```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Wed Sep 27 17:14:27 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   31C    P0    45W / 400W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

AI/ML Coding Round - Data Preparation

Problem:

Train a LLM that can answer queries about JFrog Pipelines' native steps. When posed with a question like "How do I upload an artifact?" or "What step should I use for an Xray scan?", the model should list the appropriate native step(s) and provide an associated YAML for that step.

Requirements

1. Data Collection: Acquire publicly available information on Native Steps from JFrog's website that contain information on native steps for building pipelines. Data that is not publicly accessible falls outside the scope of this coding challenge. (https://jfrog.com/help/r/jfrog-pipelines-documentation/pipelines-steps)
2. Data Preprocessing: Process the text to make it suitable for training. This might involve tokenization, stemming, and other NLP techniques.
3. Model Training: Train a LLM on the (preprocessed) dataset. You can choose one of the freely available open source model like BERT or any other model available
4. Query Handling: Implement a function that takes a user query as input and returns the appropriate native step(s) and a sample YAML configuration.
5. YAML Generation: Implement a function that can generate a sample YAML configuration based on the identified native step(s).

1.1 Importing Libraries and data in training format

```
!pip install -q  torch peft==0.4.0 bitsandbytes==0.40.2 trl==0.4.7 accelerate sentencepiece
!pip install -q git+https://github.com/huggingface/transformers.git@main accelerate

import transformers
from transformers import AutoModelForCausalLM,AutoTokenizer
from transformers import LlamaForCausalLM, LlamaTokenizer
import torch
from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    TrainingArguments,
    pipeline
)
from peft import LoraConfig # for Parameter effecient finetuning
from trl import SFTTrainer # for supervised fine tuning

# for Loding the dataset
import pyarrow as pa
import pyarrow.dataset as ds
import pandas as pd
from datasets import Dataset
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.9/72.9 kB 1.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 92.5/92.5 MB 20.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.4/77.4 kB 10.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 258.1/258.1 kB 29.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 70.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 111.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 79.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 519.6/519.6 kB 41.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 295.0/295.0 kB 33.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 110.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 14.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 23.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 17.8 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 42.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 29.9 MB/s eta 0:00:00
  Building wheel for transformers (pyproject.toml) ... done
```

1.2 Convert the Dataset to traininng HF training format

- we need to connvert pandas dataframe to `hf-dataset(arrow_dataset)` to train the huffging face models

```
# Training data converstion Experiment - 1
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Model Training/final_data_for_training.csv')
dataset = ds.dataset(pa.Table.from_pandas(df).to_batches())

### convert to Huggingface dataset
training_data = Dataset(pa.Table.from_pandas(df))


type(training_data)
```

```
datasets.arrow_dataset.Dataset
```

```
# Model 1
base_model_name = "codellama/CodeLlama-7b-Instruct-hf" # base huggingface model for finetune
refined_model = "CodeLlama-7b-Instruct-jForg-enhanced" # the model name we are going to give for our finetuned model
```

1.3 Model Loding and llama tokinezer

```
# Loding the previous chcekpoints

! cp -r '/content/drive/MyDrive/Colab Notebooks/Model Training/results_modified' /content/
! cp -r '/content/drive/MyDrive/Colab Notebooks/Model Training/CodeLlama-7b-Instruct-jForg-enhanced' /content/



#  Tokenizer
llama_tokenizer = AutoTokenizer.from_pretrained("codellama/CodeLlama-7b-hf", trust_remote_code=True)
llama_tokenizer.pad_token = llama_tokenizer.eos_token
llama_tokenizer.padding_side = "right"  # Fix for fp16

# Quantization Config
quant_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=False
)

# Model
base_model = AutoModelForCausalLM.from_pretrained(
    base_model_name,
    quantization_config=quant_config,
    device_map= 'auto'
)
base_model.config.use_cache = False
base_model.config.pretraining_tp = 1
```

