REPORT

on

FINE-TUNING AND EVALUATING LLMS USING AMAZON SAGEMAKER

Submitted by

Arnab Maity
2105951

Submitted to

Dr. Ambika Prasad Mishra

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA - 751024

April 2024

Content

1. Introduction

Amazon SageMaker is a comprehensive machine learning service offered by Amazon Web Services (AWS), designed to simplify the entire machine learning workflow from data preparation to model deployment. It provides a range of tools and functionalities to cater to users with varying levels of expertise.

One of its key features is data labeling and preparation, facilitating efficient annotation and cleaning of datasets, which is essential for training accurate models. For model training, SageMaker offers managed environments supporting popular frameworks like TensorFlow, PyTorch, and Apache MXNet, with the ability to scale training jobs across distributed clusters for faster processing and reduced costs.

2. Basic Concepts

2.1 LLMs

Large Language Models (LLMs) represent a revolutionary advancement in the field of natural language processing (NLP). These models are characterized by their ability to understand, generate, and manipulate human language at an unprecedented scale and level of complexity.

LLMs are typically built on architectures such as Transformers, which enable them to process vast amounts of text data and learn intricate patterns and structures of language through self-supervised learning techniques.

One of the defining features of LLMs is their pre-training on large corpora of text data, followed by fine-tuning on specific tasks or domains. During pre-training, the model learns to predict the next word in a sequence of text given the context provided by preceding words. This process allows the model to develop a deep understanding of language semantics, syntax, and context, enabling it to generate coherent and contextually relevant text.

2.2 LLaMA2

Llama 2 is a family of LLMs like GPT-3 and PaLM 2. While there are some technical differences between it and other LLMs, you would really need to be deep into AI for them to mean much.

All these LLMs were developed and work in essentially the exact same way; they all use the same transformer architecture and development ideas like pretraining and fine-tuning.

When you enter a text prompt or provide Llama 2 with text input in some other way, it attempts to predict the most plausible follow-on text using its neural network—a cascading algorithm with billions of variables (called "parameters") that's modeled after the human brain. By assigning different weights to all the different parameters, and throwing in a small bit of randomness, Llama 2 can generate incredibly human-like responses.

2.3 Fine-tuning a LLM

Fine-tuning LaMa2 involves adapting its pre-trained parameters to better suit a particular task or dataset. Here's a general outline of the fine-tuning process:

Task Definition: Define the specific task or tasks you want LaMa2 to perform, such as text classification, language generation, or sentiment analysis.

Data Preparation: Collect and preprocess a dataset relevant to the task at hand. Ensure the dataset is properly annotated or labeled for supervised tasks.

Model Initialization: Initialize LaMa2 with its pre-trained weights, which have been learned from a large corpus of text data.

Fine-Tuning Process: Fine-tune LaMa2 on the task-specific dataset using techniques like gradient descent and backpropagation. During fine-tuning, the model's parameters are adjusted to minimize a defined loss function.

Hyperparameter Tuning: Adjust hyperparameters such as learning rate, batch size, and regularization techniques to optimize performance on the fine-tuning task.

Validation and Monitoring: Monitor the model's performance on a validation dataset during fine-tuning to prevent overfitting and ensure generalization to unseen data.

3. Problem Statement

Fine-tuning open LLMs from Hugging face using Amazon SageMaker, involving the following four steps:

1. Setup development environment
2. Create and prepare the dataset
3. Fine-tune LLM using **trl** on Amazon SageMaker
4. Deploy & Evaluate LLM on Amazon SageMaker

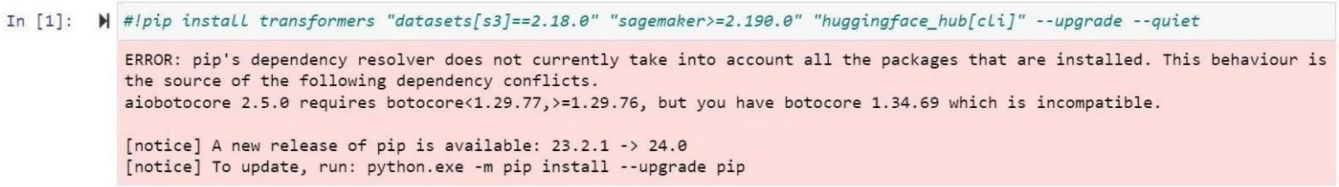
Requirement Specifications-

- AWS Account (Having an IAM role with the required permissions for SageMaker)
- Hugging Face account (For huggingface-cli login)

4. Implementation

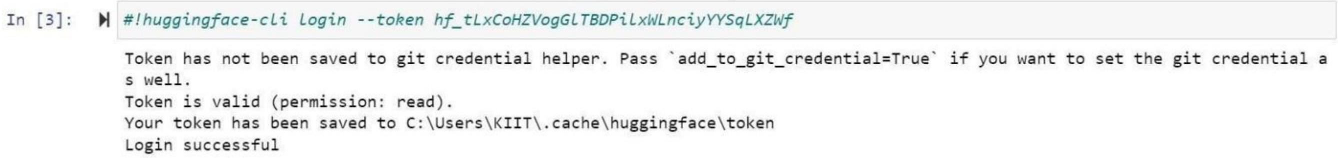
4.1 Setup development environment

Our first step is to install Hugging Face Libraries we need on the client to correctly prepare our dataset and start our training/evaluations jobs.



```
In [1]:  #!pip install transformers "datasets[s3]==2.18.0" "sagemaker>=2.190.0" "huggingface_hub[cli]" --upgrade --quiet

         ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is
         the source of the following dependency conflicts.
         aiobotocore 2.5.0 requires botocore<1.29.77,>=1.29.76, but you have botocore 1.34.69 which is incompatible.

         [notice] A new release of pip is available: 23.2.1 -> 24.0
         [notice] To update, run: python.exe -m pip install --upgrade pip
```

