**HACKATHON LEVEL-2**

**GEN-AI**

**GitHub link :** [**https://github.com/karthick2005-alpha/GenAI-Hackathon**](https://github.com/karthick2005-alpha/GenAI-Hackathon)

**Name: KARTHICK M**

**Reg No: 620122114024**

**College name : Avs engineering college**

**Department : Mechanical engineering**

**1. Basic Text Generation Using LSTM**

**We can build a simple text generation model using LSTM. Here’s an example using TensorFlow/Keras:**

**Step 1:**

**Install necessary libraries First, make sure you have the required libraries installed:**

**Python code**

**pip install tensorflow numpy**

**Step 2:**

**Prepare the Text Data We'll use a text dataset, tokenize it, and prepare it for the model.**

**Python code**

**import tensorflow as tf**

**import numpy as np**

**import os**

**import time**

**# Load dataset (you can use any text dataset; here we use Shakespeare's work for simplicity)**

**path\_to\_file = 'shakespeare.txt' # Path to your text file**

**with open(path\_to\_file, 'r') as f:**

**text = f.read()**

**# Create a mapping of unique characters to indices and vice versa**

**vocab = sorted(set(text)) # All unique characters in the text**

**char\_to\_index = {char: index for index, char in enumerate(vocab)}**

**index\_to\_char = {index: char for index, char in enumerate(vocab)}**

**# Convert text into integers**

**text\_as\_int = np.array([char\_to\_index[c] for c in text])**

**# Define the length of input sequences and the batch size**

**seq\_length = 100**

**batch\_size = 64**

**# Create training sequences and labels**

**def create\_sequences(text\_as\_int, seq\_length):**

**input\_seq = []**

**output\_seq = []**

**for i in range(0, len(text\_as\_int) - seq\_length, 1):**

**input\_seq.append(text\_as\_int[i:i + seq\_length])**

**output\_seq.append(text\_as\_int[i + seq\_length])**

**return np.array(input\_seq), np.array(output\_seq)**

**X, y = create\_sequences(text\_as\_int, seq\_length)**

**# Reshape X for the model**

**X = np.reshape(X, (X.shape[0], X.shape[1], 1)) # (num\_samples, seq\_length, 1)**

**X = X / float(len(vocab)) # Normalize input**

**# Define the LSTM model**

**model = tf.keras.Sequential([**

**tf.keras.layers.LSTM(256, input\_shape=(X.shape[1], X.shape[2]), return\_sequences=True),**

**tf.keras.layers.LSTM(256),**

**tf.keras.layers.Dense(len(vocab), activation='softmax')**

**])**

**model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam')**

**# Train the model**

**model.fit(X, y, batch\_size=batch\_size, epochs=50)**

**Step 3:**

**Generate Text Once the model is trained, we can generate text.**

**Python code**

**# Generate text from the trained model**

**def generate\_text(model, start\_string, char\_to\_index, index\_to\_char, num\_generate=500):**

**input\_eval = [char\_to\_index[s] for s in start\_string]**

**input\_eval = np.expand\_dims(input\_eval, 0)**

**predicted\_text = start\_string**

**model.reset\_states()**

**for \_ in range(num\_generate):**

**predictions = model(input\_eval)**

**predictions = predictions[:, -1, :] # Get the last timestep prediction**

**predicted\_id = tf.random.categorical(predictions, num\_samples=1)[-1, 0].numpy()**

**predicted\_char = index\_to\_char[predicted\_id]**

**predicted\_text += predicted\_char**

**input\_eval = np.expand\_dims([predicted\_id], 0)**

**return predicted\_text**

**start\_string = "ROMEO: "**

**generated\_text = generate\_text(model, start\_string, char\_to\_index, index\_to\_char)**

**print(generated\_text)**

**2. Advanced Text Generation Using GPT (Hugging Face Transformers)**

**For more sophisticated models like GPT-2 or GPT-3, you can use the Hugging Face transformers library.**

**Step 1: Install necessary libraries**

**Python code**

**pip install transformers torch**

**Step 2: Text Generation using GPT-2**

**from transformers import GPT2LMHeadModel, GPT2Tokenizer**

**Python code**

**# Load the pre-trained GPT-2 model and tokenizer**

**model\_name = 'gpt2' # You can choose other variants like 'gpt2-medium', 'gpt2-large', etc.**

**model = GPT2LMHeadModel.from\_pretrained(model\_name)**

**tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)**

**# Encode input text to tokens**

**input\_text = "Once upon a time"**

**input\_ids = tokenizer.encode(input\_text, return\_tensors='pt')**

**# Generate text**

**output = model.generate(input\_ids, max\_length=200, num\_return\_sequences=1, no\_repeat\_ngram\_size=2, temperature=0.7)**

**# Decode and print the output text**

**generated\_text = tokenizer.decode(output[0], skip\_special\_tokens=True)**

**print(generated\_text)**

**Explanation:**

**LSTM-based model: The first approach involves training an LSTM-based model for character-level text generation. It works by predicting the next character in a sequence given a certain number of previous characters.**

**GPT-based model: The second approach leverages pre-trained transformer models (GPT-2 here) from Hugging Face, which is a more powerful and sophisticated method of text generation. It requires no training and can generate long, coherent text based on a prompt.**

**Which approach to choose:**

**LSTM: Use this if you have limited resources or if you want a custom solution (e.g., character-level generation).**

**GPT: Use this if you want state-of-the-art performance and ease of use with pre-trained models. It's great for generating high-quality text without training a model from scratch**