7775
Epoch 26/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.7567
Epoch 26: loss improved from 0.80046 to 0.78104, saving model to weights-improvement-26-0.7810.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - loss: 0.7569
Epoch 27/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.7411
Epoch 27: loss improved from 0.78104 to 0.76388, saving model to weights-improvement-27-0.7639.keras
```

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 60ms/step - loss: 0.7412
Epoch 28/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.7244
Epoch 28: loss improved from 0.76388 to 0.74874, saving model to weights-improvement-28-0.7487.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 22s 59ms/step - loss: 0.7245
Epoch 29/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.7133
Epoch 29: loss improved from 0.74874 to 0.73612, saving model to weights-improvement-29-0.7361.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 19s 60ms/step - loss: 0.7134
Epoch 30/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.6972
Epoch 30: loss improved from 0.73612 to 0.72215, saving model to weights-improvement-30-0.7222.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 60ms/step - loss: 0.6974
Epoch 31/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.6874
Epoch 31: loss improved from 0.72215 to 0.71226, saving model to weights-improvement-31-0.7123.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 60ms/step - loss: 0.6876
Epoch 32/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.6803
Epoch 32: loss improved from 0.71226 to 0.70246, saving model to weights-improvement-32-0.7025.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 61ms/step - loss: 0.6804
Epoch 33/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.6722
Epoch 33: loss improved from 0.70246 to 0.69516, saving model to weights-improvement-33-0.6952.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 19s 60ms/step - loss: 0.6723
Epoch 34/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.6654
Epoch 34: loss improved from 0.69516 to 0.68754, saving model to weights-improvement-34-0.6875.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 60ms/step - loss: 0.6656
Epoch 35/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.6612
Epoch 35: loss improved from 0.68754 to 0.68283, saving model to weights-improvement-35-0.6828.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 63ms/step - loss: 0.6613
Epoch 36/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.6535
Epoch 36: loss improved from 0.68283 to 0.67792, saving model to weights-improvement-36-0.6779.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 62ms/step - loss: 0.6537
Epoch 37/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.6501
Epoch 37: loss improved from 0.67792 to 0.67223, saving model to weights-improvement-37-0.6722.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 60ms/step - loss: 0.6503
Epoch 38/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.6488
Epoch 38: loss improved from 0.67223 to 0.67057, saving model to weights-improvement-38-0.6706.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 60ms/step - loss: 0.6489
Epoch 39/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.6472
Epoch 39: loss improved from 0.67057 to 0.66899, saving model to weights-improvement-39-0.6690.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 61ms/step - loss: 0.6473
Epoch 40/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.6432
Epoch 40: loss improved from 0.66899 to 0.66483, saving model to weights-improvement-40-0.6648.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 61ms/step - loss: 0.6434
Epoch 41/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.6423
Epoch 41: loss improved from 0.66483 to 0.66314, saving model to weights-improvement-41-0.6631.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 63ms/step - loss: 0.6424
Epoch 42/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.6423
Epoch 42: loss did not improve from 0.66314
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 61ms/step - loss: 0.6424
Epoch 43/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.6408
Epoch 43: loss improved from 0.66314 to 0.66101, saving model to weights-improvement-43-0.6610.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 19s 59ms/step - loss: 0.6409
Epoch 44/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.6408
Epoch 44: loss did not improve from 0.66101
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 60ms/step - loss: 0.6409
Epoch 45/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.6428
Epoch 45: loss improved from 0.66101 to 0.66068, saving model to weights-improvement-45-0.6607.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 62ms/step - loss: 0.6429
Epoch 46/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.6395
Epoch 46: loss did not improve from 0.66068
172/172 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - loss: 0.6396
Epoch 47/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.6401
Epoch 47: loss improved from 0.66068 to 0.66056, saving model to weights-improvement-47-0.6606.keras
172/172 ━━━━━━━━━━━━━━━━━━━━ 21s 61ms/step - loss: 0.6402
Epoch 48/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.6425
```

```
Epoch 48: loss did not improve from 0.66056
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 60ms/step - loss: 0.6426
Epoch 49/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.6454
Epoch 49: loss did not improve from 0.66056
172/172 ━━━━━━━━━━━━━━━━━━━━ 13s 61ms/step - loss: 0.6455
Epoch 50/50
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.6461
Epoch 50: loss did not improve from 0.66056
172/172 ━━━━━━━━━━━━━━━━━━━━ 20s 60ms/step - loss: 0.6462
```

In [79]:
```
render_training_history(history_gru)
```



Text Generation

- In this section, we will generate text using the trained RNN model. We'll start with a given string and generate a specified number of characters based on the model's predictions.