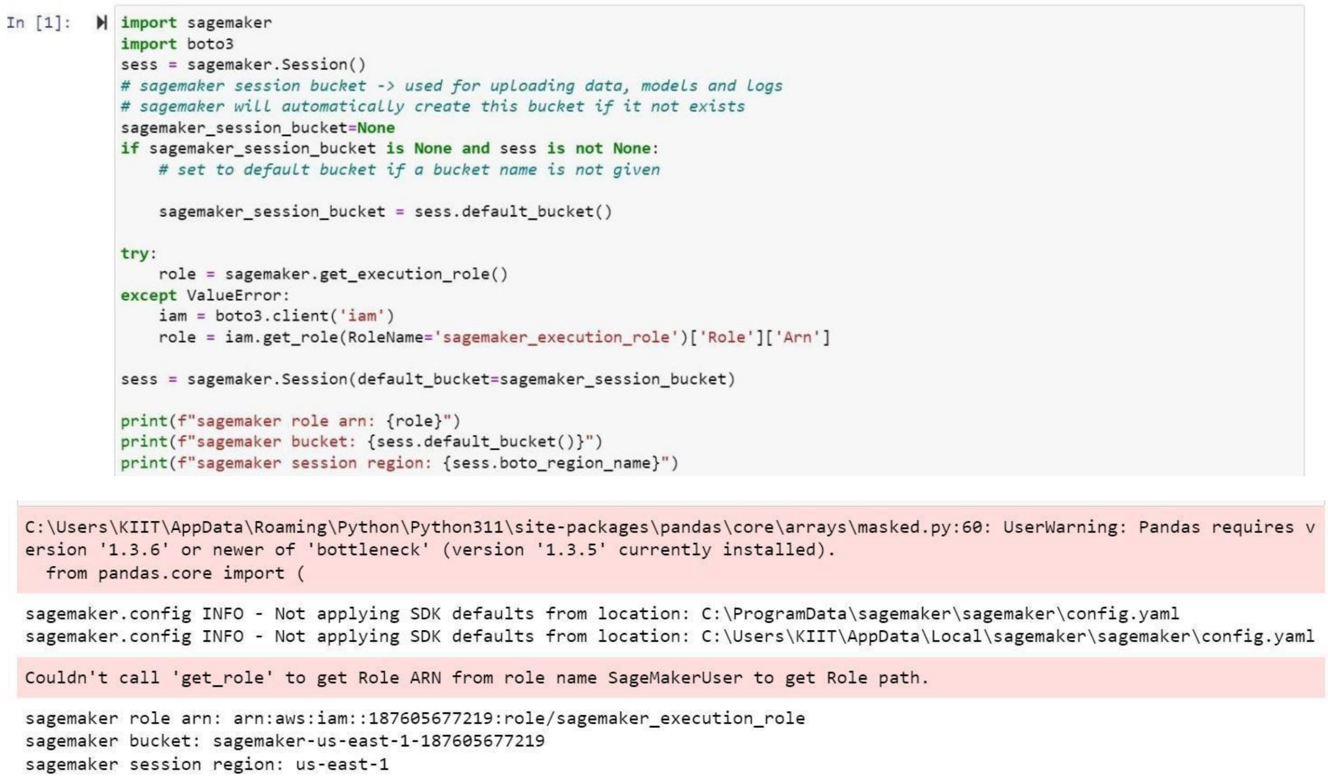
And as we will be using Llama 2 so, we need to login into our Hugging face account, for accessing the gated libraries.



```
In [3]:  #!huggingface-cli login --token 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

         Token has not been saved to git credential helper. Pass `add_to_git_credential=True` if you want to set the git credential a
         s well.
         Token is valid (permission: read).
         Your token has been saved to C:\Users\KIIT\.cache\huggingface\token
         Login successful
```

Since, we are going to use SageMaker in a local environment so, we will require an IAM role with the required permissions for SageMaker, which can be created using the AWS dashboard.

The output of the below code mentions the S3 bucket name i.e. sagemaker-us-east-1-187605677219, which we will be using further in the process and also the region in which the SageMaker is created i.e. us-east-1.



```
In [1]:  import sagemaker
         import boto3
         sess = sagemaker.Session()
         # sagemaker session bucket -> used for uploading data, models and logs
         # sagemaker will automatically create this bucket if it not exists
         sagemaker_session_bucket=None
         if sagemaker_session_bucket is None and sess is not None:
             # set to default bucket if a bucket name is not given

             sagemaker_session_bucket = sess.default_bucket()

         try:
             role = sagemaker.get_execution_role()
         except ValueError:
             iam = boto3.client('iam')
             role = iam.get_role(RoleName='sagemaker_execution_role')['Role']['Arn']

         sess = sagemaker.Session(default_bucket=sagemaker_session_bucket)

         print(f"sagemaker role arn: {role}")
         print(f"sagemaker bucket: {sess.default_bucket()}")
         print(f"sagemaker session region: {sess.boto_region_name}")
```

```
C:\Users\KIIT\AppData\Roaming\Python\Python311\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires v
ersion '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (

sagemaker.config INFO - Not applying SDK defaults from location: C:\ProgramData\sagemaker\sagemaker\config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: C:\Users\KIIT\AppData\Local\sagemaker\sagemaker\config.yaml

Couldn't call 'get_role' to get Role ARN from role name SageMakerUser to get Role path.

sagemaker role arn: arn:aws:iam::187605677219:role/sagemaker_execution_role
sagemaker bucket: sagemaker-us-east-1-187605677219
sagemaker session region: us-east-1
```

