Train a model to generate coherent and contextually relevant text based on a given prompt.

Install Required Libraries

```
pip install transformers datasets torch pandas
```

```
Requirement already satisfied: transformers in
/usr/local/lib/python3.10/dist-packages (4.44.2)
Requirement already satisfied: datasets in
/usr/local/lib/python3.10/dist-packages (3.0.1)
Requirement already satisfied: torch in
/usr/local/lib/python3.10/dist-packages (2.4.1+cu121)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.24.7)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers)
(2024.9.11)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.66.5)
Requirement already satisfied: pyarrow>=15.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (16.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: xxhash in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.6.1,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from
fsspec[http]<=2024.6.1,>=2023.1.0->datasets) (2024.6.1)
Requirement already satisfied: aiohttp in
```

```
/usr/local/lib/python3.10/dist-packages (from datasets) (3.10.10)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in
/usr/local/lib/python3.10/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in
/usr/local/lib/python3.10/dist-packages (from torch) (3.4)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(2.4.3)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.3.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(6.1.0)
Requirement already satisfied: yarl<2.0,>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.14.0)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(4.0.3)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(3.4.0)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
```

```
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2024.8.30)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch) (3.0.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from yarl<2.0,>=1.12.0-
>aiohttp->datasets) (0.2.0)
```

Loading and Inspecting the Dataset

```
import pandas as pd

# Load the dataset
data = pd.read_csv('/content/annotated_coding_data.csv')

# Display the dataset
print(data.head())
```

```
                                        clinical_note   icd10_codes  \
0  Patient is a 25-year-old female diagnosed with...  J45.40;J30.2
1  Patient is a Child diagnosed with asthma. Admi...        J45.40
2  Patient is a 45-year-old male diagnosed with h...           I10
3  Patient is a 30-year-old female diagnosed with...        J45.40
4  Patient is a 30-year-old female diagnosed with...       M23.221

           cpt_codes
0  29881;D0120;D1110
1              99213
2              90832
3       D0120;D1110
4              97110
```

Concatenating Codes for Generation

```
# Convert 'icd10_codes' and 'cpt_codes' to strings
data['icd10_codes'] = data['icd10_codes'].astype(str)
data['cpt_codes'] = data['cpt_codes'].astype(str)

# Combine ICD-10 and CPT codes into a single string
data['codes'] = data['icd10_codes'] + ' ' + data['cpt_codes']

# Create the prompt by appending the codes to the clinical note
data['prompt'] = data['clinical_note'] + '###' + ' ' + data['codes']

# Display the updated dataset
print(data[['prompt']].head())
```

```
                                                prompt
0  Patient is a 25-year-old female diagnosed with...
1  Patient is a Child diagnosed with asthma. Admi...
2  Patient is a 45-year-old male diagnosed with h...
3  Patient is a 30-year-old female diagnosed with...
4  Patient is a 30-year-old female diagnosed with...
```

Saving the Preprocessed Data

```
import json

# Function to convert a row to JSON
def row_to_json(row):
    return {
        "prompt": row['prompt']
    }

# Convert the DataFrame to JSONL
with open('preprocessed_coding_data.jsonl', 'w') as f:
    for _, row in data.iterrows():
        json.dump(row_to_json(row), f)
        f.write('\n')
```

Fine-Tuning GPT-2

We'll use Hugging Face's transformers library to fine-tune GPT-2 on our sample dataset.

```
#Loading the Dataset
from datasets import load_dataset

# Load the dataset from the JSONL file
dataset = load_dataset('json',
data_files='preprocessed_coding_data.jsonl', split='train')

# Inspect the dataset
print(dataset)
print(dataset[0])
```


```
{"model_id":"5969483c1086458787d0849450c8a1b3","version_major":2,"vers
ion_minor":0}
```


```
Dataset({
    features: ['prompt'],
    num_rows: 100
})
{'prompt': 'Patient is a 25-year-old female diagnosed with asthma.
Performed knee arthroscopy. Scheduled follow-up appointment.###
J45.40;J30.2 29881;D0120;D1110'}
```

Initializing the Tokenizer and Model

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load the pre-trained GPT-2 tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Since GPT-2 doesn't have a pad token, set it to eos_token
tokenizer.pad_token = tokenizer.eos_token
model.config.pad_token_id = tokenizer.eos_token_id
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_token.py:89: 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), set it as
secret in your Google Colab and restart your session.
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(
```


```
{"model_id":"b479eead444b4a91af1f7cc166649ca7","version_major":2,"vers
ion_minor":0}

{"model_id":"173a78aa693e45f1995adc022e518587","version_major":2,"vers
ion_minor":0}

{"model_id":"c29f5bdc40e84cd8a37a3fe3cb5e63fb","version_major":2,"vers
ion_minor":0}

{"model_id":"b89cffb6fab84008a5499ec4752f3702","version_major":2,"vers
ion_minor":0}

{"model_id":"874594728db34ad08f07ad8dfe42acab","version_major":2,"vers
ion_minor":0}
```


