Lab 2 – Case Study

Aim: Autocomplete System for Email Composition

Problem Statement:

A software company wants to develop an intelligent autocomplete system for email composition. The goal is to assist users in generating coherent and contextually appropriate sentences while composing emails. The system should predict the next word or phrase based on the user's input and the context of the email.

Objectives:

The objectives of the provided program are to implement a simple email autocomplete system using the GPT-2 language model. The program aims to facilitate user interaction by suggesting autocompletions based on the context provided and the user's input. Key objectives include initializing and integrating the GPT-2 model and tokenizer from the Hugging Face Transformers library, defining a class structure (`EmailAutocompleteSystem`) to encapsulate the autocomplete system, and creating a method (`generate_suggestions`) to generate context-aware suggestions. The program encourages user engagement by incorporating a user input loop, allowing continuous interaction until the user chooses to exit. The ultimate goal is to demonstrate the practical use of a pre-trained language model for generating relevant suggestions in the context of email composition, showcasing the capabilities of the GPT-2 model for natural language processing tasks.

Approach:

1. **Data Collection:**
    - Collect a diverse dataset of emails, including different writing styles, topics, and formality levels.
    - Annotate the dataset with proper context information, such as sender, recipient, subject, and the body of the email.
2. **Data Preprocessing:**
    - Clean and tokenize the text data.
    - Handle issues like punctuation, capitalization, and special characters.
    - Split the dataset into training and testing sets.
3. **Model Selection:**
    - Choose a suitable NLP model for word generation. Options may include recurrent neural networks (RNNs), long short-term memory networks (LSTMs), or transformer models like GPT-3.

    - Fine-tune or train the model on the email dataset to understand the specific language patterns used in emails.

4. **Context Integration:**
    - Design a mechanism to incorporate contextual information from the email, such as the subject, previous sentences, and the relationship between the sender and recipient.
    - Implement a way for the model to understand the context shift within the email body.
5. **User Interface:**
    - Develop a user-friendly interface that integrates with popular email clients or standalone applications.
    - Allow users to enable or disable the autocomplete feature as needed.
    - Provide visual cues to indicate suggested words or phrases.
6. **Model Evaluation:**
    - Evaluate the model's performance on the test dataset using metrics like perplexity, accuracy, and precision.
    - Gather user feedback on the effectiveness and usability of the autocomplete system.
7. **Fine-Tuning and Iteration:**
    - Analyze user feedback and performance metrics to identify areas for improvement.
    - Consider refining the model based on user suggestions and addressing any limitations.
8. **Deployment:**
    - Deploy the trained model as a service that can be accessed by the email application.
    - Ensure scalability and reliability of the autocomplete system.

Potential Challenges:

- **Context Understanding:** Ensuring the model effectively understands and incorporates the context of the email.
- **Ambiguity Handling:** Dealing with ambiguous phrases and understanding the user's intended meaning.

- **Personalization:** Tailoring the system to individual writing styles and preferences.

Success Criteria:

- Improved email composition efficiency and speed.
- Positive user feedback on the accuracy and relevance of autocomplete suggestions.
- Reduction in typing errors and improved overall user experience.

By successfully developing and implementing this word generation program, the company aims to enhance the productivity and user experience of individuals engaged in email communication.

Program :

```
!pip install transformers

import torch

from transformers import GPT2LMHeadModel, GPT2Tokenizer


class EmailAutocompleteSystem:
    def __init__(self):
        self.model_name = "gpt2"
        self.tokenizer = GPT2Tokenizer.from_pretrained(self.model_name)
        self.model = GPT2LMHeadModel.from_pretrained(self.model_name)


    def generate_suggestions(self, user_input, context):
        input_text = f"{context} {user_input}"
        input_ids = self.tokenizer.encode(input_text, return_tensors="pt")


        with torch.no_grad():
            output = self.model.generate(input_ids, max_length=50, num_return_sequences=1,
no_repeat_ngram_size=2)


        generated_text = self.tokenizer.decode(output[0], skip_special_tokens=True)
        suggestions = generated_text.split()[len(user_input.split()):]
        return suggestions

# Example usage
if __name__ == "__main__":
    autocomplete_system = EmailAutocompleteSystem()

    # Assume user is composing an email with some context
    email_context = "Subject: Discussing Project Proposal\nHi [Recipient],"

    while True:
        user_input = input("Enter your sentence (type 'exit' to end): ")

        if user_input.lower() == 'exit':
            break

        suggestions = autocomplete_system.generate_suggestions(user_input, email_context)

        if suggestions:
            print("Autocomplete Suggestions:", suggestions)
        else:
            print("No suggestions available.")
```

Output:

```
Enter your sentence (type 'exit' to end): hello, how are you ? How's
everything going on !
The attention mask and the pad token id were not set. As a consequence, you
may observe unexpected behavior. Please pass your input's `attention_mask` to
obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
Autocomplete Suggestions: ["How's", 'everything', 'going', 'on!', "I'm", 'a',
'programmer', 'and', "I've", 'been', 'working', 'on', 'a', 'project', 'for',
'a', 'while', 'now.', 'I', 'have', 'a', 'lot', 'of', 'ideas', 'for', 'the']
Enter your sentence (type 'exit' to end): exit
```

Result:

The result demonstrates the integration of a powerful language model for enhancing user experience in composing emails through intelligent autocomplete suggestions.