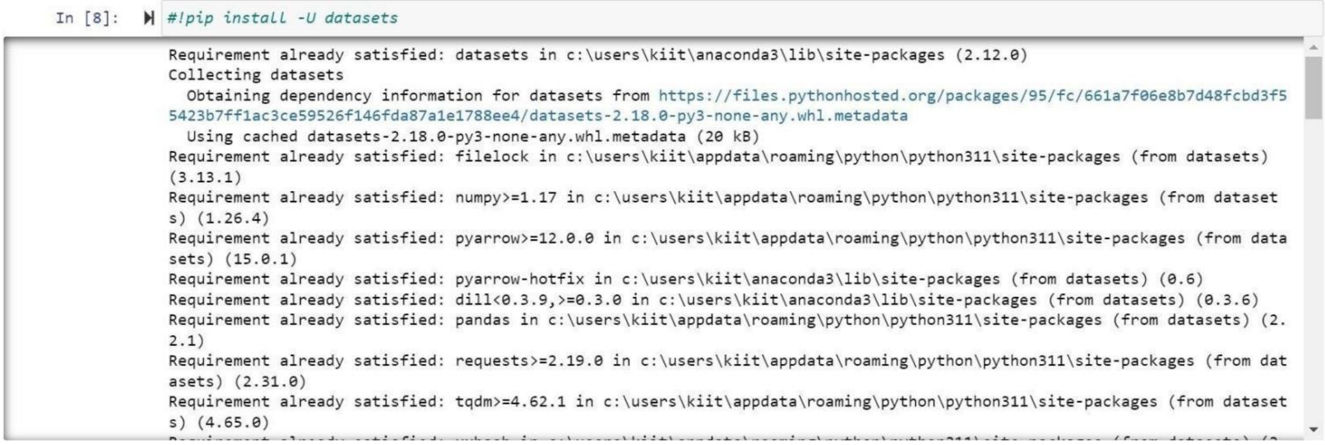
4.2 Create and prepare the dataset

After our environment is set up, we can start creating and preparing our dataset.

We are using an existing open source dataset i.e. Spider, which contains samples of natural language instructions, schema definitions and the corresponding SQL query.

We are going to use trl for fine-tuning, which supports popular instruction and conversation dataset formats. This means we only need to convert our dataset to one of the supported formats and trl will take care of the rest.

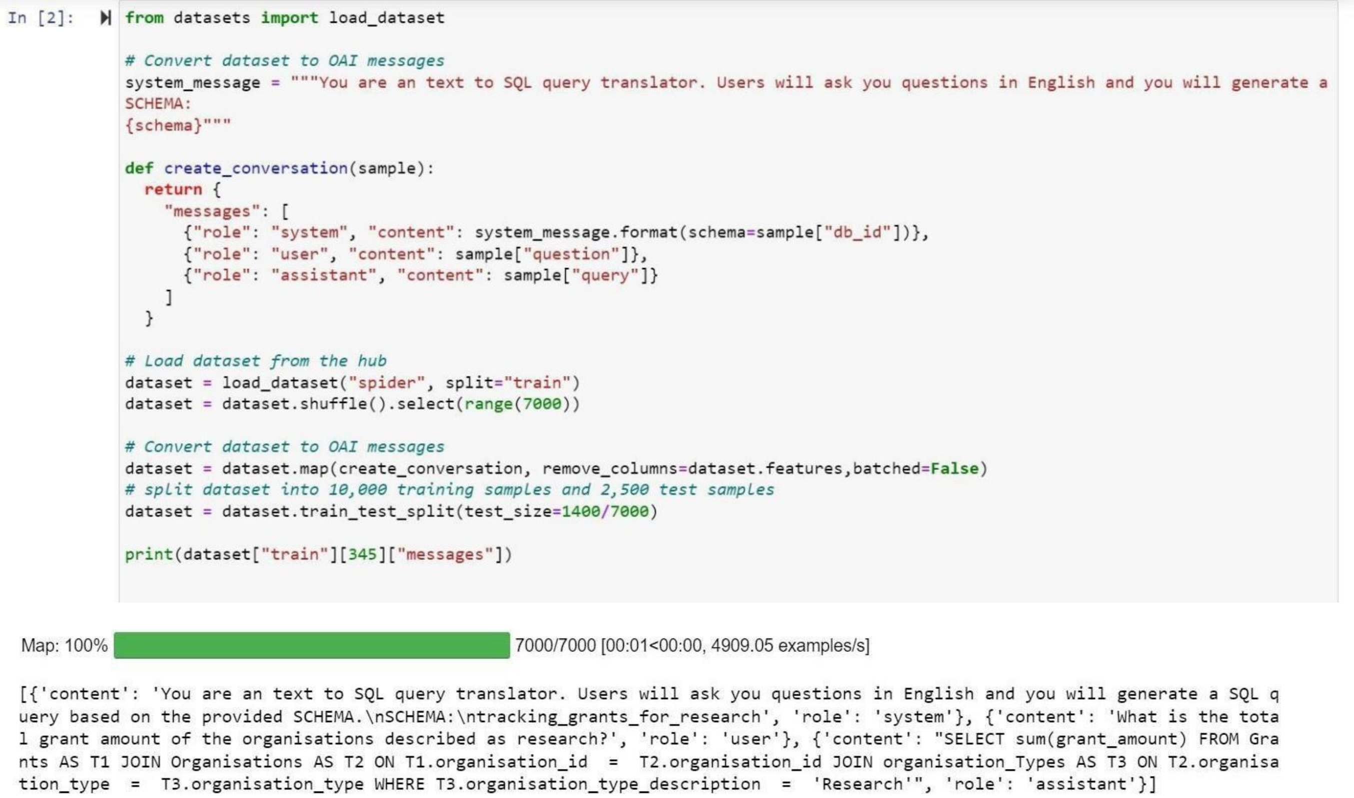
Before we write the instruction to load the dataset, we need to install the datasets library using pip.



```
In [8]:  #!pip install -U datasets
```

```
Requirement already satisfied: datasets in c:\users\kiit\anaconda3\lib\site-packages (2.12.0)
Collecting datasets
  Obtaining dependency information for datasets from https://files.pythonhosted.org/packages/95/fc/661a7f06e8b7d48fcbd3f5
5423b7ff1ac3ce59526f146fda87a1e1788ee4/datasets-2.18.0-py3-none-any.whl.metadata
  Using cached datasets-2.18.0-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: filelock in c:\users\kiit\appdata\roaming\python\python311\site-packages (from datasets)
(3.13.1)
Requirement already satisfied: numpy>=1.17 in c:\users\kiit\appdata\roaming\python\python311\site-packages (from dataset
s) (1.26.4)
Requirement already satisfied: pyarrow>=12.0.0 in c:\users\kiit\appdata\roaming\python\python311\site-packages (from data
sets) (15.0.1)
Requirement already satisfied: pyarrow-hotfix in c:\users\kiit\anaconda3\lib\site-packages (from datasets) (0.6)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in c:\users\kiit\anaconda3\lib\site-packages (from datasets) (0.3.6)
Requirement already satisfied: pandas in c:\users\kiit\appdata\roaming\python\python311\site-packages (from datasets) (2.
2.1)
Requirement already satisfied: requests>=2.19.0 in c:\users\kiit\appdata\roaming\python\python311\site-packages (from dat
asets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in c:\users\kiit\appdata\roaming\python\python311\site-packages (from dataset
s) (4.65.0)
```