```
/usr/local/lib/python3.10/dist-packages/transformers/
tokenization_utils_base.py:1601: FutureWarning:
`clean_up_tokenization_spaces` was not set. It will be set to `True`
by default. This behavior will be depracted in transformers v4.45, and
will be then set to `False` by default. For more details check this
issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
```


```
{"model_id":"6f980c002ca94d3089c7f5426af6131c","version_major":2,"vers
ion_minor":0}

{"model_id":"e4218e4511d54582b7682dd7bee8f650","version_major":2,"vers
ion_minor":0}
```

Tokenizing the Dataset

```
def tokenize_function(examples):
    # Tokenize the prompt
    tokenized = tokenizer(
        examples['prompt'],
        padding='max_length',
        truncation=True,
        max_length=512
    )
    # Set labels equal to input_ids for language modeling
    tokenized['labels'] = tokenized['input_ids'].copy()
    return tokenized

# Apply the tokenization to the dataset
tokenized_dataset = dataset.map(tokenize_function, batched=True)

# Remove the original 'prompt' column to save memory
tokenized_dataset = tokenized_dataset.remove_columns(['prompt'])

# Set the format for PyTorch, including 'labels'
tokenized_dataset.set_format(type='torch', columns=['input_ids',
'attention_mask', 'labels'])
```

{"model_id":"e4d79bb8fbe541789015f87b695cdb1d","version_major":2,"version_minor":0}

Setting Up Training Arguments

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="./gpt2_medbilling",
    overwrite_output_dir=True,
    num_train_epochs=3,
    per_device_train_batch_size=1,  # Adjust based on your GPU
    save_steps=10,
    save_total_limit=2,
    logging_steps=5,
    prediction_loss_only=True,
)
```

Initializing the Trainer

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
)
```

Fine-Tuning the Model

```
# Start the training
trainer.train()
```

```
<IPython.core.display.HTML object>

TrainOutput(global_step=300, training_loss=0.1338209815820058,
metrics={'train_runtime': 3680.7419, 'train_samples_per_second':
0.082, 'train_steps_per_second': 0.082, 'total_flos':
78387609600000.0, 'train_loss': 0.1338209815820058, 'epoch': 3.0})
```

```
#Save the model and tokenizer
model.save_pretrained("./gpt2_medbilling")
tokenizer.save_pretrained("./gpt2_medbilling")
```

```
('./gpt2_medbilling/tokenizer_config.json',
 './gpt2_medbilling/special_tokens_map.json',
 './gpt2_medbilling/vocab.json',
 './gpt2_medbilling/merges.txt',
 './gpt2_medbilling/added_tokens.json')
```

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load the fine-tuned model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("./gpt2_medbilling")
model = GPT2LMHeadModel.from_pretrained("./gpt2_medbilling")
model.config.pad_token_id = tokenizer.eos_token_id

import torch

def generate_codes(clinical_note, max_length=50):
    # Create the prompt
    prompt = clinical_note + "###"

    # Encode the input
    inputs = tokenizer.encode(prompt, return_tensors='pt')

    # Generate output
    outputs = model.generate(
        inputs,
        max_length=inputs.shape[1] + max_length,
        num_return_sequences=1,
        no_repeat_ngram_size=2,
        early_stopping=True
    )

    # Decode the output
    generated_text = tokenizer.decode(outputs[0],
skip_special_tokens=True)
```

```
    # Extract the codes
    codes = generated_text.split("###")[-1].strip()

    return codes

# Example clinical note
new_clinical_note = "Patient is a 45-year-old male diagnosed with
asthma. Administered nasal corticosteroids."

# Generate codes
suggested_codes = generate_codes(new_clinical_note)

print("Suggested Codes:", suggested_codes)
```

```
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.
```

```
Suggested Codes: J45.9 99195
```

```
# Load the fine-tuned model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("./gpt2_medbilling")
model = GPT2LMHeadModel.from_pretrained("./gpt2_medbilling")

# Ensure the model is in evaluation mode
model.eval()
```

```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2SdpaAttention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
```

```
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=768, out_features=50257, bias=False)
)

# Example clinical note
new_clinical_note = "Patient is a 45-year-old male diagnosed with
allergic rhinitis. Initiated insulin therapy."

# Tokenize the input
inputs = tokenizer(new_clinical_note, return_tensors='pt',
padding=True, truncation=True)

# Generate codes with adjusted parameters
with torch.no_grad():
    outputs = model.generate(
        **inputs,
        max_length=50,            # Maximum length of generated codes
        num_return_sequences=1,   # Number of sequences to return
        pad_token_id=tokenizer.eos_token_id,  # Padding token ID
        temperature=0.5,          # Adjust the temperature for
randomness
        top_k=50,                 # Only consider the top k tokens
        top_p=0.95                # Nucleus sampling
    )

# Decode the output
generated_codes = tokenizer.decode(outputs[0],
skip_special_tokens=True)

print("Suggested Codes:", generated_codes)

Suggested Codes: Patient is a 45-year-old male diagnosed with allergic
rhinitis. Initiated insulin therapy.### J18.9 99214
```