Restoring the Model

- First, we need to restore the model from the latest checkpoint to use the trained weights for text generation.
- We will also build the model to accept a batch size of 1 for our predictions.

In [47]:
```
simplified_batch_size = 1

# build the model again with the simplified batch size
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=simplified_batch_size)

# build the model with the expected input shape
model.build(tf.TensorShape([simplified_batch_size, None]))

# load the weights
filename = "/content/weights-improvement-50-0.4297.keras"
model.load_weights(filename)
```

In [48]:
```
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(1, None, 256)	16,640
lstm_2 (LSTM)	(1, None, 1024)	5,246,976
dense_2 (Dense)	(1, None, 65)	66,625

Total params: 5,330,241 (20.33 MB)
Trainable params: 5,330,241 (20.33 MB)
Non-trainable params: 0 (0.00 B)

Text Generation

The prediction loop

The following code block generates the text:

- It Starts by choosing a start string, initializing the RNN state and setting the number of characters to generate.
- Get the prediction distribution of the next character using the start string and the RNN state.
- Then, use a categorical distribution to calculate the index of the predicted character. Use this predicted character as our next input to the model.
- The RNN state returned by the model is fed back into the model so that it now has more context, instead than only one character. After predicting the next character, the modified RNN states are again fed back into the model, which is how it learns as it gets more context from the previously predicted characters.

Text Generation Function

The `generate_text` function generates text using the trained model. It takes a start string and the number of characters to generate.

- **Input**:
  - `model` : The trained RNN model.
  - `start_string` : The string to start generating from.
  - `num_generate` : The total number of characters to generate.
  - `temperature` : Controls the randomness of predictions. Lower values make the model more conservative.
  - Low temperatures results in more predictable text.
  - Higher temperatures results in more surprising text.
  - Experiment to find the best setting.

In [49]:
```
def generate_text(model, start_string, num_generate=1000, temperature=1.0):
    # Evaluation step (generating text using the learned model)

    # Converting our start string to numbers (vectorizing).
    input_indices = [char2index[s] for s in start_string]
    input_indices = tf.expand_dims(input_indices, 0)

    # Empty string to store our results.
    text_generated = []

    # Reset the states of the recurrent layers within the model
    for layer in model.layers:
       if hasattr(layer, 'reset_states'):
           layer.reset_states()

    for char_index in range(num_generate):
        predictions = model(input_indices)
        # remove the batch dimension
        predictions = tf.squeeze(predictions, 0)

        # Using a categorical distribution to predict the character returned by the model.
        predictions = predictions / temperature
        predicted_id = tf.random.categorical(
            predictions,
            num_samples=1
        )[-1, 0].numpy()

        # We pass the predicted character as the next input to the model
        # along with the previous hidden state.
        input_indices = tf.expand_dims([predicted_id], 0)

        text_generated.append(index2char[predicted_id])

    return (start_string + ''.join(text_generated))
```

In [50]:
```
# Generate the text with default temperature (1.0).
print(generate_text(model, start_string=u"ROMEO: ", temperature=1))
```

```
ROMEO: for that I read themselves
Like rags that should endure us broke them more
To swift be itself to be brief wash'd all;
And vouch it to the heavy caused thou strikest me sore wither'd have I in her through they us.

Second Keeper:
But, as it is, Caius Marcius: there my hearts!

TYBALTH:
Which so hang'd up thy friend
Is my poor trade, flesh with the English peers,
That raise his body to the cushion of his mother;
Cry 'Centeracted the king's house, Marcius, whose circums
In he remembering whom we think they take upon
me; I will one nor enemy; you home to crow;
And all comforts are hollow'd friendships.

SOMERSET:
A sixt of all, he's more to purge her forth,
But 'twas the wise for which he play'd it stankedowe a thousand-fold more less;
Therefore die Richard that struck upon thyself?

JULIET:
Farewell! good Pompey. is good night, I would have head
A man well known that we mean to lo;
And he shall scarce call thus, for it good
And be in char he hath shortly of the fire
Of every we to Barthla
```

In [51]:
```python
# Generate the text with higher temperature to get more unexpected results.
print(generate_text(model, start_string=u"BRUTUS: ", temperature=0.8))
```

```
BRUTUS: 'Glay siture from such great majesty,
His statutes and to fear. Now when my soul is an old proportion of your own dreads?