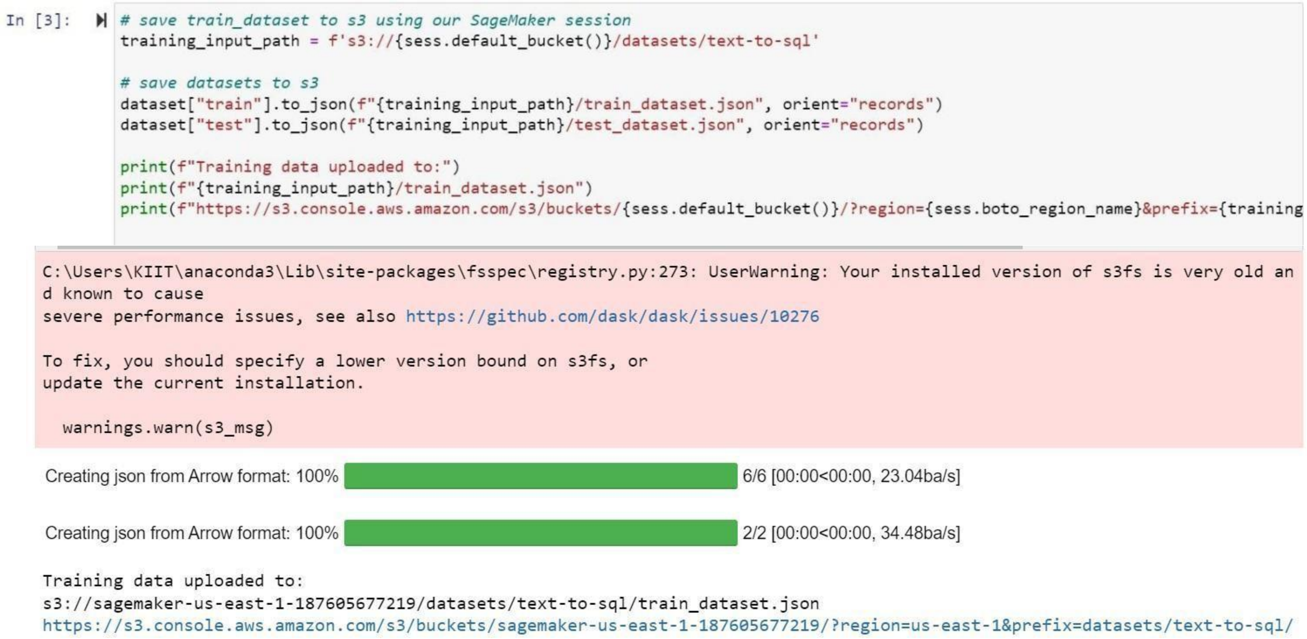
Now we will load our open-source dataset using the Hugging Face Datasets library and then convert it into the conversational format, where we include the schema definition in the system message for our assistant.



```
In [2]:  from datasets import load_dataset

         # Convert dataset to OAI messages
         system_message = """You are an text to SQL query translator. Users will ask you questions in English and you will generate a
         SCHEMA:
         {schema}"""

         def create_conversation(sample):
           return {
             "messages": [
               {"role": "system", "content": system_message.format(schema=sample["db_id"])},
               {"role": "user", "content": sample["question"]},
               {"role": "assistant", "content": sample["query"]}
             ]
           }

         # Load dataset from the hub
         dataset = load_dataset("spider", split="train")
         dataset = dataset.shuffle().select(range(7000))

         # Convert dataset to OAI messages
         dataset = dataset.map(create_conversation, remove_columns=dataset.features,batched=False)
         # split dataset into 10,000 training samples and 2,500 test samples
         dataset = dataset.train_test_split(test_size=1400/7000)

         print(dataset["train"][345]["messages"])
```

```
Map: 100% 7000/7000 [00:01<00:00, 4909.05 examples/s]

[{'content': 'You are an text to SQL query translator. Users will ask you questions in English and you will generate a SQL q
uery based on the provided SCHEMA.\nSCHEMA:\ntracking_grants_for_research', 'role': 'system'}, {'content': 'What is the tota
l grant amount of the organisations described as research?', 'role': 'user'}, {'content': "SELECT sum(grant_amount) FROM Gra
nts AS T1 JOIN Organisations AS T2 ON T1.organisation_id  =  T2.organisation_id JOIN organisation_Types AS T3 ON T2.organisa
tion_type  =  T3.organisation_type WHERE T3.organisation_type_description  =  'Research'", 'role': 'assistant'}]
```