```
Downloading (…)okenizer_config.json: 100%  749/749 [00:00<00:00, 65.7kB/s]
Downloading tokenizer.model: 100%  500k/500k [00:00<00:00, 14.2MB/s]
Downloading (…)/main/tokenizer.json: 100%  1.84M/1.84M [00:00<00:00, 7.47MB/s]
Downloading (…)cial_tokens_map.json: 100%  411/411 [00:00<00:00, 33.7kB/s]
Loading the tokenizer from the `special_tokens_map.json` and the `added_tokens.json` will be removed in `transformers 5`,
Downloading (…)lve/main/config.json: 100%  646/646 [00:00<00:00, 54.3kB/s]
Downloading (…)fetensors.index.json: 100%  25.1k/25.1k [00:00<00:00, 1.48MB/s]
Downloading shards: 100%  2/2 [00:47<00:00, 21.73s/it]
Downloading (…)of-00002.safetensors: 100%  9.98G/9.98G [00:34<00:00, 272MB/s]
Downloading (…)of-00002.safetensors: 100%  3.50G/3.50G [00:12<00:00, 249MB/s]
Loading checkpoint shards: 100%  2/2 [00:10<00:00, 4.80s/it]
Downloading (…)neration_config.json: 100%  116/116 [00:00<00:00, 9.79kB/s]
```

Double-click (or enter) to edit

```
# LoRA Config
peft_parameters = LoraConfig(
    lora_alpha=16,
    lora_dropout=0.1,
    r=8,
    bias="none",
    task_type="CAUSAL_LM"
)

# Training Params
train_params = TrainingArguments(
    output_dir="./results_modified",
    num_train_epochs=1,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=1,
    optim="paged_adamw_32bit",
    save_steps=25,
    logging_steps=25,
    learning_rate=2e-4,
    weight_decay=0.001,
    fp16=False,
    bf16=False,
    max_grad_norm=0.3,
    max_steps=-1,
    warmup_ratio=0.03,
    group_by_length=True,
    lr_scheduler_type="constant",
    report_to="tensorboard"
)

# Trainer
fine_tuning = SFTTrainer(
    model=base_model,
    train_dataset=training_data,
    peft_config=peft_parameters,
    dataset_text_field="PiplineProcess",
    tokenizer=llama_tokenizer,
    args=train_params
)

# Training
fine_tuning.train()
# Save Model
fine_tuning.model.save_pretrained(refined_model)
```

```
/usr/local/lib/python3.10/dist-packages/peft/utils/other.py:102: FutureWarning: prepare_model_for_int8_training is deprec
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:159: UserWarning: You didn't pass a `max_seq_length` a
  warnings.warn(
```

```
Map: 100%  244/244 [00:00<00:00, 377.41 examples/s]
```

```
You are using 8-bit optimizers with a version of `bitsandbytes` < 0.41.1. It is recommended to update your version as a m
You're using a CodeLlamaTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is f
```



```
[10/10 00:33, Epoch 0/1]
```

```
# Saving the chcekpoints to drive

! cp -r /content/results_modified/ '/content/drive/MyDrive/Colab Notebooks/Model Training'
! cp -r /content/CodeLlama-7b-Instruct-jForg-enhanced/ '/content/drive/MyDrive/Colab Notebooks/Model Training'
```

```
# Generate Text
query = "what is the YAMl for jfrog docker push"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')
output = text_gen(f"<s>[INST] {query} [/INST]",
                  do_sample=True,
                  top_k=10,
                  top_p = 0.9,
                  temperature = 0.2,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=200) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%  2/2 [00:04<00:00, 2.04s/it]
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
<s>[INST] what is the YAMl for jfrog docker push [/INST]  The YAML file for JFrog Docker push is used to define the confi
```

```
version: 1

jobs:
  - name: docker-push
    docker:
      - image: jfrog/docker-client
      - image: jfrog/docker-client:latest
    steps:
      - name: docker-push
        command: docker push
        args:
          - image: my-image
          - tag: my-tag
          - registry: my-registry
          - username: my-username
          - password: my-password
```

```
This YAML file defines a job called "docker-push" that uses the JFrog Docker client image
```

```
# Generate Text
system_message = "<<SYS>>You are a helpful, respectful and honest assistant. Always answers only users jFrog pipline related q
query = "Write a pipeline to do a Docker Build & Publish?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]{system_message} {query} [/INST]",
                  do_sample=True,
                  top_k=10,
                  top_p = 0.9,
                  temperature = 0.2,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=200) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%  2/2 [00:03<00:00, 1.72s/it]
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Always answers only users jFrog pipline related quest
```

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'docker build -t my-image.'
            }
        }
        stage('Publish') {
            steps {
                sh 'docker push my-image'
            }
        }
    }
}
```

Experiment 1 failed we have adjusted the data and will train on new data with mutiple epochs

```
# Training data converstion Experiment - 2 - new_final_data
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Model Training/new_final_data_for_training.csv')
dataset = ds.dataset(pa.Table.from_pandas(df).to_batches())

### convert to Huggingface dataset
training_data = Dataset(pa.Table.from_pandas(df))
type(training_data)
```

```
datasets.arrow_dataset.Dataset
```

```
#  Tokenizer
llama_tokenizer = AutoTokenizer.from_pretrained("codellama/CodeLlama-7b-hf", trust_remote_code=True)
llama_tokenizer.pad_token = llama_tokenizer.eos_token
llama_tokenizer.padding_side = "right"  # Fix for fp16

# Quantization Config
quant_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=False
)

# Model
base_model = AutoModelForCausalLM.from_pretrained(
    base_model_name,
    quantization_config=quant_config,
    device_map= 'auto'
)
base_model.config.use_cache = False
base_model.config.pretraining_tp = 1
```

```
Downloading (…)okenizer_config.json: 100%  749/749 [00:00<00:00, 57.1kB/s]
Downloading tokenizer.model: 100%  500k/500k [00:00<00:00, 15.3MB/s]
Downloading (…)/main/tokenizer.json: 100%  1.84M/1.84M [00:00<00:00, 9.88MB/s]
Downloading (…)cial_tokens_map.json: 100%  411/411 [00:00<00:00, 33.6kB/s]
Loading the tokenizer from the `special_tokens_map.json` and the `added_tokens.json` will be removed in `transformers 5`,
Downloading (…)lve/main/config.json: 100%  646/646 [00:00<00:00, 48.0kB/s]
Downloading (…)fetensors.index.json: 100%  25.1k/25.1k [00:00<00:00, 1.24MB/s]
Downloading shards: 100%  2/2 [00:39<00:00, 18.03s/it]
Downloading (…)of-00002.safetensors: 100%  9.98G/9.98G [00:28<00:00, 371MB/s]
Downloading (…)of-00002.safetensors: 100%  3.50G/3.50G [00:10<00:00, 354MB/s]
Loading checkpoint shards: 100%  2/2 [00:10<00:00, 4.65s/it]
Downloading (…)neration_config.json: 100%  116/116 [00:00<00:00, 9.60kB/s]
```

```
# LoRA Config
peft_parameters = LoraConfig(
    lora_alpha=16,
```

```
    lora_dropout=0.1,
    r=8,
    bias="none",
    task_type="CAUSAL_LM"
)

# Training Params
train_params = TrainingArguments(
    output_dir="./results_modified",
    num_train_epochs=1,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=1,
    optim="paged_adamw_32bit",
    save_steps=25,
    logging_steps=25,
    learning_rate=2e-4,
    weight_decay=0.001,
    fp16=False,
    bf16=False,
    max_grad_norm=0.3,
    max_steps=250,
    warmup_ratio=0.03,
    group_by_length=True,
    lr_scheduler_type="constant",
    report_to="tensorboard"
)

# Trainer
fine_tuning = SFTTrainer(
    model=base_model,
    train_dataset=training_data,
    peft_config=peft_parameters,
    dataset_text_field="text",
    tokenizer=llama_tokenizer,
    args=train_params
)

# Training
fine_tuning.train()
# Save Model
fine_tuning.model.save_pretrained(refined_model)
```

```
Map: 100%                                                 100/100 [00:00<00:00, 368.40 examples/s]
You are using 8-bit optimizers with a version of `bitsandbytes` < 0.41.1. It is recommended to update your version as a m
You're using a CodeLlamaTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is f
[250/250 14:47, Epoch 10/10]
```

Step	Training Loss
25	1.736500
50	1.207800
75	0.958900
100	0.830800
125	0.752600
150	0.686700
175	0.628800
200	0.565000
225	0.509500
250	0.457600

```
# Saving the chcekpoints to drive

! cp -r /content/results_modified/ '/content/drive/MyDrive/Colab Notebooks/Model Training'
! cp -r /content/CodeLlama-7b-Instruct-jForg-enhanced/ '/content/drive/MyDrive/Colab Notebooks/Model Training'


# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = " Write a jFrog pipeline to do a pipeline-example-hello-world?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
```

```
                      device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message} <</SYS>> {query} [/INST]",
                  do_sample=True,
                  top_k=10,
                  top_p = 0.9,
                  temperature = 0.2,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=200) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%                                    2/2 [00:05<00:00, 2.56s/it]
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
<s>[INST]<s>[INST]<<SYS>> You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answe
```

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh'mvn clean package'
            }
        }

        stage('Deploy') {
            steps {
                sh'mvn deploy'
            }
        }
    }
}
```

```
This pipeline defines two stages: "Build" and "Deploy". The "Build" stage uses the `mvn
```

```
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a pipeline to do a Docker Build & Publish?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message} <</SYS>> {query} [/INST]",
                  do_sample=True,
                  top_k=10,
                  top_p = 0.9,
                  temperature = 0.2,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=200) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%                                    2/2 [00:05<00:00, 2.52s/it]
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about j
```

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'docker build -t my-image.'
            }
        }
        stage('Publish') {
            steps {
                sh 'docker push my-image'
            }
        }
    }
}
```

```
This pipeline defines two stages: `Build` and `Publish`. The `Build` stage uses the `docker build` command to build the D
```

```
 # This is formatted as code
```

Experiment 2 Also Failed

- Could be model not performing in this dataset
- Will try with diffrent llama2 model

```
# Loding the previous chcekpoints

! cp -r '/content/drive/MyDrive/Colab Notebooks/Model Training/results_modified_new' /content/
! cp -r '/content/drive/MyDrive/Colab Notebooks/Model Training/Llama-2-7b-chat-jForg-enhanced/' /content/


# Model 3
# Training data converstion Experiment - 3 - new_final_data
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Model Training/new_final_data_for_training.csv')
dataset = ds.dataset(pa.Table.from_pandas(df).to_batches())

### convert to Huggingface dataset
training_data = Dataset(pa.Table.from_pandas(df))
type(training_data)
base_model_name = "NousResearch/Llama-2-7b-chat-hf" # base huggingface model for finetune
refined_model = "Llama-2-7b-chat-jForg-enhanced" # the model name we are going to give for our finetuned model



#  Tokenizer
llama_tokenizer = AutoTokenizer.from_pretrained(base_model_name, trust_remote_code=True)
llama_tokenizer.pad_token = llama_tokenizer.eos_token
llama_tokenizer.padding_side = "right"  # Fix for fp16

# Quantization Config
quant_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=False,
    llm_int8_enable_fp32_cpu_offload=True
)

# Model
base_model = AutoModelForCausalLM.from_pretrained(
    base_model_name,
    quantization_config=quant_config,
    device_map= 'auto'
)
base_model.config.use_cache = False
base_model.config.pretraining_tp = 1

# LoRA Config
peft_parameters = LoraConfig(
    lora_alpha=16,
    lora_dropout=0.1,
    r=8,
    bias="none",
    task_type="CAUSAL_LM"
)

# Training Params
train_params = TrainingArguments(
    output_dir="./results_modified_new",
    num_train_epochs=1,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=1,
    optim="paged_adamw_32bit",
    save_steps=25,
    logging_steps=25,
    learning_rate=2e-4,
    weight_decay=0.001,
    fp16=False,
    bf16=False,
    max_grad_norm=0.3,
    max_steps=500,
    warmup_ratio=0.03,
    group_by_length=True,
    lr_scheduler_type="constant",
    report_to="tensorboard"
)
```

```
# Trainer
fine_tuning = SFTTrainer(
    model=base_model,
    train_dataset=training_data,
    peft_config=peft_parameters,
    dataset_text_field="text",
    tokenizer=llama_tokenizer,
    args=train_params
)

# Training
fine_tuning.train()
# Save Model
fine_tuning.model.save_pretrained(refined_model)
```

```
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a pipeline to do a GitHub Integration?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]{query} [/INST]",
                  do_sample=True,
                  top_k=5,
```

```
                    top_p = 0.9,
                    temperature = 0.1,
                    num_return_sequences=1,
                    eos_token_id=llama_tokenizer.eos_token_id,
                    max_length=200) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%                                                 2/2 [00:01<00:00, 1.52it/s]
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:362: UserWarning: `do_sample` is s
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:367: UserWarning: `do_sample` is s
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1421: UserWarning: You have modified the pretrai
  warnings.warn(
<s>[INST]Write a pipeline to do a GitHub Integration? [/INST]  To set up a GitHub integration pipeline, you can follow th

1. Create a new pipeline:
        * In your Jenkins instance, click on "New Item" and select "Pipeline" from the drop-down menu.
        * Give your pipeline a name and select the type of pipeline you want to create (e.g., "GitHub Integration").
        * Click "Save" to create the pipeline.
2. Add a GitHub plugin:
        * In the pipeline configuration page, click on the "Manage Plugins" button.
        * Search for the "GitHub" plugin and install it.
        * Once the plugin is installed, you can configure it by providing your GitHub credentials and selecting the repos
3. Define the pipeline stages:
        * In the pipeline configuration page, you
```

```
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFro
query = "Write a jfrog pipeline to do a docker push?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message}<</SYS>>{query} [/INST]",
                  do_sample=True,
                  top_k=10,
                  top_p = 0.6,
                  temperature = 0.4,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=700) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%  2/2 [00:01<00:00, 1.44it/s]
```

```
<s>[INST]Write a jfrog pipeline to do a docker push? [/INST]  Sure! Here is an example JFrog Pipeline that pushes a Docke

1. Create a new pipeline in JFrog Artifactory by going to the "Pipelines" section in the top menu and clicking "New Pipel
2. Give the pipeline a name, such as "Docker Push".
3. Add a new stage to the pipeline by clicking the "Add Stage" button. Select "Docker" from the list of available stages.
4. In the "Docker" stage, you will need to provide the following configuration:
        * "Image": the name of the Docker image that you want to push.
        * "Repository": the name of the Docker registry where you want to push the image.
        * "Tag": the tag or label that you want to assign to the image.
        * "Push": set this to "true" to push the image to the registry.
5. Add any additional stages to the pipeline as needed, such as a "Build" stage to build the Docker image or a "Deploy" s
6. Save and activate the pipeline.

Here is an example of a JFrog Pipeline that pushes a Docker image to a registry:
```
{
  "stages": [
```

```
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a jfrog pipeline to do a GradleBuild?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message}<</SYS>>{query} [/INST]",
                  do_sample=True,
                  top_k=10,
                  top_p = 0.6,
                  temperature = 0.4,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=700) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%  2/2 [00:01<00:00, 1.37it/s]
```

```
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about j
```
```
# Define the pipeline
pipeline {
    agent any

    # Define the stages
    stages {
        stage('Build') {
            steps {
                # Run the Gradle build
                sh 'gradle build'
            }
        }
    }
}
```
```
This pipeline has a single stage, `Build`, which runs the `gradle build` command. This will execute the Gradle build scri

You can customize this pipeline by adding additional stages and steps as needed. For example, you might want to add a sta

Here are some additional examples of stages and steps you might want to include in a Gradle pipeline:

* `stage('Test') {` - Adds a stage for testing the project.
        + `steps {` - Adds a list of steps to run in this stage.
                - `sh 'gradle test'` - Runs the Gradle test task.
        + `}`
* `stage('Deploy') {` - Adds a stage for deploying the project.
        + `steps {` - Adds a list of steps to run in this stage.
                - `sh 'gradle deploy'` - Runs the Gradle deploy task.
        + `}`

You can also use the `sh` command to run any Gradle task directly in the pipeline. For example:
```
sh 'gradle build'
```
This will run the `gradle build` task directly in the pipeline, without creating a separate stage for it.

I hope this helps! Let me know if you have any questions or need further assistance.
```

Experiment 3 Failed

- Still our data dosent get high probability and model hallucinated with the data already trained on
```

- From all three expriments we tried 7B parameter model with completely new dataset
- Since these models are too complicated or not enough GPUs we might need to think about lowering the simple model for more simple dataset.
- we also reqired to long GPU run for better results from our dataset
- Further dataset tuning is required

```
training_data
```

```
Dataset({
    features: ['Unnamed: 0', 'text'],
    num_rows: 100
})
```

```
# Model 4 Long Training results
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a jfrog pipeline to do a GradleBuild?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message}<</SYS>>{query} [/INST]",
                  do_sample=True,
                  top_k=10,
                  top_p = 0.6,
                  temperature = 0.4,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=700) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%                                    2/2 [00:04<00:00, 2.26s/it]
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:362: UserWarning: `do_sample` is s
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:367: UserWarning: `do_sample` is s
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1421: UserWarning: You have modified the pretrai
  warnings.warn(
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about j
```yaml
# Define the pipeline
pipeline:
  - step:
      name: Checkout Code
        uses: actions/checkout@v2
      - step:
        name: Install Gradle
        uses: actions/install-gradle@v1
      - step:
        name: Run Gradle Build
        run: |
          gradle build
```
Let me explain each step in the pipeline:

1. `Checkout Code`: This step uses the `actions/checkout` action to check out the code from your repository. You can spec
2. `Install Gradle`: This step uses the `actions/install-gradle` action to install Gradle on the machine running the pipe
3. `Run Gradle Build`: This step runs the `gradle build` command to build your project. The `|` character is used to pass

You can customize this pipeline to fit your needs by modifying the `uses` keywords to use different actions or by adding

I hope this helps! Let me know if you have any questions or need further assistance.
```

```
# Model 4 Long Training results
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a jfrog pipeline to do a HelmBlueGreenDeploy?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message}<</SYS>>{query} [/INST]",
                  do_sample=True,
                  top_k=10,
```

```
                    top_p = 0.6,
                    temperature = 0.4,
                    num_return_sequences=1,
                    eos_token_id=llama_tokenizer.eos_token_id,
                    max_length=700) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%                                    2/2 [00:04<00:00, 2.04s/it]
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about j
```yaml
# Define the pipeline stages
stages:
  - stage: prepare
    displayName: 'Prepare deployment'
    jobs:
      - job: download-helm
        displayName: 'Download Helm'
        steps:
          - name: Download Helm
            url: https://raw.githubusercontent.com/helm/helm/v2.16.0/bin/helm
            path: helm
      - job: install-helm
        displayName: 'Install Helm'
        steps:
          - name: Install Helm
            run: |
              chmod +x./helm
             ./helm

  - stage: deploy
    displayName: 'Deploy application'
    jobs:
      - job: deploy-blue
        displayName: 'Deploy Blue'
        steps:
          - name: Deploy Blue
            helm upgrade --set-image-name blue=blue:latest --set-image-port 80 blue

  - stage: deploy-green
    displayName: 'Deploy Green'
    jobs:
      - job: deploy-green
        displayName: 'Deploy Green'
        steps:
          - name: Deploy Green
            helm upgrade --set-image-name green=green:latest --set-image-port 80 green

  - stage: verify
    displayName: 'Verify deployment'
    jobs:
      - job: verify
        displayName: 'Verify deployment'
        steps:
          - name: Verify deployment
            run: |
              helm list
              helm status
```
Let me explain each stage of the pipeline:

1. `stage: prepare`: This stage is used to prepare the deployment environment. In this stage, we download the Helm binary
2. `stage: deploy`: This stage is used to deploy the application. In this stage, we use the `helm upgrade` command to dep
3. `stage: verify`: This stage is used to verify that the deployment was successful. In this stage, we use the `helm list
```

```
# Model 4 Long Training results
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a jfrog pipeline to do a xRayscan?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message}<</SYS>>{query} [/INST]",
                  do_sample=True,
                  top_k=5,
                  top_p = 0.9,
                  temperature = 0.1,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=500) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards: 100%                                  2/2 [00:03<00:00, 1.74s/it]
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about j

Additionally, jFrog is a tool primarily used for automating and managing software development pipelines, and it is not de

I strongly advise against attempting to perform any medical procedure, including X-ray scans, without proper training and

If you have any questions or concerns about medical imaging or any other medical procedure, I encourage you to consult wi
```

