# Project Based Learning Report

On

**TEXT** **GENERATION USING RECURRENT NEURAL NETWORK**

Submitted in the partial fulfilment of the requirement

For the Project Based Learning in **Fuzzy Logic, Neural Network and Genetic Algorithms**

In

### Electronics&CommunicationEngineering

By

**2214110407 Pritul Agrawal**

**2214110427 Rajan Kumar**

**2214110446 Ayush Mishra**

Under the guidance of Prof. **V. P Kaduskar**



Department of Electronics & Communication Engineering Bharati Vidyapeeth

(Deemed to be University) College Of Engineering Pune – 410043

Academic Year: 2024-25

## Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune – 411043

**DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING**

## CERTIFICATE

Certified that the Project Based Learning report entitled, **“(TEXT** **GENERATION USING RECURRENT NEURAL NETWORK)”**

is work done by

|  |  |
| --- | --- |
| **2214110407** | **PritulAgrawal** |
| **2214110427** | **Rajan Kumar** |
| **2214110446** | **Ayush Mishra** |

in partial fulfilment of the requirements for the award of credits for Project Based Learning (PBL) in **Fuzzy Logic, Neural Network and Genetic Algorithms** Bachelor of Technology Semester V, in Electronics & Communication Engineering.

**Date:**

**Prof. V. P. Kaduskar Dr.ArundhatiA.Shinde Course In-charge Professor & Head**

# Index

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Topic** | **Page No.** |
| 1. | Problem Statement | 1 |
| 2. | Abstract | 2 |
| 3. | Introduction | 3 |
| 4. | GradientDescent Algorithm | 4 |
| 5. | LinearNeuronModel andCost Function | 5 |
| 6. | LearningRateandIts ImpactonConvergence | 6 |
| 7. | Python Code | 7-10 |
| 8. | Visualizationand Output | 11-12 |
| 9. | Conclusion | 13 |

**Problem Statement**

**Text Generation Using Recurrent Neural Network**

**Objective:** Create a model that can generate coherent text sequence based on an input prompt.

# Abstract

This Project Based Learning explores the text generation using recurrent neural network (RNNs) involves generating sequence of text by predicting the next word based on learned dependencies from previous words. RNNs maintain a hidden state that allow them to remember past input , making them effective for task like language modeling . Advance variations such as long short-term memory (LSTM) and gated recurrent units (GRU) help address challenges like vanishing gradient. While RNNs can generate coherent text , they face limitation with long-range dependencies , leading to the development of newer architectures like transform.

# Introduction

In machine learning, the objective of training a model is to minimize a loss function, which measures the difference between the predicted outputs and the actual data. One of the most common and effective optimization techniques for this Text generation using Recurrent Neural Networks (RNNs) involves teaching a model to predict and produce sequences of text based on patterns it has learned from large amounts of data. RNNs are especially good for tasks where the order of things matters, like sentences or paragraphs, because they can take into account the words that came before to predict what should come next.

The process starts by feeding the RNN a sequence of words (or characters). The network processes one word at a time, and its internal "memory" helps it remember important details about earlier words in the sequence. Based on this memory and what it has learned during training, the RNN generates the next word in the sequence.

Through this approach, RNNs can create coherent text, write stories, or even translate languages, though they often require a lot of training data and can sometimes struggle with longer contexts, which is why newer models like LSTMs and GRUs were developed to improve memory over long sequences.

# Text Generation Using Recurrent Neural Network

### Overview of Text Generation

### Recurrent Neural Network (RNNs) are a type of neural network particularly wellsuited for sequential data, such as text . they process data in sequence, maintaining a hidden state that captures information about previous inputs, which allow them to generatw coherent and contextually relevant text.

### Key Componenet And Formula

### The basic formula for an RNNs at time step t is:

### 1.HIDDEN STATE UPDATE:

### Ht =f(whht-1 + w x xt +b)

### Ht : hidden state at time t

### ht-1 : hidden state from the previous time step

### xt : input at time t

### Wh: weight matrix for the hidden state

### Wx: weight matrix for the input

### b: bias vector

### f: activation function ?9oftern tanh or ReLU)

### 2.OUTPUT GENERATION:

### Yt = softmax ( Wy ht + by)

### Yt: Output vector(often probabilities over possible next characters/words)

### Wy : Weight matrix for the output

### by : bias vector for the output

### Softmax: activation function that covers the output into a probability distribution over the vocabulary for text generation.

### Explanation

### At each time stept,the current hidden state ht is computed using the previous hidden state ht-1 and the current input xt .

### The hidden state ht captures the information from all previous time steps upto t.

### The output yt is then computed using ht , and a softmax function is typically applied to generate a probability distribution over the vocabulary.

### The mostly likely word or character is then chosen based on this distribution ,and it is fed into the network as the input for the next time step .

### 

### This process repeats for each time step , generating one token (character or word ) at a time.

### Types of RNNs

### Recurrent Neural Networks (RNNs) are widely used for various types of text generation tasks due to their ability to handle sequential data. Here are some common types of text generation using RNNs

### Vanilla RNNs: The basic form of RNNs, which can suffer from vanishing gradient problems in long sequences.

### Long Short-Term Memory (LSTM): Designed to handle long-range dependencies better than vanilla RNNs. It uses gates (input, forget, and output) to control the flow of information.

### Gated Recurrent Unit (GRU): A simplified version of LSTM, which combines the input and forget gates into a single update gate

### Character-Level Generation:In this method, the model generates text one character at a time. It learns the probability of the next character based on the previous ones.

### Use case: Generating text in a specific style, like mimicking Shakespeare or other authors

### Word-Level Generation:Here, the model generates text one word at a time, rather than by individual characters. It learns the relationships between words in a sequence.

### Use case: Sentence completion or generating coherent paragraphs.

### Conditional Text Generation:The model generates text based on a given input condition, such as a prompt or a particular topic.

### Use case: Automatic storytelling or code generation, where the generation is guided

### Sequence-to-Sequence (Seq2Seq) Generation:This involves using an encoder-decoder RNN architecture where one sequence (input) is mapped to another sequence (output). It’s often used in tasks like translation or summarization.

### Use case: Machine translation or text summarization.

### Language Modeling:The model generates text based on its understanding of language patterns, predicting the next word in a sequence.

### Use case: Predictive typing or autocomplete features in applications.

### Dialogue Generation:RNNs can be used to generate conversational text, learning how to respond appropriately in a dialogue context.

### Use case: Chatbots or virtual assistants.

### These techniques leverage the ability of RNNs to remember previous context through their internal hidden states, making them suitable for tasks where word order and context matter.