After we processed the datasets we are going to use the FileSystem integration to upload our dataset to S3. We are using the sess.default_bucket() for the same.



```
In [3]:  # save train_dataset to s3 using our SageMaker session
         training_input_path = f's3://{sess.default_bucket()}/datasets/text-to-sql'

         # save datasets to s3
         dataset["train"].to_json(f"{training_input_path}/train_dataset.json", orient="records")
         dataset["test"].to_json(f"{training_input_path}/test_dataset.json", orient="records")

         print(f"Training data uploaded to:")
         print(f"{training_input_path}/train_dataset.json")
         print(f"https://s3.console.aws.amazon.com/s3/buckets/{sess.default_bucket()}/?region={sess.boto_region_name}&prefix={training
```

```
C:\Users\KIIT\anaconda3\Lib\site-packages\fsspec\registry.py:273: UserWarning: Your installed version of s3fs is very old an
d known to cause
severe performance issues, see also https://github.com/dask/dask/issues/10276

To fix, you should specify a lower version bound on s3fs, or
update the current installation.

  warnings.warn(s3_msg)

Creating json from Arrow format: 100%  6/6 [00:00<00:00, 23.04ba/s]

Creating json from Arrow format: 100%  2/2 [00:00<00:00, 34.48ba/s]

Training data uploaded to:
s3://sagemaker-us-east-1-187605677219/datasets/text-to-sql/train_dataset.json
https://s3.console.aws.amazon.com/s3/buckets/sagemaker-us-east-1-187605677219/?region=us-east-1&prefix=datasets/text-to-sql/
```

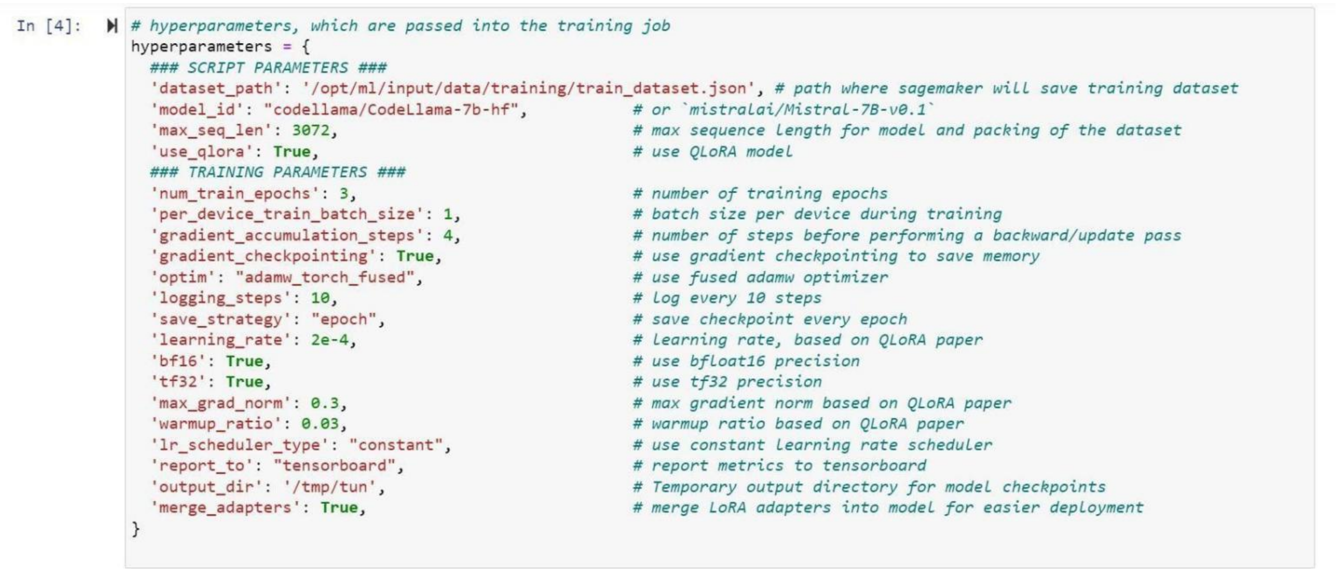
4.3 Fine-tune LLM using **trl** on Amazon SageMaker

We are now ready to fine-tune our model. We will use the SFTTrainer from trl to fine-tune our model. The SFTTrainer makes it straightforward to supervise fine-tune open LLMs.

The SFTTrainer is a subclass of the Trainer from the transformers library and supports all the same features, including logging, evaluation, and checkpointing, but adds additional quality of life features, including:

- Dataset formatting, including conversational and instruction format
- Training on completions only, ignoring prompts
- Packing datasets for more efficient training
- PEFT (parameter-efficient fine-tuning) support including Q-LoRA

- Preparing the model and tokenizer for conversational fine-tuning (e.g. adding special tokens)

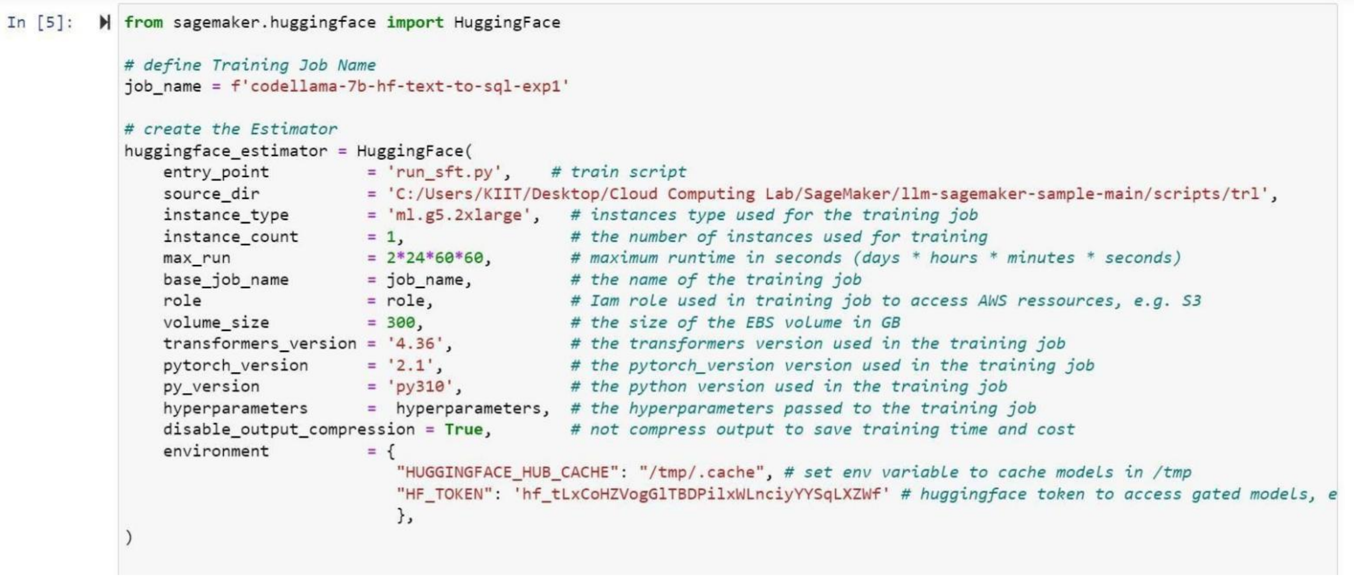


```
In [4]:  # hyperparameters, which are passed into the training job
         hyperparameters = {
           ### SCRIPT PARAMETERS ###
           'dataset_path': '/opt/ml/input/data/training/train_dataset.json', # path where sagemaker will save training dataset
           'model_id': "codellama/CodeLlama-7b-hf",           # or `mistralai/Mistral-7B-v0.1`
           'max_seq_len': 3072,                               # max sequence length for model and packing of the dataset
           'use_qlora': True,                                 # use QLoRA model
           ### TRAINING PARAMETERS ###
           'num_train_epochs': 3,                             # number of training epochs
           'per_device_train_batch_size': 1,                  # batch size per device during training
           'gradient_accumulation_steps': 4,                  # number of steps before performing a backward/update pass
           'gradient_checkpointing': True,                    # use gradient checkpointing to save memory
           'optim': "adamw_torch_fused",                      # use fused adamw optimizer
           'logging_steps': 10,                               # log every 10 steps
           'save_strategy': "epoch",                          # save checkpoint every epoch
           'learning_rate': 2e-4,                             # learning rate, based on QLoRA paper
           'bf16': True,                                      # use bfloat16 precision
           'tf32': True,                                      # use tf32 precision
           'max_grad_norm': 0.3,                              # max gradient norm based on QLoRA paper
           'warmup_ratio': 0.03,                              # warmup ratio based on QLoRA paper
           'lr_scheduler_type': "constant",                   # use constant learning rate scheduler
           'report_to': "tensorboard",                        # report metrics to tensorboard
           'output_dir': '/tmp/tun',                          # Temporary output directory for model checkpoints
           'merge_adapters': True,                            # merge LoRA adapters into model for easier deployment
         }
```

In order to create a sagemaker training job we need a HuggingFace Estimator. The Estimator handles end-to-end Amazon SageMaker training and deployment tasks. The Estimator manages the infrastructure use.

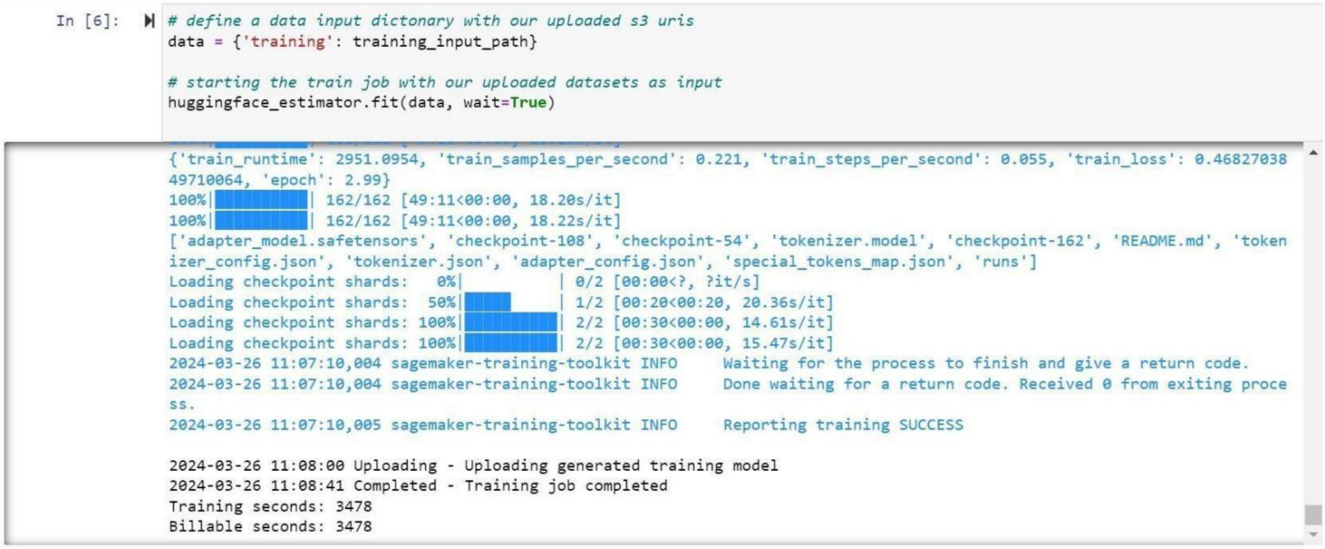
Amazon SageMaker takes care of starting and managing all the required ec2 instances for us, provides the correct huggingface container, uploads the provided scripts and downloads the data from our S3 bucket into the container at /opt/ml/input/data. Then, it starts the training job by running.