▾ Model thought medical Xray scan here... Intresting

Model 4 Long GPU training results (1.5 Hours run)

```
# Model 4 Long Training results
# Generate Text
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a jfrog pipeline to do a forceXrayScan?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message}<</SYS>>{query} [/INST]",
                  do_sample=True,
                  top_k=5,
                  top_p = 0.9,
                  temperature = 0.1,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=500) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards:                                       2/2 [00:04<00:00,
100%                                                             2.10s/it]
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Helps use

To create a JFrog pipeline for a forceXrayScan, you will need to perform the

1. Install the necessary dependencies:
        * `pip install jfrog-cli`
        * `pip install xray-scan`
2. Create a new JFrog pipeline file (`jfrog-pipeline.yml`) in your project di
```yaml
# jfrog-pipeline.yml

pipelines:
  force-xray-scan:
    - step:
        name: Install dependencies
        script:
          - pip install xray-scan
          - pip install jfrog-cli
    - step:
        name: Run forceXrayScan
        script:
          - xray-scan --force
```
Explanation:

* `pipelines`: This is the top-level key that defines the pipeline.
* `force-xray-scan`: This is the name of the pipeline.
* `- step`: This is the key that defines a step in the pipeline.
* `name`: This is the name of the step.
* `script`: This is the script that will be executed in the step.

In this example, we are installing the `xray-scan` package and the `jfrog-cli`

3. Save the pipeline file and run it using the `jfrog-cli` command:
```
jfrog-cli run --pipeline-file=jfrog-pipeline.yml
```
This will execute the pipeline and run the `forceXrayScan` step.

Note: The `xray-scan` command is not included in the JFrog pipeline by defaul
```

```
# Model 4 Long Training results
# Generate Text
```

```
system_message = "You are a helpful, respectful and honest assistant. Helps user to write jFrog pipline and answers about jFrc
query = "Write a jfrog pipeline to do a Github integration?"
text_gen = pipeline(task="text-generation",
                    model=refined_model,
                    torch_dtype=torch.float16,
                    tokenizer=llama_tokenizer,
                    max_length=200,
                    device_map='auto')

output = text_gen(f"<s>[INST]<<SYS>>{system_message}<</SYS>>{query} [/INST]",
                  do_sample=False,
                  top_k=5,
                  top_p = 0.9,
                  temperature = 0.1,
                  num_return_sequences=1,
                  eos_token_id=llama_tokenizer.eos_token_id,
                  max_length=500) # can increase the length of sequence
print(output[0]['generated_text'])
```

```
Loading checkpoint shards:                                  2/2 [00:04<00:00,
100%                                                        1.97s/it]
```

```
<s>[INST]<<SYS>>You are a helpful, respectful and honest assistant. Helps use
```

```yaml
# Define the pipeline stages
stages:
  - stage: fetch
    displayName: 'Fetching code from GitHub'
    jobs:
      - job: fetch-code
        displayName: 'Fetching code from GitHub'
        steps:
          - name: Checkout code
            uses: actions/checkout@v2
          - name: Login to GitHub
            uses: GitHub-Actions/login@v1
            with:
              github_token: ${{ secrets.GITHUB_TOKEN }}
          - name: Fetch code
            run: |
              git fetch --all

  - stage: build
    displayName: 'Building code'
    jobs:
      - job: build
        displayName: 'Building code'
        steps:
          - name: Checkout code
            uses: actions/checkout@v2
          - name: Run build
            run: |
              npm run build

  - stage: deploy
    displayName: 'Deploying code'
    jobs:
      - job: deploy
        displayName: 'Deploying code'
        steps:
          - name: Checkout code
            uses: actions/checkout@v2
          - name: Deploy code
            run: |
              npm run deploy
```

```
Let me explain each stage of the pipeline:

1. `stage: fetch`: This stage fetches the code from GitHub using the `git fet
2. `stage: build`: This stage builds the code using the `npm run build` comma
```

Mode 4 results

- After running long GPU hours model now able to shape the outputs as YAML from our data
- But still its giving wide answers like some madeup answers not completely from trained data
- Since LLMS needs to be trained more time we also need to train them on Long running servers for expected reusults ex: 6-8 Hours GPU runtime
- Alos some data finetuning required - for better results

```
Note:- Due to No GPU subscription left I have stopped the further training and Infrenecing
```