Interview Task: Generative AI with LSTM - Text Generation

Overview

In this task, you will develop a text generator using an LSTM (Long Short-Term Memory) model. The goal is to train the model on a dataset of text sequences and generate new, coherent text based on seed input. You will work through data preprocessing, model design, training, and text generation.

Dataset

Use a public domain text dataset such as Shakespeare's works, which can be found on Project Gutenberg or from other sources like Kaggle. The dataset should be in .txt format, containing a large body of text.

Example Dataset:

- - Shakespeare's Complete Works
- - Kaggle Text Datasets

Task Instructions

1. Dataset Loading and Preprocessing:

- - Download or load a text dataset (a large body of text, such as a book).
  - Convert the text to lowercase and remove punctuation.
  - Tokenize the text into sequences of words or characters.
  - Prepare input-output pairs where the input is a sequence of tokens, and the output is the next token in the sequence.

2. Model Design:

- - Use TensorFlow/Keras or PyTorch to build an LSTM-based text generation model.
  - Include an embedding layer, one or more LSTM layers, and a dense output layer with softmax activation for token prediction.
  - Compile the model with a suitable loss function (e.g., categorical crossentropy) and optimizer (e.g., Adam).

3. Model Training:

- - Split the dataset into training and validation sets.
  - Train the model on the input-output pairs.
  - Implement early stopping or checkpoints to prevent overfitting.

4. Text Generation:

- - Once the model is trained, generate new text by feeding it a seed sequence.
  - The seed sequence can be the first few words from the dataset.
  - Generate a sequence of new tokens by predicting the next token in the sequence iteratively.
  - Convert the generated token indices back to readable text.

Deliverables

1. **Code:**

- A Python script (or notebook) that includes:
  - Data preprocessing.
  - Model architecture and training process.
  - Text generation logic.

Ensure the script is well-documented, with clear comments explaining each part of the code.

2. **Generated Text Output:**

- Include sample outputs of the generated text based on a few different seed inputs.

3. **Bonus:**

- Experiment with different model architectures (e.g., deeper LSTM layers or different sequence lengths) and report how these affect the quality of the generated text.

4. **Dataset:**

- Include a link to the dataset you used, or provide instructions for downloading it.

Evaluation Criteria

- - Model Performance: How well the model generates coherent text based on seed input.
- - Code Quality: Readability, organization, and use of best practices in Python.
- - Creativity: Experimentation with different architectures or hyperparameters.
- - Problem-Solving: Approach to preprocessing the data and overcoming challenges in model training.