The HuggingFace() estimator is instantiated with parameters specifying the training script, source directory, instance type, maximum runtime, and other configurations such as hyperparameters and environment variables. This setup enables efficient training of the model on SageMaker infrastructure, leveraging Hugging Face's powerful Transformers library for natural language processing tasks.



```
In [5]:  from sagemaker.huggingface import HuggingFace

         # define Training Job Name
         job_name = f'codellama-7b-hf-text-to-sql-exp1'

         # create the Estimator
         huggingface_estimator = HuggingFace(
             entry_point          = 'run_sft.py',    # train script
             source_dir           = 'C:/Users/KIIT/Desktop/Cloud Computing Lab/SageMaker/llm-sagemaker-sample-main/scripts/trl',
             instance_type        = 'ml.g5.2xlarge',   # instances type used for the training job
             instance_count       = 1,                 # the number of instances used for training
             max_run              = 2*24*60*60,        # maximum runtime in seconds (days * hours * minutes * seconds)
             base_job_name        = job_name,          # the name of the training job
             role                 = role,              # Iam role used in training job to access AWS ressources, e.g. S3
             volume_size          = 300,               # the size of the EBS volume in GB
             transformers_version = '4.36',            # the transformers version used in the training job
             pytorch_version      = '2.1',             # the pytorch_version version used in the training job
             py_version           = 'py310',           # the python version used in the training job
             hyperparameters      =  hyperparameters,  # the hyperparameters passed to the training job
             disable_output_compression = True,        # not compress output to save training time and cost
             environment          = {
                                     "HUGGINGFACE_HUB_CACHE": "/tmp/.cache", # set env variable to cache models in /tmp
                                     "HF_TOKEN": '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' # huggingface token to access gated models, e
                                     },
         )
```

We can now start our training job, with the **.fit()** method passing our S3 path to the training script.

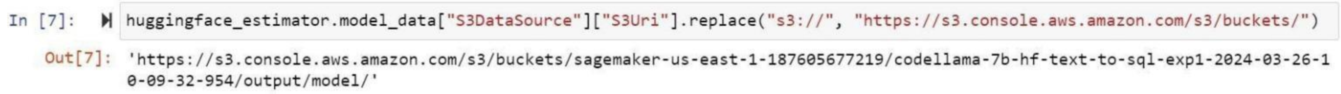


```
In [6]:  # define a data input dictonary with our uploaded s3 uris
         data = {'training': training_input_path}

         # starting the train job with our uploaded datasets as input
         huggingface_estimator.fit(data, wait=True)
```

```
{'train_runtime': 2951.0954, 'train_samples_per_second': 0.221, 'train_steps_per_second': 0.055, 'train_loss': 0.46827038
49710064, 'epoch': 2.99}
100%|██████████| 162/162 [49:11<00:00, 18.20s/it]
100%|██████████| 162/162 [49:11<00:00, 18.22s/it]
['adapter_model.safetensors', 'checkpoint-108', 'checkpoint-54', 'tokenizer.model', 'checkpoint-162', 'README.md', 'token
izer_config.json', 'tokenizer.json', 'adapter_config.json', 'special_tokens_map.json', 'runs']
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
Loading checkpoint shards:  50%|█████     | 1/2 [00:20<00:20, 20.36s/it]
Loading checkpoint shards: 100%|██████████| 2/2 [00:30<00:00, 14.61s/it]
Loading checkpoint shards: 100%|██████████| 2/2 [00:30<00:00, 15.47s/it]
2024-03-26 11:07:10,004 sagemaker-training-toolkit INFO     Waiting for the process to finish and give a return code.
2024-03-26 11:07:10,004 sagemaker-training-toolkit INFO     Done waiting for a return code. Received 0 from exiting proce
ss.
2024-03-26 11:07:10,005 sagemaker-training-toolkit INFO     Reporting training SUCCESS

2024-03-26 11:08:00 Uploading - Uploading generated training model
2024-03-26 11:08:41 Completed - Training job completed
Training seconds: 3478
Billable seconds: 3478
```