BUCKINGHAM:
No, by my troth, my lord?

KING RICHARD II:
So proud that Angelo hath made to do this wrong.

KING HENRY VI:
Infusing on the old pantal of you
To plant unrightful kings and state of friends.
Now jound, some large enough to be my heir.
What you will have, I'll give, prepared to die.

ISABELLA:
Ay, with favour.

SICINIUS:
Does the urged that we may foreign these poor stars,
An one of you so pale?

O both of the four substitution.

First Lord:
You must not endure him.

ANGELO:
We will make that have more time to come by thy brow
And take it on the ground, which I would I cannot,--
With teach my trust that was hither come so far,
And jocund day befall'd, I would not force the world now with a man
That like a new-hearter of a full as bold in them,
But in the streets, the eyes would do not,
Before you find you out at gates:
So many thoughts of nis should spea
```

In [81]:
```python
simplified_batch_size = 1

# build the model again with the simplified batch size
gru_model = build_gru_model(vocab_size, embedding_dim, rnn_units, batch_size=simplified_batch_size)

# build the model with the expected input shape
gru_model.build(tf.TensorShape([simplified_batch_size, None]))

# load the weights
filename = "/content/weights-improvement-47-0.6606.keras"
gru_model.load_weights(filename)
```

In [82]:
```python
gru_model.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
embedding_7 (Embedding)	(1, None, 256)	16,640
gru_3 (GRU)	(1, None, 1024)	3,938,304
dense_7 (Dense)	(1, None, 65)	66,625

Total params: 4,021,569 (15.34 MB)
Trainable params: 4,021,569 (15.34 MB)
Non-trainable params: 0 (0.00 B)

Generate text with GRU model

In [83]:

```
# Generate the text with default temperature (1.0).
print(generate_text(gru_model, start_string=u"ROMEO: ", temperature=1))
```

```
ROMEO: come Sigor, thou
wilt quench me here lies valiant is all:
I never see the brotor from the hiltest chueack,
And spoke of her and our innatural deaths:
Why after I have kill'd him; we will abtor thee.

CLAUDIO:
Perhaps you have the pence.

GREMIO:
But how past eleven did for the middle and our people
With all my heart, the letter he enough;
But loves not so forfier:

HENRY PERCY:
No rage to make voices. The ne'er was respected heir,
And seen the child-ship will we fight.

LADY ANNE:
Some dunatou I did take these walse camest thou beet,
On me, where away are to't, Warwick,
That fill aid, bring him dead, with
such'd out and seeks that thou so, 'what I think, our city respect,
Which and tumbling summer's well-meaning.

MIRANDA:
O, that's the unwilligency; she's run as limit:
Be patient, gentle Nept to Bianca make him rascal frown,
The brother body shows,
That seest thy spoil: suffer'd, thou never affright?

HESS OF YORK:
Who multitude, here's furbish woman,
Hath let me bear it. As the house
```

In [84]:

```
# Generate the text with higher temperature to get more unexpected results.
print(generate_text(gru_model, start_string=u"BRUTUS: ", temperature=0.8))
```

```
BRUTUS: O prince, is an earthly modest, some pardon
Are of themselves, that to the palace gall'd in the hour,
For she is spoken of my country's light,
Seldoms, and Romeo did before you go;
And now I fear some ill: Signior Placent in the gove; next, that would have held unto the king.

Second Citizen:
Marry, we will bestrew them, and I hate;
But this alliance may shoot?
O, thou look'st on my journey, and must die with me,
But my true love me well, good follow.

First Senator:
D'd you y, but surely.

Second Servant:
O, these are the music of Time.

SICINIUS:
For the marance and the greater fierce hands no foot,
As if the rest were your ancess.
You are treacher! and he shall turn of you;
And with shall prove false friends; him not am I king!
Edward the man, slow, go with me;
Who now came I him in the best, a beggar.

MENENIUS:
Not to him, and leapthee.

CORIOLANUS:
Cut me not, something that is not the king, and rene
Be satisfied, and beguit home:
Now come too lightnfolk from Pardon for it,
And s
```

Save the model

In [85]:
```
model.save('shakespeare_text_gen_lstm.keras')
```

In [86]:
```
gru_model.save('shakespeare_text_gen_gru.keras')
```

In [ ]:

In [ ]:

In [ ]: