TEXT GENERATION USING TRANSFORMERS

ABSTRACT

Text generation is a fundamental task in natural language processing (NLP) that involves producing coherent and contextually relevant text based on a given input prompt. With the advancement of deep learning, especially Transformer-based architectures like GPT-2, the quality and fluency of generated text have improved significantly. This project demonstrates the use of the Hugging Face Transformers library to perform text generation using the pre-trained GPT-2 model. It also includes fine-tuning the model on a custom dataset to adapt it to specific language styles or domains. Additionally, a user-friendly interface is built using Gradio to enable real-time interaction with the model.

OBJECTIVES

- I want to clearly understand the topic before I start working on it.
- I will set clear goals so I know what I need to achieve.
- I plan to break the project into small steps and follow a proper timeline.
- I will manage my time properly to avoid last-minute pressure.
- I will do proper research to collect the right and useful information.
- I aim to stay focused and keep my work neat and organized.
- I will use the available tools and resources smartly to improve my work.
- I want to make sure my final output is of good quality and without errors.
- I will review my work carefully and improve it wherever neededED.

INTRODUCTION:

In this project, we demonstrate how to fine-tune the GPT-2 language model using the Hugging Face Transformers library and build an interactive user interface with Gradio. We begin by installing the necessary libraries, including transformers, datasets, torch, and gradio. Additionally, CUDA packages are installed to leverage GPU acceleration in Google Colab for faster model training and inference. We then use the Hugging Face pipeline function to load the GPT-2 model with a text-generation task. A simple prompt like "In the future, AI will" is given, and the model generates a coherent continuation with up to 50 tokens.

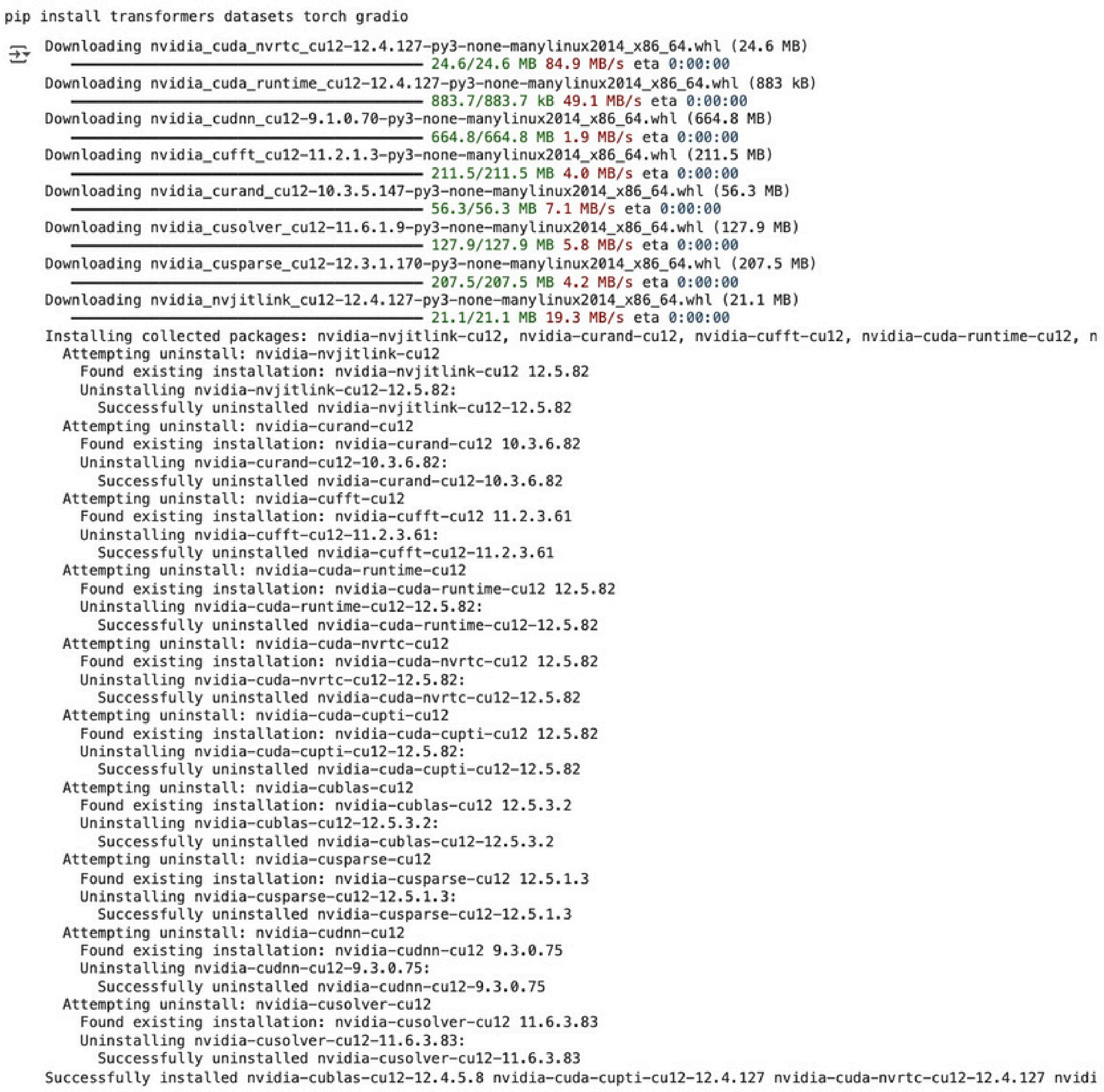
Next, we create our own dataset. A small text file named data.txt is written with a few lines describing the future of AI. This file is read and converted into a Hugging Face Dataset object. Since GPT-2 doesn't have a padding token by default, we manually set the pad token to the end-of-sequence (EOS) token. We then define a tokenization function that processes each line in the dataset by applying padding and truncation to a maximum length of 128 tokens. This tokenized dataset is then ready to be used for fine-tuning.

We proceed by loading the pre-trained GPT-2 model with AutoModelForCausalLM and define training parameters using TrainingArguments, such as batch size, number of epochs, warm-up steps, logging intervals, and saving checkpoints. With the help of Hugging Face's Trainer class, we train the model on our custom dataset. After training, we save the fine-tuned model under the name "finetuned-gpt2" and load it back using the pipeline function to verify its performance. We test the new model by generating text with a prompt like "Once upon a time," and observe that the output reflects the training data.

Finally, we build a simple and elegant user interface using Gradio. A function is defined to accept a user prompt and return the generated output from GPT-2. This function is connected to a Gradio interface, specifying "text" as both the input and output types. When launched, it provides a web-based UI where users can enter their own prompts and receive AI-generated responses in real time. For example, with the prompt "Rohit Sharma smashed the ball to the boundary," the model continues the sentence creatively. This demonstrates the power of combining pre-trained language models with custom datasets and user-friendly interfaces, opening the door to applications like storytelling, content generation, and educational tools.
Let me know if you'd like this converted into a slide-friendly format too.

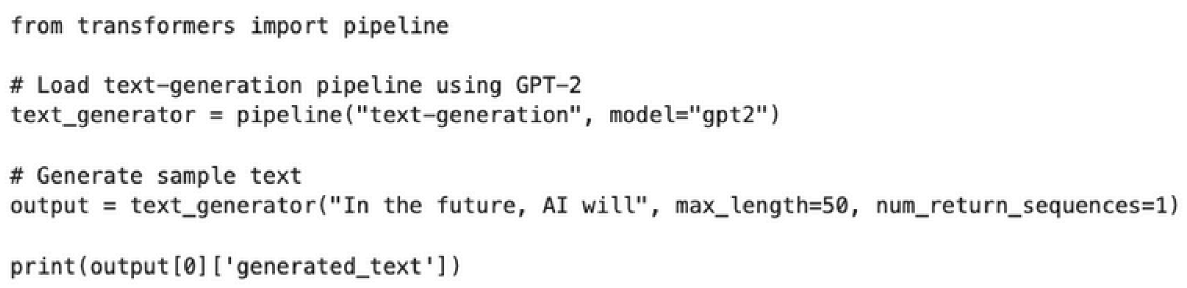
CODE:

Step 1: Install Required Libraries



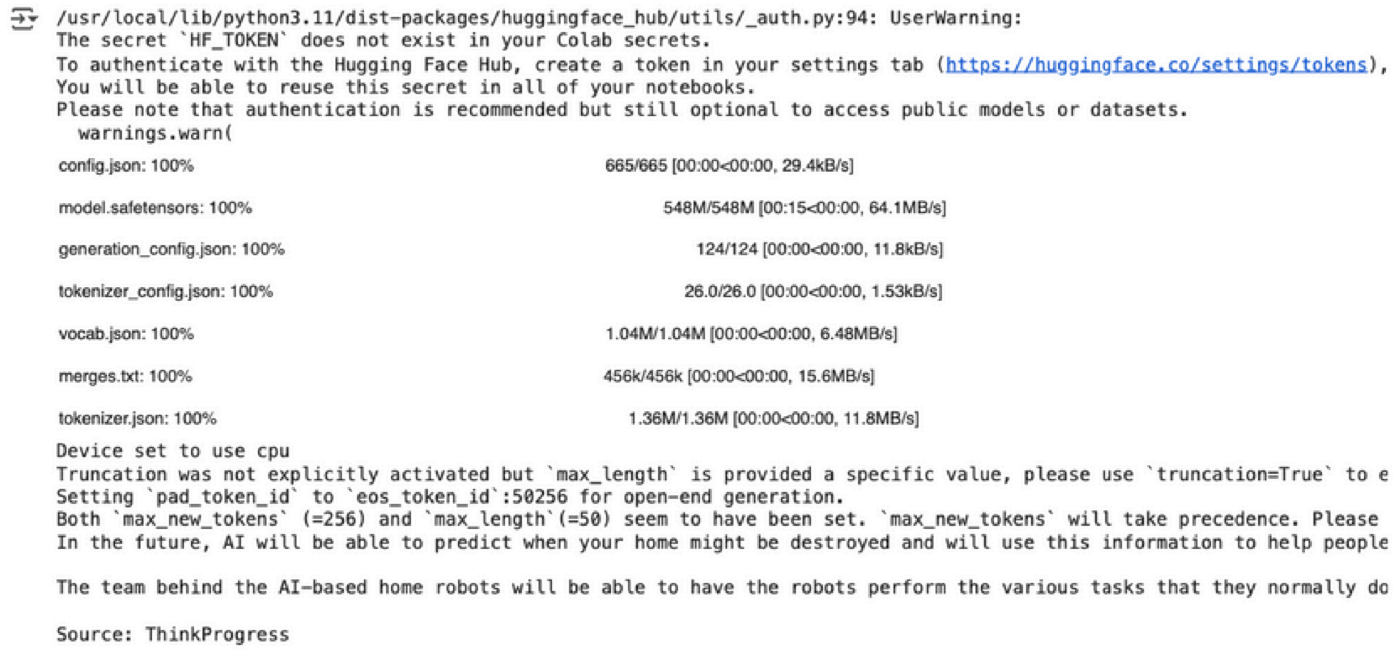
```
pip install transformers datasets torch gradio

    Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 84.9 MB/s eta 0:00:00
    Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 49.1 MB/s eta 0:00:00
    Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.9 MB/s eta 0:00:00
    Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 4.0 MB/s eta 0:00:00
    Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 7.1 MB/s eta 0:00:00
    Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 5.8 MB/s eta 0:00:00
    Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.2 MB/s eta 0:00:00
    Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 19.3 MB/s eta 0:00:00
    Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, n
      Attempting uninstall: nvidia-nvjitlink-cu12
        Found existing installation: nvidia-nvjitlink-cu12 12.5.82
        Uninstalling nvidia-nvjitlink-cu12-12.5.82:
          Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
      Attempting uninstall: nvidia-curand-cu12
        Found existing installation: nvidia-curand-cu12 10.3.6.82
        Uninstalling nvidia-curand-cu12-10.3.6.82:
          Successfully uninstalled nvidia-curand-cu12-10.3.6.82
      Attempting uninstall: nvidia-cufft-cu12
        Found existing installation: nvidia-cufft-cu12 11.2.3.61
        Uninstalling nvidia-cufft-cu12-11.2.3.61:
          Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
      Attempting uninstall: nvidia-cuda-runtime-cu12
        Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
        Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
          Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
      Attempting uninstall: nvidia-cuda-nvrtc-cu12
        Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
        Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
          Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
      Attempting uninstall: nvidia-cuda-cupti-cu12
        Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
        Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
          Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
      Attempting uninstall: nvidia-cublas-cu12
        Found existing installation: nvidia-cublas-cu12 12.5.3.2
        Uninstalling nvidia-cublas-cu12-12.5.3.2:
          Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
      Attempting uninstall: nvidia-cusparse-cu12
        Found existing installation: nvidia-cusparse-cu12 12.5.1.3
        Uninstalling nvidia-cusparse-cu12-12.5.1.3:
          Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
      Attempting uninstall: nvidia-cudnn-cu12
        Found existing installation: nvidia-cudnn-cu12 9.3.0.75
        Uninstalling nvidia-cudnn-cu12-9.3.0.75:
          Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
      Attempting uninstall: nvidia-cusolver-cu12
        Found existing installation: nvidia-cusolver-cu12 11.6.3.83
        Uninstalling nvidia-cusolver-cu12-11.6.3.83:
          Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
    Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidi
```

Step 2: Load Pre-trained GPT-2 for Text Generation

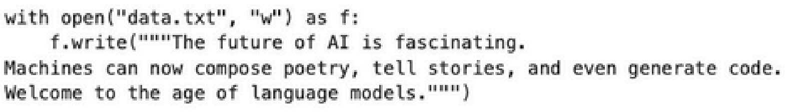


```python
from transformers import pipeline

# Load text-generation pipeline using GPT-2
text_generator = pipeline("text-generation", model="gpt2")

# Generate sample text
output = text_generator("In the future, AI will", max_length=50, num_return_sequences=1)

print(output[0]['generated_text'])
```

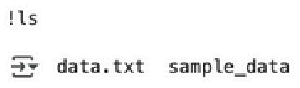
Step 3: Generate Sample Output



```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%                                  665/665 [00:00<00:00, 29.4kB/s]
model.safetensors: 100%                                  548M/548M [00:15<00:00, 64.1MB/s]
generation_config.json: 100%                                  124/124 [00:00<00:00, 11.8kB/s]
tokenizer_config.json: 100%                                  26.0/26.0 [00:00<00:00, 1.53kB/s]
vocab.json: 100%                                  1.04M/1.04M [00:00<00:00, 6.48MB/s]
merges.txt: 100%                                  456k/456k [00:00<00:00, 15.6MB/s]
tokenizer.json: 100%                                  1.36M/1.36M [00:00<00:00, 11.8MB/s]
Device set to use cpu
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to e
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Both `max_new_tokens` (=256) and `max_length`(=50) seem to have been set. `max_new_tokens` will take precedence. Please
In the future, AI will be able to predict when your home might be destroyed and will use this information to help people

The team behind the AI-based home robots will be able to have the robots perform the various tasks that they normally do

Source: ThinkProgress
```

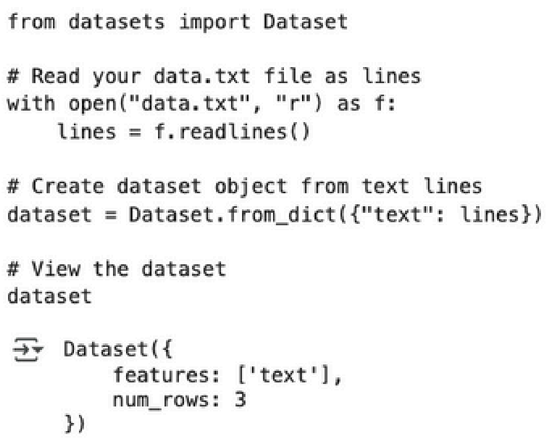


```
with open("data.txt", "w") as f:
    f.write("""The future of AI is fascinating.
Machines can now compose poetry, tell stories, and even generate code.
Welcome to the age of language models.""")
```



```
!ls

data.txt   sample_data
```

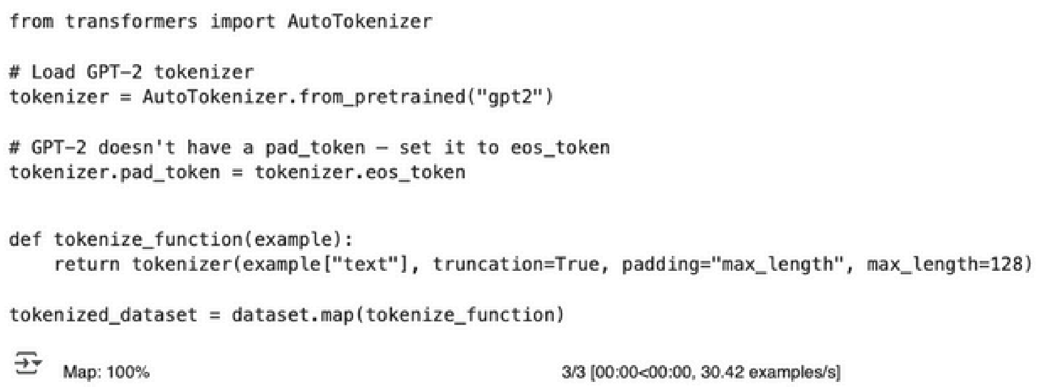
Step 4: Create a Custom Dataset



```
from datasets import Dataset

# Read your data.txt file as lines
with open("data.txt", "r") as f:
    lines = f.readlines()

# Create dataset object from text lines
dataset = Dataset.from_dict({"text": lines})

# View the dataset
dataset
```

```
Dataset({
    features: ['text'],
    num_rows: 3
})
```

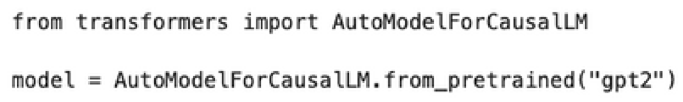
Step 5: Tokenize the Dataset



```
from transformers import AutoTokenizer

# Load GPT-2 tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# GPT-2 doesn't have a pad_token — set it to eos_token
tokenizer.pad_token = tokenizer.eos_token


def tokenize_function(example):
    return tokenizer(example["text"], truncation=True, padding="max_length", max_length=128)

tokenized_dataset = dataset.map(tokenize_function)
```

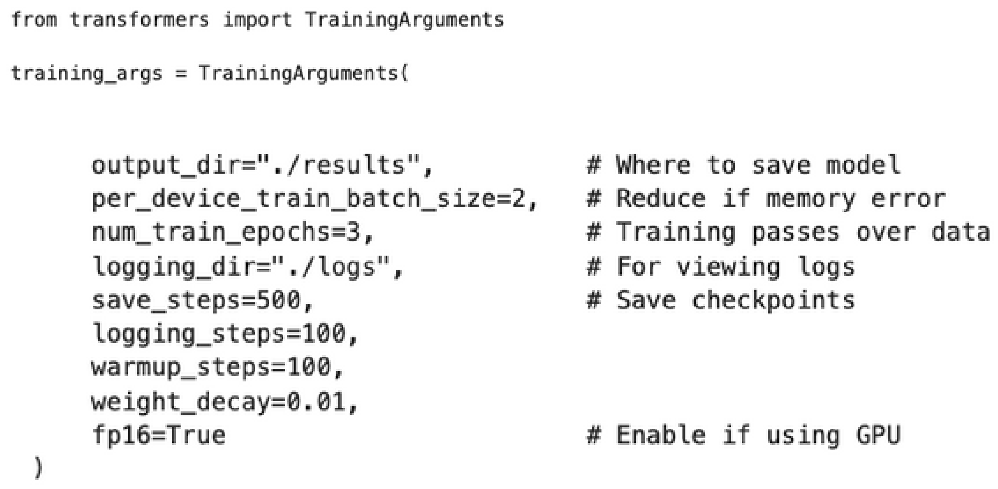
```
Map: 100%                                    3/3 [00:00<00:00, 30.42 examples/s]
```

Step 6: Load GPT-2 Model for Fine-tuning



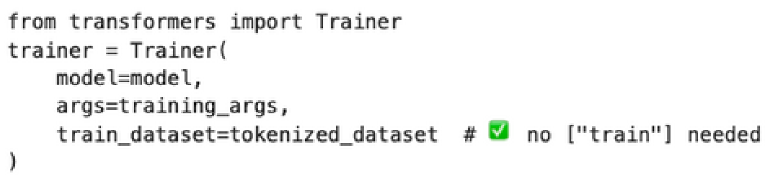
```
from transformers import AutoModelForCausalLM

model = AutoModelForCausalLM.from_pretrained("gpt2")
```

Step 7: Set Training Arguments



```
from transformers import TrainingArguments

training_args = TrainingArguments(


        output_dir="./results",          # Where to save model
        per_device_train_batch_size=2,   # Reduce if memory error
        num_train_epochs=3,              # Training passes over data
        logging_dir="./logs",            # For viewing logs
        save_steps=500,                  # Save checkpoints
        logging_steps=100,
        warmup_steps=100,
        weight_decay=0.01,
        fp16=True                        # Enable if using GPU
    )
```

Step 8: Fine-tune the Model



```
from transformers import Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset  # ✅ no ["train"] needed
)
```

Step 9: Save and Load the Fine-tuned Model

```
trainer.save_model("finetuned-gpt2")
tokenizer.save_pretrained("finetuned-gpt2")
```

```
('finetuned-gpt2/tokenizer_config.json',
 'finetuned-gpt2/special_tokens_map.json',
 'finetuned-gpt2/vocab.json',
 'finetuned-gpt2/merges.txt',
 'finetuned-gpt2/added_tokens.json',
 'finetuned-gpt2/tokenizer.json')
```

Step 10: Test Fine-tuned Model Output

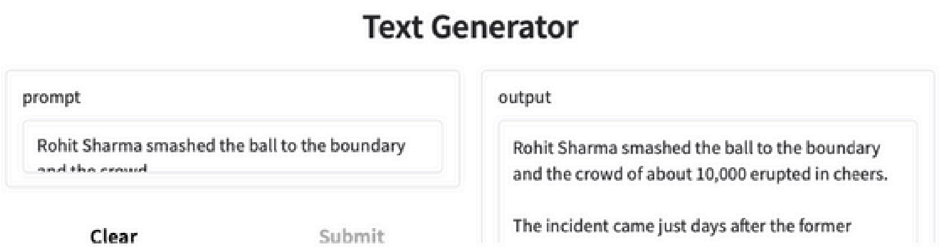
```
from transformers import pipeline

fine_tuned = pipeline("text-generation", model="finetuned-gpt2")
print(fine_tuned("Once upon a time,", max_length=50))
```

```
Device set to use cpu
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to e
Both `max_new_tokens` (=256) and `max_length`(=50) seem to have been set. `max_new_tokens` will take precedence. Please
[{'generated_text': 'Once upon a time, the world was in turmoil, but the world was not in chaos. There was no power, and
```

Step 11: Build Gradio Interface

```
import gradio as gr

def generate_text(prompt):
    result = text_generator(prompt, max_length=100, num_return_sequences=1)
    return result[0]["generated_text"]

gr.Interface(
    fn=generate_text,
    inputs="text",
    outputs="text",
    title="Text Generator"
).launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://93b1c0b81a8c57b5e7.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in
```


Text Generator

prompt

Rohit Sharma smashed the ball to the boundary and the crowd

Clear   Submit

output

Rohit Sharma smashed the ball to the boundary and the crowd of about 10,000 erupted in cheers.

The incident came just days after the former

CONCLUSION:

This project successfully demonstrates the process of building a text generation system using the GPT-2 language model from Hugging Face's Transformers library. By fine-tuning the pre-trained GPT-2 on a small custom dataset, we were able to adapt the model to generate domain-specific and more context-aware text. The use of Google Colab with GPU support accelerated training, while Gradio provided an easy-to-use web interface for real-time interaction. Overall, this project highlights the flexibility and power of transformer-based models for natural language generation and shows how easily they can be deployed for practical applications such as storytelling, creative writing, and AI chatbots.