**Learning Rate and Its Impact on Convergence**

### The Role of Learning Rate

### The learning rate is a crucial hyperparameter in training neural networks, including recurrent neural networks (RNNs) used for text generation. It determines the size of the steps taken towards the minimum of the loss function during optimization.

### Too High Learning Rate:

### Overshooting: The model may take excessively large steps, causing it to overshoot the optimal solution. This can lead to divergence, where the loss increases rather than decreases.

### Instability: Training becomes unstable, resulting in erratic fluctuations in loss and model parameters.

### Too Low Learning Rate:

### Slow Convergence: Training can become very slow, requiring many epochs to reach a satisfactory loss level.

### Local Minima: The model may get stuck in local minima, failing to explore better solutions in the

### Optimal Learning Rate:

### Efficient Training: An appropriately set learning rate enables faster convergence to a good solution. The loss decreases smoothly and steadily.

### Balance: It provides a balance between exploration (to find better minima) and exploitation (refining current solutions).

### Effects of different learning rates

### The learning rate significantly impacts the training of recurren neural networks (RNNs) for text:

### 1. Low Learning Rate (e.g., 0.0001):

### Pros: Precise adjustments; suitable for fine-tuning.

### Cons: Slow convergence; risk of getting stuck in local minima.

### 2. Moderate Learning Rate (e.g., 0.001 to 0.01):

### Pros: Balanced and efficient convergence; smoother loss reduction.

### Cons: May still require tuning for optimal performance.

### 3. High Learning Rate (e.g., 0.1 or above):

### Pros: Fast initial training.

### Cons: Overshooting; instability; potential divergence.

### Strategies for Setting Learning Rate

### 1. Learning Rate Schedulers: Adjust the learning rate dynamically based on training progress (e.g., reducing it when the loss plateaus).

### 2. Adaptive Learning Rate Methods: Techniques like Adam or RMS prop automatically adjust the learning rate based on the gradients, often leading to improved convergence behavior.

### In summary, the learning rate significantly affects the training dynamics of RNNs for text generation. Finding the right value is essential for achieving optimal performance and efficiency.

### Visualization of Learning Rates

Visualizations of Learning Rate in RNN Text Generation

**1. Loss Curves:**

**Low Rate:** Slow decrease**.**

**Moderate Rate:** Steady decline**.**

**High Rate:** Erratic fluctuations.

**2. Learning Rate Schedulers:**

Shows adjustments over time, stabilizing training.

**3. Parameter Updates:**

**Low Rate**: Small, stable updates.

**High Rate:** Large, erratic updates**.**

**4. Text Generation Quality:**

**Low Rate:** Gradual improvement.

**Moderate Rate**: High coherence.

**High Rate:** Chaotic outputs**.**

These visualizations highlight how learning rates impact training and text quality in RNNs.

**Libraries Used**

The code utilizes three primary libraries:

* + **NumPy(np)**:For numerical operations, particularly with array
  + **Random:** for generating random number
  + **Tensorflow:** for building and training the neural network

### Functions in the Code

### The function include

* **tf. keras .utils .get\_ file**: Downloads the dataset.
* **open:** Reads the text file.
* **sorted:** Sorts the characters for mapping.
* **Sequential:** Creates a sequential model.
* **LSTM:** Adds an LSTM layer.
* **Dense:** Adds a dense layer.
* **Activation:** Applies the softmax activation function.
* **RMS prop:** Defines the optimizer.
* **model. compile:** Compiles the model with loss and optimizer.
* **model.fit**: Trains the model on the input data.
* **model. predict:** Makes predictions using the trained model.

**Other Functions:**

* **sample:** Samples from the predicted probability distribution.
* **generate\_ text:** Generates text based on the trained model and a specified temperature.

These components work together to create a character-level language model that generates text in the style of Shakespeare.

**Application**

An application of text generation using Recurrent Neural Networks (RNNs) is chatbots. RNN-based chatbots can generate human-like conversations by predicting the next word or sentence based on previous inputs. Here's how it works:

**Customer Service:** An RNN-based chatbot can answer customer queries by generating responses based on the context of the conversation, providing quick and accurate replies. Over time, it learns from previous interactions and improves its ability to respond.

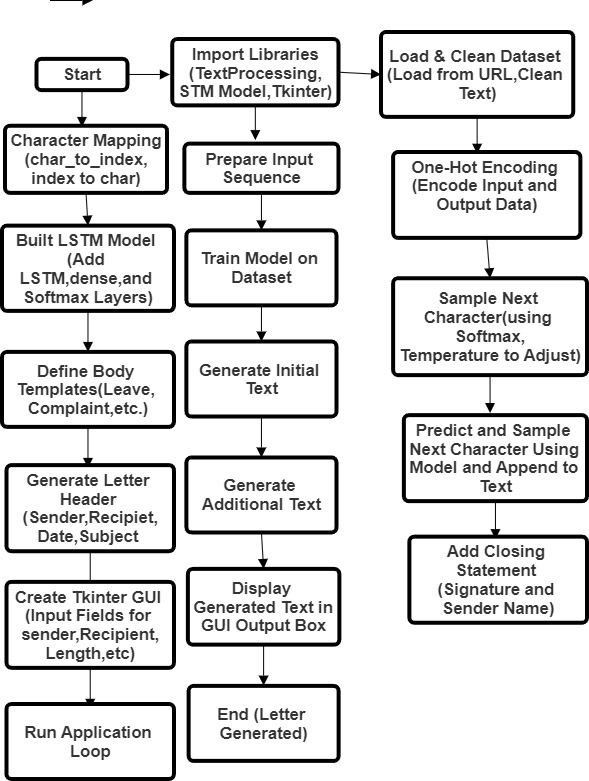
**Content Creation:** RNNs are used to assist writers by generating text, such as articles, stories, or even poetry. Given a prompt, the RNN can predict and generate coherent and contextually appropriate text that follows the narrative style.

**Language Translation:** RNNs can be used in translation applications where a sentence in one language is fed into the network, and the RNN generates its equivalent in another language. The recurrent structure allows the network to maintain a memory of words from earlier in the sentence, ensuring translations are accurate and meaningful.

**Speech Recognition:** Text generation using RNNs plays a role in converting spoken words into text by analyzing audio sequences. The network predicts the most likely sequence of words that corresponds to the given audio input, enabling real-time speech-to-text applications.

These applications utilize RNNs’ ability to handle sequences of data, making them suitable for tasks where context and order are important.

**Flowchart**



**Pseudo Code**

BEGIN

IMPORT required libraries (random, numpy, tensorflow, tkinter, etc.)

FUNCTION decode\_html\_entities(text):

RETURN html.unescape(text)

FUNCTION clean\_text(text):

REMOVE unwanted characters and HTML tags from text

RETURN cleaned text

LOAD dataset from a specified URL

READ text from the file and convert it to lowercase

CLEAN the text using clean\_text function

TRIM text to the first 500,000 characters

INITIALIZE character mappings (char\_to\_index, index\_to\_char)

DEFINE constants for sequence length (SEQ\_LENGTH) and step size (STEP\_SIZE)

INITIALIZE empty lists for sentences and next characters

FOR each index in the text:

APPEND sequences of characters to sentences

APPEND the following character to next\_char

INITIALIZE one-hot encoded arrays x and y for inputs and outputs

FOR each sentence:

ONE-HOT encode the characters and populate x and y arrays

BUILD the LSTM model:

INITIALIZE Sequential model

ADD input layer

ADD LSTM layer with 128 units

ADD Dense layer for output

ADD Activation layer with softmax

COMPILE the model with categorical crossentropy loss and RMSprop optimizer

TRAIN the model on the dataset

FUNCTION sample(preds, temperature):

CONVERT predictions to probabilities based on temperature

RETURN the index of the sampled character

FUNCTION get\_fixed\_body\_template(purpose):

RETURN a predefined body template based on the purpose of the letter

FUNCTION generate\_text(length, temperature, recipient, sender, purpose):

RANDOMLY select a starting index from the text

CALL get\_fixed\_body\_template(purpose) to get fixed body message

GENERATE letter header with sender and receiver addresses

INITIALIZE generated text with a sequence from the text

APPEND the fixed body message to generated text

FOR length of text to generate:

ONE-HOT encode the current sentence

PREDICT the next character using the model

SAMPLE the next character based on predictions

UPDATE the generated text with the new character

ADD closing statement with the sender's name

RETURN the complete generated text

FUNCTION run\_app():

CREATE a GUI window using Tkinter

DEFINE input fields for recipient, sender, purpose, text length, and temperature

FUNCTION on\_generate\_text():

TRY to retrieve input values and generate the letter

CALL generate\_text function

DECODE generated text and display it in the output box

EXCEPT ValueError:

DISPLAY an error message

INITIALIZE the main window and widgets

START the Tkinter event loop

CALL run\_app() to start the application

END

**Code**

import random

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.layers import Activation, Dense, LSTM, Input

import tkinter as tk

from tkinter import ttk

import html

import re

from datetime import datetime

# Decoding function for HTML entities

def decode\_html\_entities(text):

return html.unescape(text)

# Function to clean the text data

def clean\_text(text):

# Remove any unwanted characters and HTML tags

text = re.sub(r'\s+', ' ', text) # Replace multiple spaces/newlines with a single space

text = re.sub(r'<.\*?>', '', text) # Remove HTML tags

text = re.sub(r'[^a-zA-Z0-9.,!?\'\"\- ]', '', text) # Remove any non-alphanumeric characters except punctuation

return text

# Load the dataset of formal letters or applications

filepath = tf.keras.utils.get\_file(

'formal\_letters.txt',

'https://raw.githubusercontent.com/ayushmish2990/Mail-Format/main/Formal%20letter%20format.txt'

)

text = open(filepath, 'r', encoding='utf-8').read().lower()

# Clean the dataset text

text = clean\_text(text)

# Use a portion of the text for training

text = text[:500000] # Use the first 500,000 characters

# Character mapping

characters = sorted(set(text))

char\_to\_index = {c: i for i, c in enumerate(characters)}

index\_to\_char = {i: c for i, c in enumerate(characters)}

# Define sequence length and step size

SEQ\_LENGTH = 40

STEP\_SIZE = 3

# Prepare input sequences and corresponding next characters

sentences = []

next\_char = []

for i in range(0, len(text) - SEQ\_LENGTH, STEP\_SIZE):

sentences.append(text[i: i + SEQ\_LENGTH])

next\_char.append(text[i + SEQ\_LENGTH])

# One-hot encoding for input and output

x = np.zeros((len(sentences), SEQ\_LENGTH, len(characters)), dtype=np.bool\_)

y = np.zeros((len(sentences), len(characters)), dtype=np.bool\_)

for i, sentence in enumerate(sentences):

for t, char in enumerate(sentence):

x[i, t, char\_to\_index[char]] = True

y[i, char\_to\_index[next\_char[i]]] = True

# Build the model

model = Sequential()

model.add(Input(shape=(SEQ\_LENGTH, len(characters)))) # Define the input layer explicitly

model.add(LSTM(128))

model.add(Dense(len(characters)))

model.add(Activation('softmax'))

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer=RMSprop(learning\_rate=0.01))

# Train the model

model.fit(x, y, batch\_size=256, epochs=4)

# Sampling function

def sample(preds, temperature=1.0):

preds = np.asarray(preds).astype('float64')

preds = np.log(preds) / temperature

exp\_preds = np.exp(preds)

preds = exp\_preds / np.sum(exp\_preds)

probas = np.random.multinomial(1, preds, 1)

return np.argmax(probas)

# Define different fixed body templates

def get\_fixed\_body\_template(purpose):

if "leave" in purpose.lower():

return (

"\n\nI would like to kindly request a leave of absence from work "

"for the following reasons. I will ensure that all my duties and "

"responsibilities are properly handed over before my departure, "

"and I will make sure to return by the specified date."

)

elif "complaint" in purpose.lower():

return (

"\n\nI am writing to formally lodge a complaint regarding the issue. "

"I hope that you will look into this matter at the earliest and "

"take the necessary actions to resolve the issue as soon as possible."

)

elif "enquiry" in purpose.lower():

return (

"\n\nThis is with reference to your advertisement in the ‘The Times of India’ for CAT Coaching classes."

" I have passed the B.Sc. degree examination with Statistics as the main subject."

"I am keen on joining your institute for the coaching classes."

"Kindly let me know about the procedure of applying for the qualifying test and its date."

"I would like to enroll as soon as possible. Your early response will enable me to decide fast."

)

elif "order" in purpose.lower():

return (

"\n\nThis is with reference to the Order No.() placed on Nov 17, 20xx."

"The order consists of letterhead and business cards."

"As per the agreement, we were promised to receive the order by Nov 22, 20xx."

"The order did not reach on time, and the quality of the papers and design selected for business cards "

"does not match the one selected. We faced a lot of embarrassment and inconvenience, and our reputation "

"is at stake in the eyes of our clients. Kindly ensure that the order will be replaced by Dec 4, 20xx."

)

elif "promotion" in purpose.lower():

return (

"\n\nWe are glad to announce the grand opening of a new branch of our company in QPR Colony, Delhi on Dec 05, 20xx."

"As a respected client, we are delighted to inform you that this branch offers various solutions to your problems."

"We are dedicated to providing you with the best service and would be happy to have you as our guest."

)

elif "application" in purpose.lower():

return (

"\n\nI am submitting this application for the position that is available. "

"I have attached my resume and relevant documents for your consideration. "

"I am eager to discuss how my skills can contribute to your organization."

)

elif "invitation" in purpose.lower():

return (

"\n\nIt is with great pleasure that I invite you to the event being held on "

"[date]. We would be honored to have your presence at this occasion, and I "

"look forward to your confirmation of attendance."

)

return (

"\n\nI am writing to you regarding the above-mentioned subject. "

"I would like to formally request your attention to this matter and provide the necessary assistance. "

"Your prompt response and cooperation would be highly appreciated."

)

# Text generation function for letter/application format

def generate\_text(length, temperature, recipient, sender, purpose):

start\_index = random.randint(0, len(text) - SEQ\_LENGTH - 1)

# Get the fixed body message based on the purpose

fixed\_body\_message = get\_fixed\_body\_template(purpose)

# Generate the header with sender's address, date, and receiver's address

date\_today = datetime.now().strftime("%B %d, %Y")

generated = (

f"Sender's Address:\n[Your Address Here]\n\n"

f"Date: {date\_today}\n\n"

f"Receiver's Address:\n{recipient}\n\n"

f"Subject: {purpose}\n\n"

f"Sir/Madam,\n\n"

)

sentence = text[start\_index: start\_index + SEQ\_LENGTH]

generated += sentence

# Insert the fixed body message after the initial generated text

generated += fixed\_body\_message

# Generate additional text

for \_ in range(length):

x\_predictions = np.zeros((1, SEQ\_LENGTH, len(characters)))

for t, char in enumerate(sentence):

x\_predictions[0, t, char\_to\_index[char]] = 1

predictions = model.predict(x\_predictions, verbose=0)[0]

next\_index = sample(predictions, temperature)

next\_character = index\_to\_char[next\_index]

generated += next\_character

sentence = sentence[1:] + next\_character

# Add closing with sender's name

generated += f"\n\nSincerely,\n{sender}"

return generated

# Create the application window using Tkinter

def run\_app():

# Function to generate letter based on user input and display it

def on\_generate\_text():

try:

length = int(length\_entry.get())

temperature = float(temp\_entry.get())

recipient = recipient\_entry.get()

sender = sender\_entry.get()

purpose = purpose\_entry.get()

generated\_text = generate\_text(length, temperature, recipient, sender, purpose)

# Decode the generated text before displaying

decoded\_text = decode\_html\_entities(generated\_text)

output\_text.delete('1.0', tk.END) # Clear the previous output

output\_text.insert(tk.END, decoded\_text)

except ValueError:

output\_text.delete('1.0', tk.END)

output\_text.insert(tk.END, "Invalid input! Please enter valid numbers.")

# Create the main window

window = tk.Tk()

window.title("Formal Letter/Application Generator")

# Input fields for recipient, sender, and purpose

recipient\_label = ttk.Label(window, text="Recipient Name:")

recipient\_label.grid(column=0, row=0, padx=10, pady=10)

recipient\_entry = ttk.Entry(window)

recipient\_entry.grid(column=1, row=0, padx=10, pady=10)

sender\_label = ttk.Label(window, text="Sender Name:")

sender\_label.grid(column=0, row=1, padx=10, pady=10)

sender\_entry = ttk.Entry(window)

sender\_entry.grid(column=1, row=1, padx=10, pady=10)

purpose\_label = ttk.Label(window, text="Purpose of Letter:")

purpose\_label.grid(column=0, row=2, padx=10, pady=10)

purpose\_entry = ttk.Entry(window)

purpose\_entry.grid(column=1, row=2, padx=10, pady=10)

# Input fields for text generation length and temperature

length\_label = ttk.Label(window, text="Length of Text:")

length\_label.grid(column=0, row=3, padx=10, pady=10)

length\_entry = ttk.Entry(window)

length\_entry.grid(column=1, row=3, padx=10, pady=10)

length\_entry.insert(0, '300') # Default value

temp\_label = ttk.Label(window, text="Temperature (0.1 - 1.0):")

temp\_label.grid(column=0, row=4, padx=10, pady=10)

temp\_entry = ttk.Entry(window)

temp\_entry.grid(column=1, row=4, padx=10, pady=10)

temp\_entry.insert(0, '0.5') # Default value

# Button to generate text

generate\_button = ttk.Button(window, text="Generate Letter", command=on\_generate\_text)

generate\_button.grid(column=0, row=5, columnspan=2, padx=10, pady=10)

# Output text box to display the generated text

output\_text = tk.Text(window, wrap='word', width=60, height=20)

output\_text.grid(column=0, row=6, columnspan=2, padx=10, pady=10)

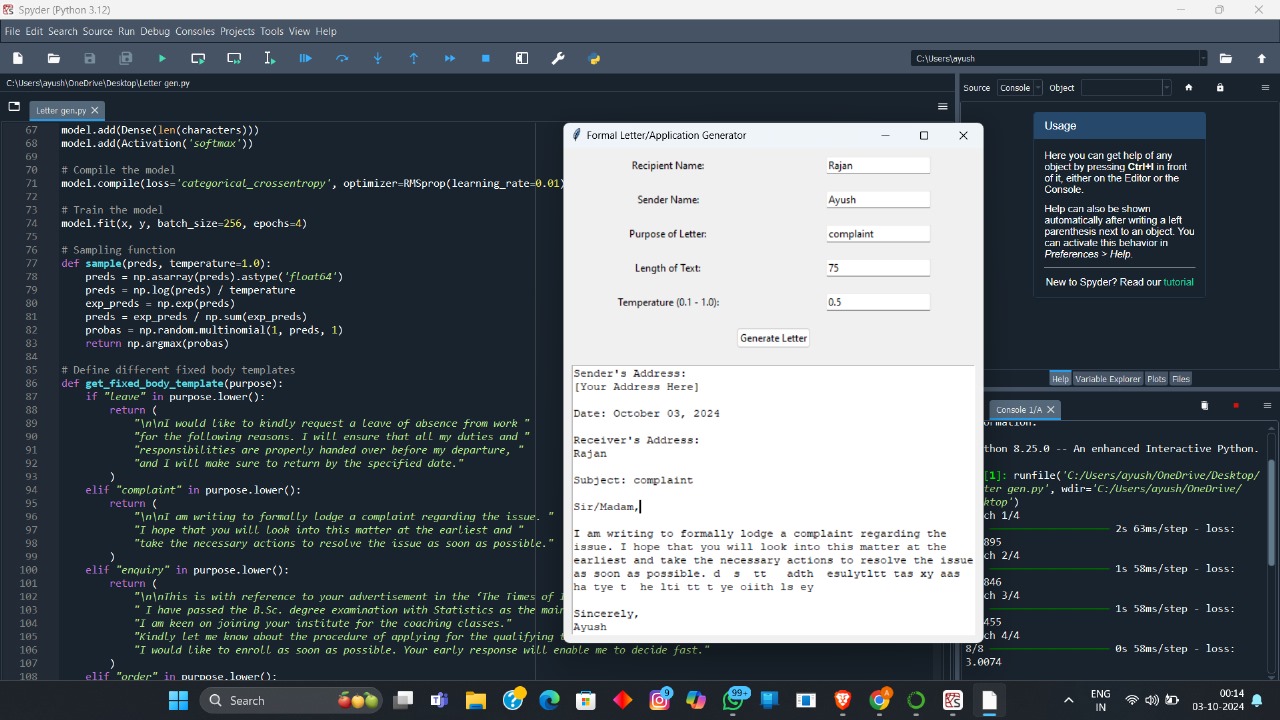
# Start the Tkinter event loop

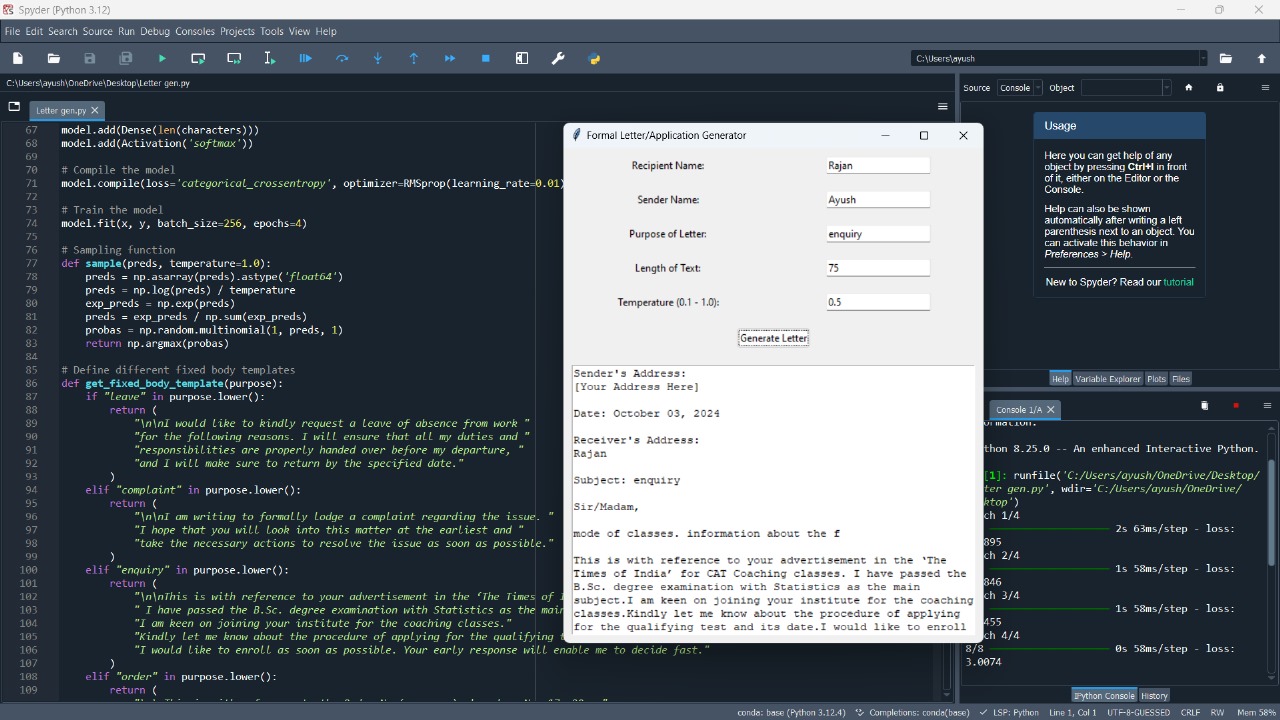
window.mainloop()

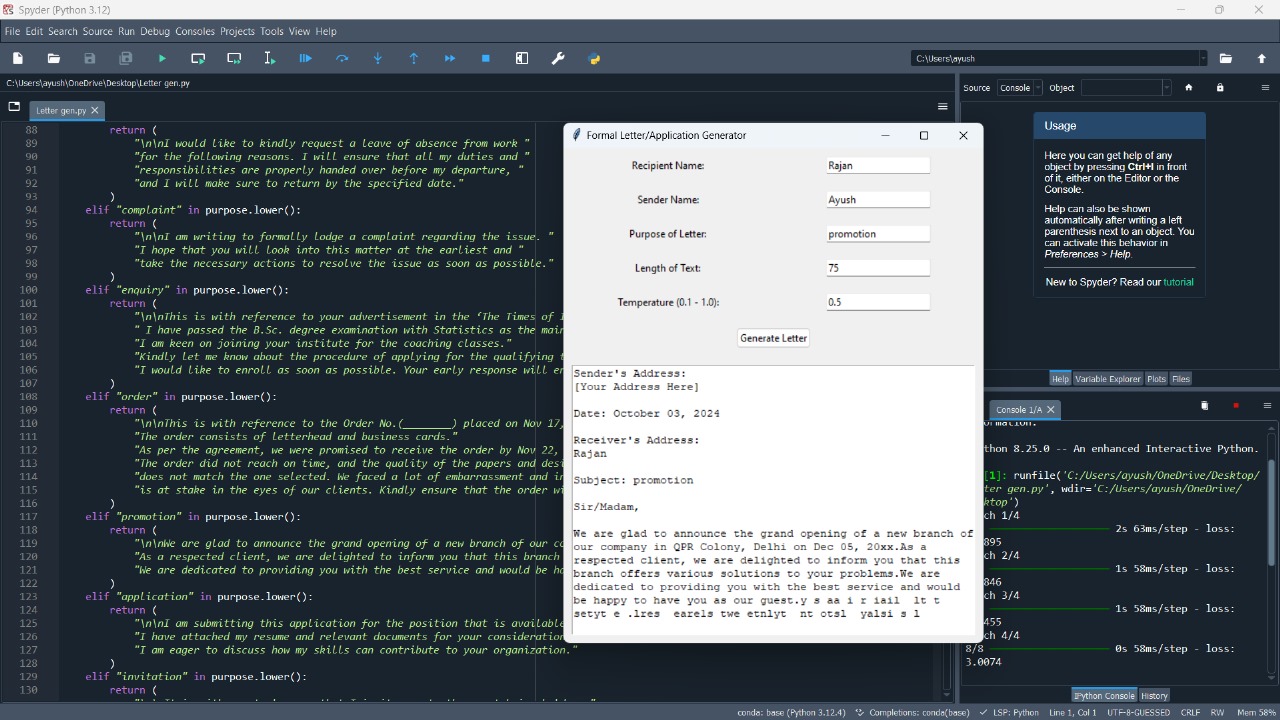
# Run the app

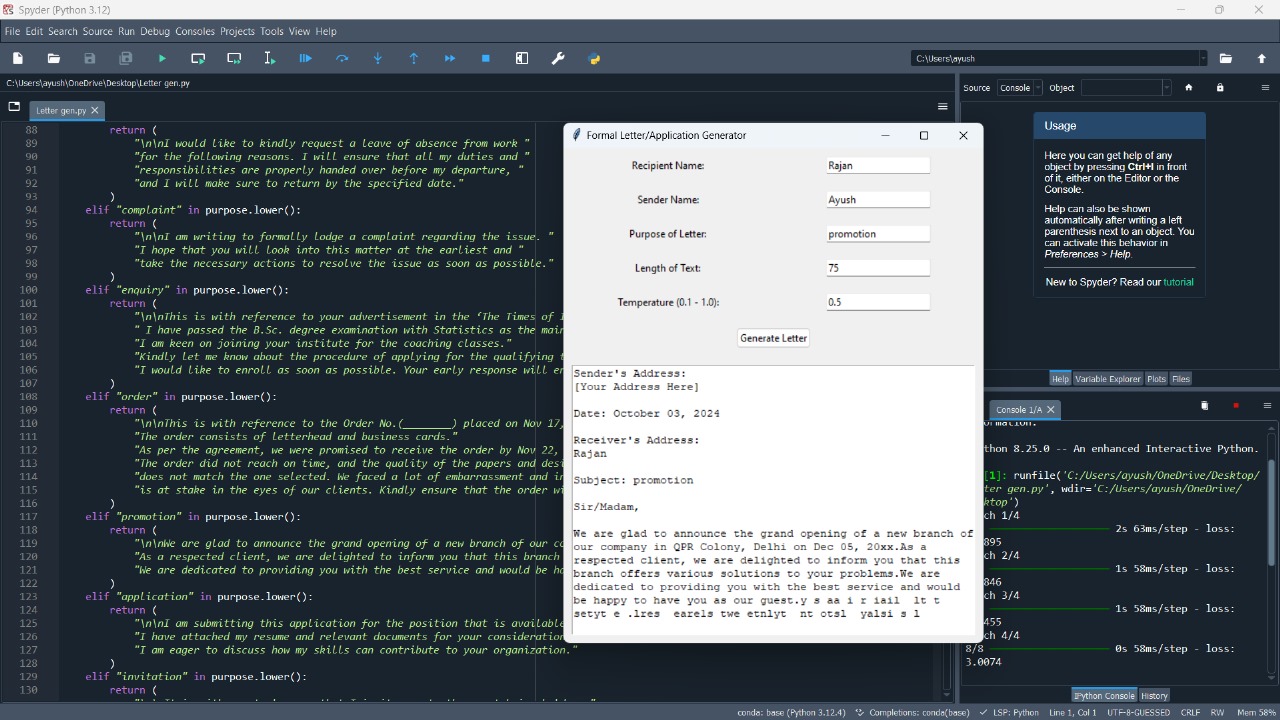
run\_app()

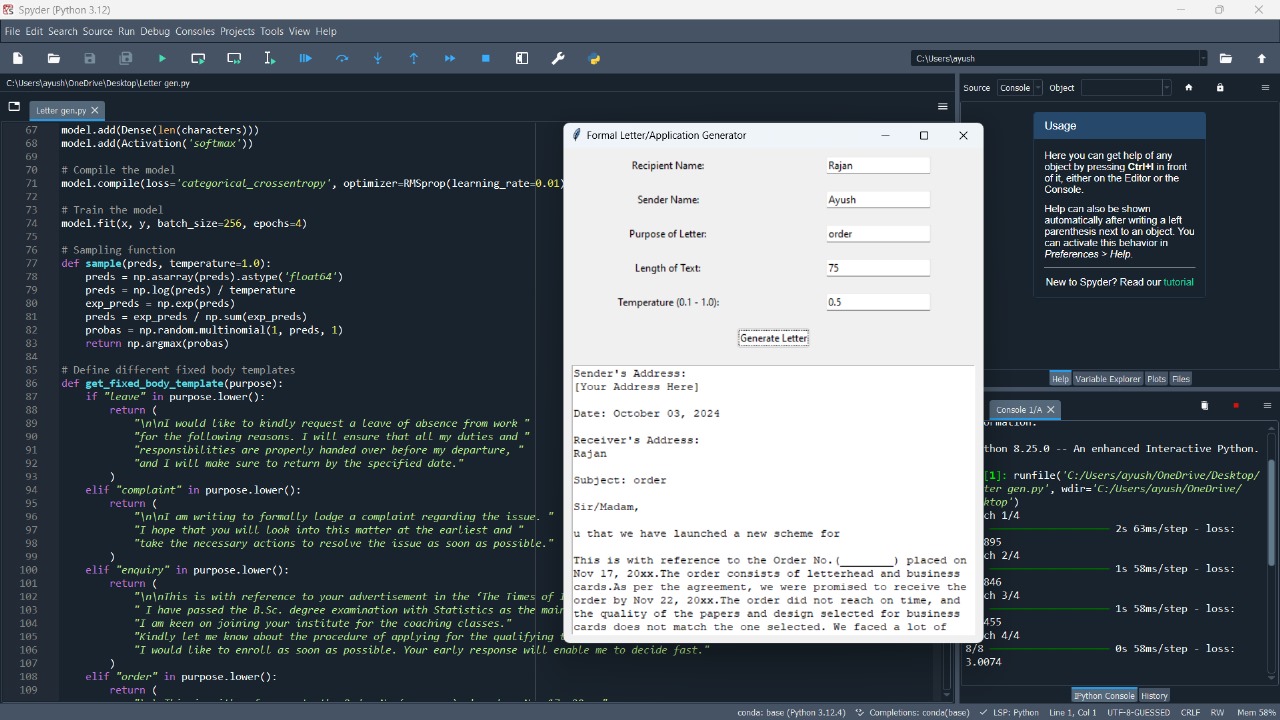
**Visualization and** **Output**

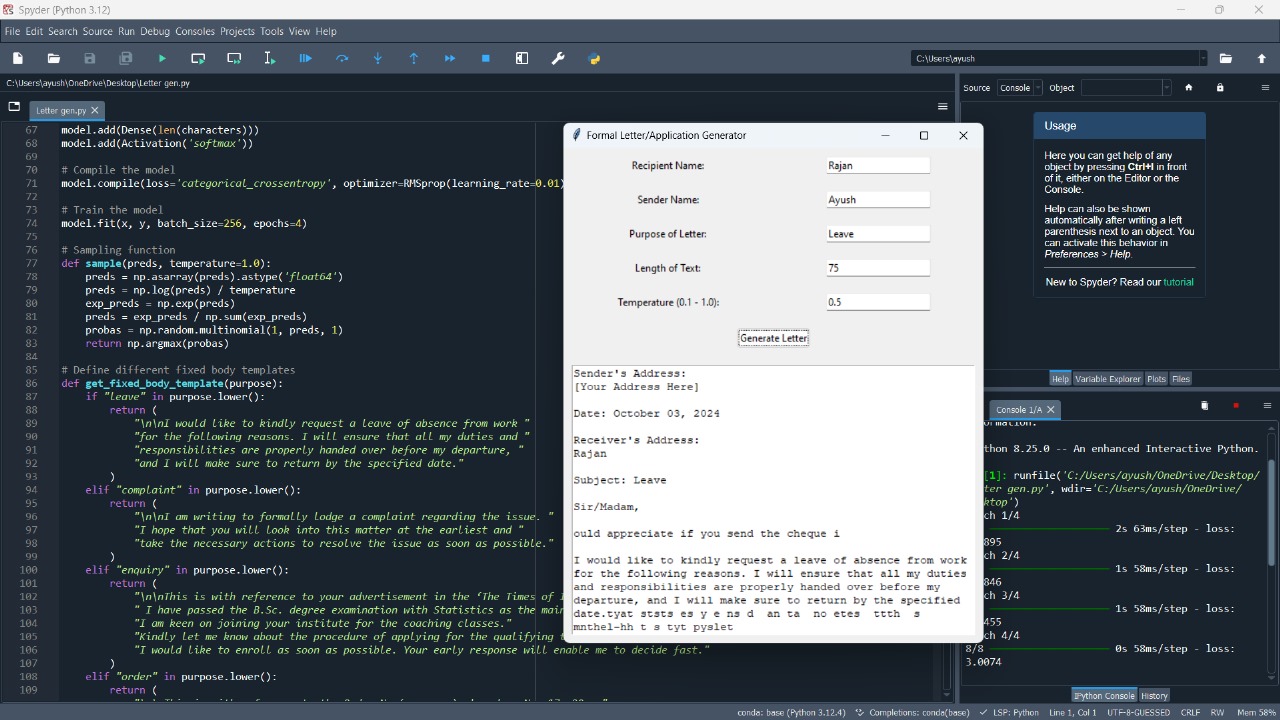


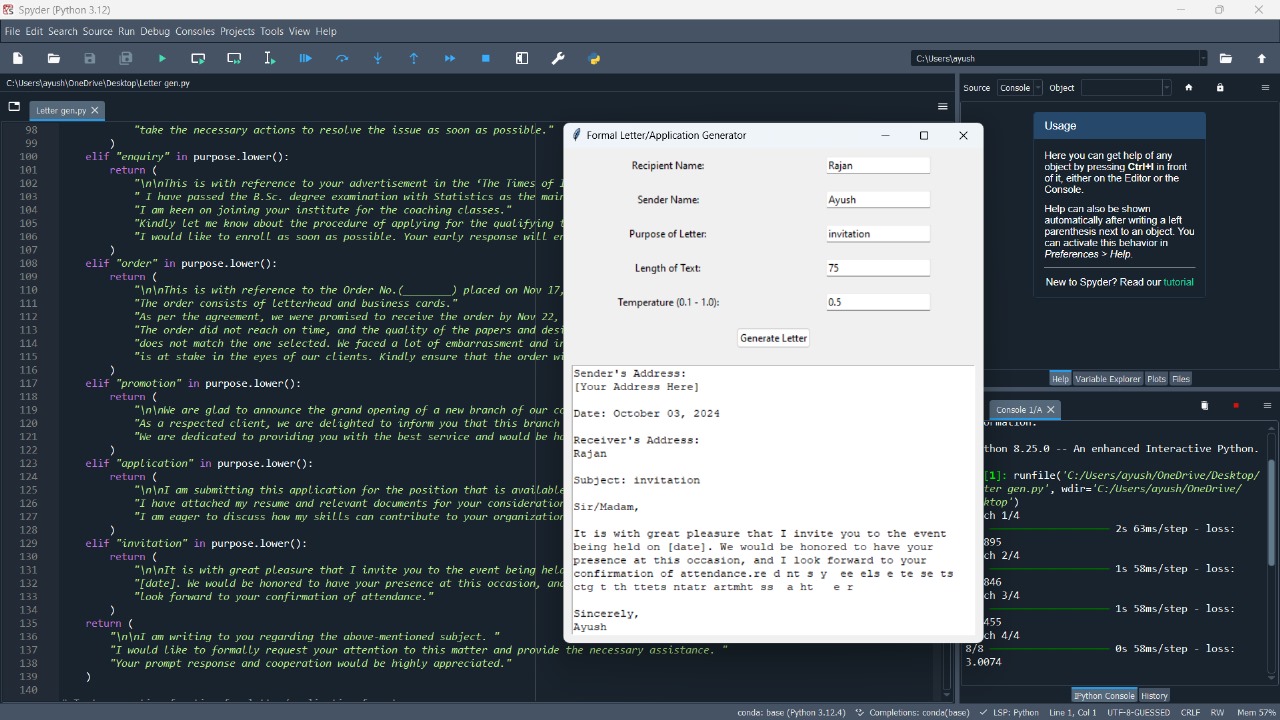


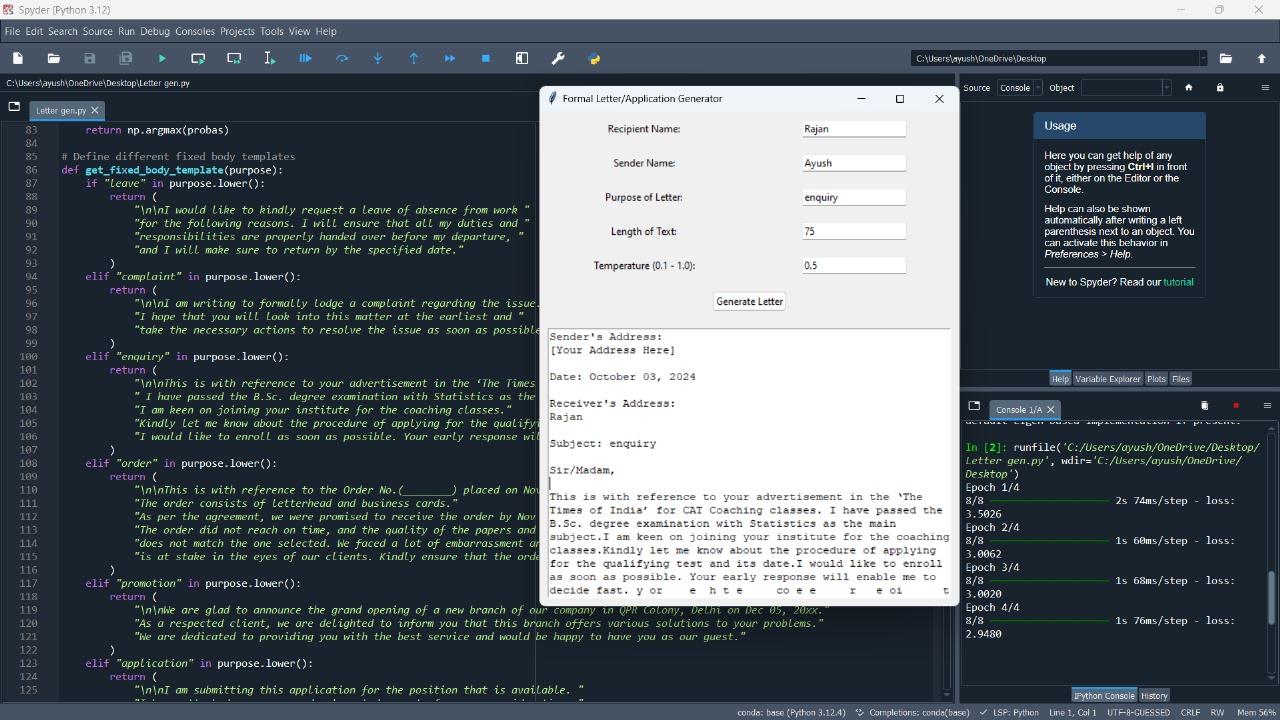












**Algorithm**

Here’s an algorithm to describe the formal letter/application generator process in a structured manner.

Algorithm: Formal Letter/Application Generator

**1. Import Necessary Libraries**

Import libraries for text processing, deep learning (LSTM model), and GUI creation (Tkinter).

**2. Define Helper Functions**

decode\_html\_entities(text)

Input: text (a string with potential HTML entities).

Output: Decoded text with HTML entities converted to their respective characters.

clean\_text(text)

Input: text (raw input text data).

Process:

Remove unwanted characters, multiple spaces, and HTML tags.

Output: Cleaned and processed text.

**3. Load and Preprocess Dataset**

Load dataset\* from a specified URL.

Convert text\* to lowercase and cleanusing the clean\_text function.

Trim the dataset\* to the first 500,000 characters.

**4. Character Mapping**

Create a set of unique characters in the text.

Map characters to indices (for the model) and vice-versa:

char\_to\_index: Dictionary mapping each character to an index.

index\_to\_char: Dictionary mapping each index back to its corresponding character.

**5. Prepare Input and Output Sequences**

Define sequence length (SEQ\_LENGTH) and step size (STEP\_SIZE).

Initialize two lists:

sentences: Holds sequences of characters (input data).

next\_char: Holds the character following each sequence (output data).

Loop through the text to create sequences:

Append each sequence of length SEQ\_LENGTH to sentences.

Append the character following the sequence to next\_char.

**6. One-Hot Encode Input and Output**

Initialize two numpy arrays:

x: One-hot encoding for input sequences (shape: number of sequences × sequence length × number of characters).

y: One-hot encoding for next characters (shape: number of sequences × number of characters).

Encode sequences and next characters into x and y matrices using char\_to\_index.

**7. Build and Train LSTM Model**

Define a Sequential model:

1. Add an LSTM layer with 128 units.

2. Add a Dense layer for output (equal to the number of unique characters).

3. Add a softmax activation function to compute probabilities.

Compile the model using categorical cross-entropy loss and RMSprop optimizer.

Train the model using the input x and output y for a specified number of epochs.

**8. Define Text Sampling Function**

1. sample(predictions, temperature)

Input: predictions (model’s predicted probabilities for next character) and temperature (a control for randomness).

Process:

Convert predictions into probabilities.

Sample the next character based on the adjusted probabilities.

Output: Index of the sampled character.

**9. Define Fixed Body Templates**

get\_fixed\_body\_template(purpose)

Input: purpose (the purpose of the letter).

Process:

Return a predefined body template based on the purpose (leave, complaint, order, etc.).

Output: A text template that fits the given purpose.

**10. Generate Letter Text**

1. generate\_text(length, temperature, recipient, sender, purpose)

Input: length (how many characters to generate), temperature (randomness), recipient, sender, and purpose.

Process:

1. Select a random starting point in the text.

2. Get the fixed body message for the letter using get\_fixed\_body\_template(purpose).

3. Generate letter header (sender/recipient addresses, date, subject).

4. Append a text sequence (initial part of the letter) from the dataset.

5. Insert the fixed body message after the generated text.

6. Loop to generate additional text (using the LSTM model predictions):

Use one-hot encoded input to predict the next character.

Sample the character and append it to the letter.

7. Add closing statement and sender’s name.

Output: The full letter text.

**11. Create Tkinter GUI**

1. run\_app()

Define a GUI window with input fields for:

Recipient name, sender name, purpose of the letter, length of text, and temperature.

Define an output text box to display the generated letter.

Function on\_generate\_text()

Retrieve user input.

Call generate\_text() to generate the letter.

Display the generated letter in the output text box.

Start the Tkinter event loop.

**12. Run the Application**

Call run\_app() to start the GUI and allow user interaction.

Time Complexity Analysis:

Data Preprocessing: Iterating through the text to create sequences has a complexity of O(n), where n is the number of characters in the text.

Model Training: Training an LSTM model typically involves O(n × m), where n is the number of sequences and m is the complexity of backpropagation in each LSTM layer (related to the number of parameters).

Text Generation: For each character to be generated, the LSTM model needs O(m) operations, where m is the number of parameters in the model.

**13. Space Complexity**:

The space complexity is O(n) for storing the dataset, O(n) for the one-hot encoding of inputs, and O(m) for storing the LSTM model parameters. The overall complexity depends on the size of the dataset and the model.

**Conclusion**

In conclusion, text generation using recurrent neural networks (RNNs) represents a significant advancement in natural language processing. RNNs, particularly with the incorporation of long short-term memory (LSTM) and gated recurrent units (GRUs), effectively capture sequential dependencies in text, enabling the generation of coherent and contextually relevant sentences. Despite their strengths, RNNs can struggle with long-range dependencies and may face challenges with computational efficiency. However, with ongoing research and the development of more sophisticated architectures, such as transformers, RNNs continue to play a crucial role in text generation tasks. Their ability to learn patterns from large datasets makes them valuable for applications ranging from conversational agents to creative writing.

**Github link:**