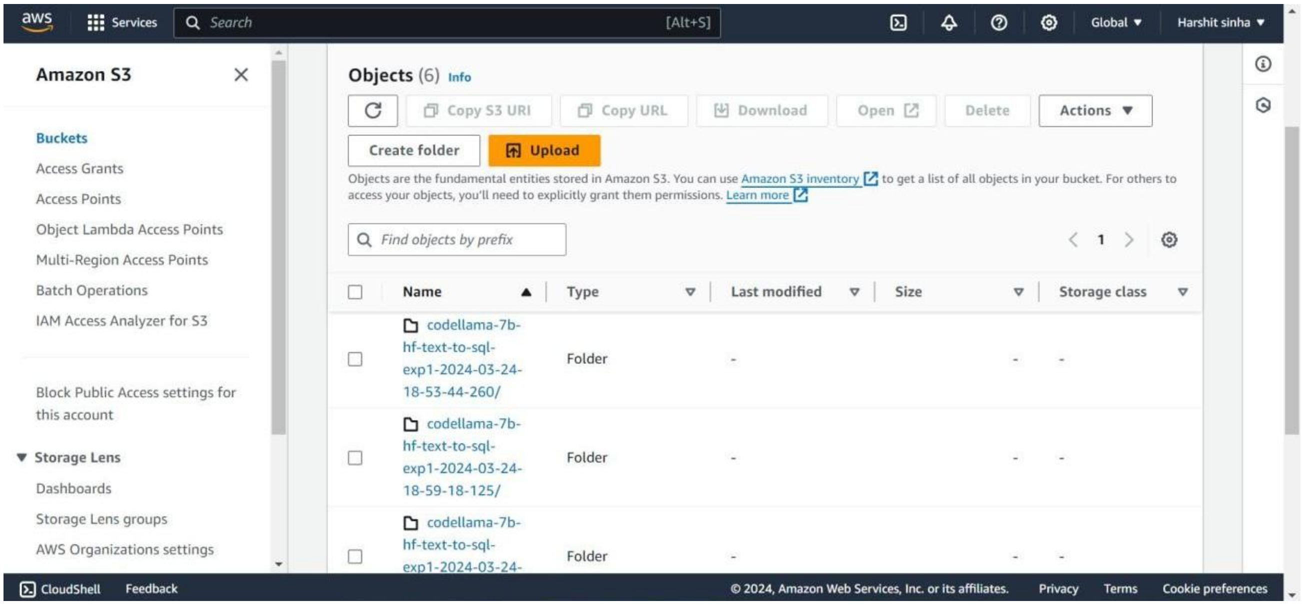
In our case for CodeLlama 7B, the SageMaker training job took 3478 seconds, which is about 0.96611 hour (~ 1hour). The ml.g5.2xlarge instance we used costs $1.15 per hour for on-demand usage. As a result, the total cost for training our fine-tuned Code LLama model was only ~$1.76.

Now make sure SageMaker has successfully uploaded the model to S3. We can use the model_data property of the estimator to get the S3 path to the model. Since we used merge_weights=True and disable_output_compression=True the model is stored as raw files in the S3 bucket.



```
In [7]:  huggingface_estimator.model_data["S3DataSource"]["S3Uri"].replace("s3://", "https://s3.console.aws.amazon.com/s3/buckets/")

Out[7]: 'https://s3.console.aws.amazon.com/s3/buckets/sagemaker-us-east-1-187605677219/codellama-7b-hf-text-to-sql-exp1-2024-03-26-1
        0-09-32-954/output/model/'
```

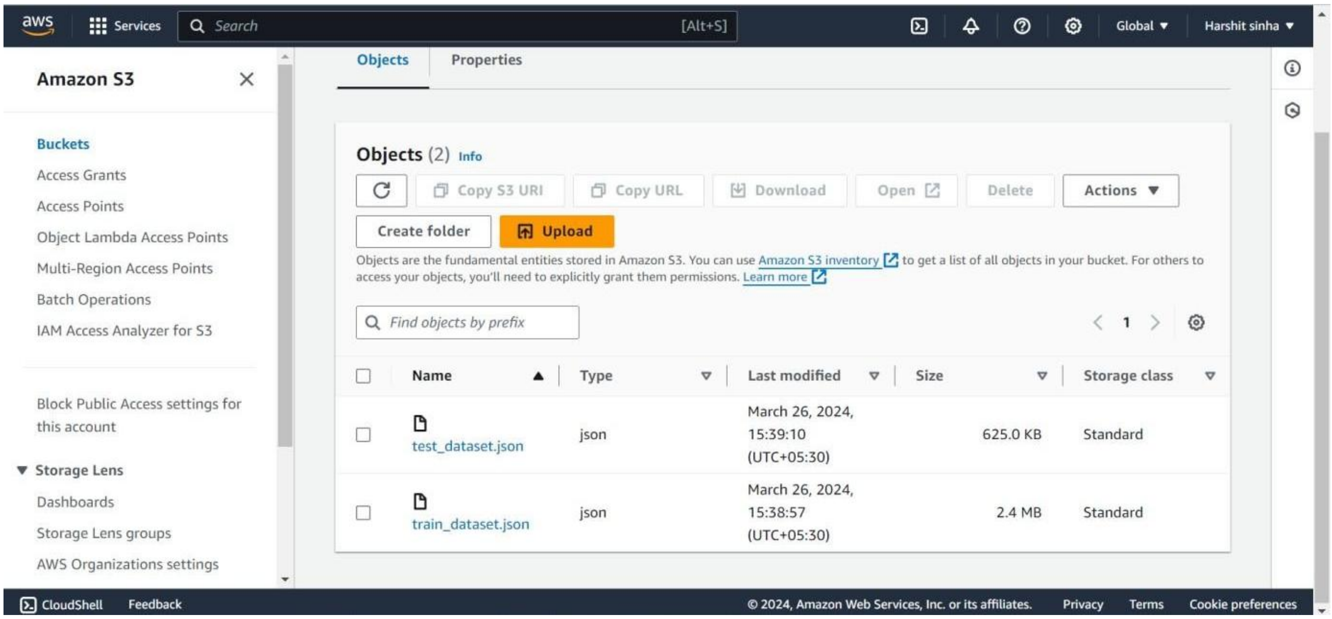
Using the AWS dashboard we can see the below folder structure in our S3 bucket.



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class
□	codellama-7b-hf-text-to-sql-exp1-2024-03-24-18-53-44-260/	Folder	-	-	-
□	codellama-7b-hf-text-to-sql-exp1-2024-03-24-18-59-18-125/	Folder	-	-	-
□	codellama-7b-hf-text-to-sql-exp1-2024-03-24-	Folder	-	-	-

And also the training & testing dataset JSON files.

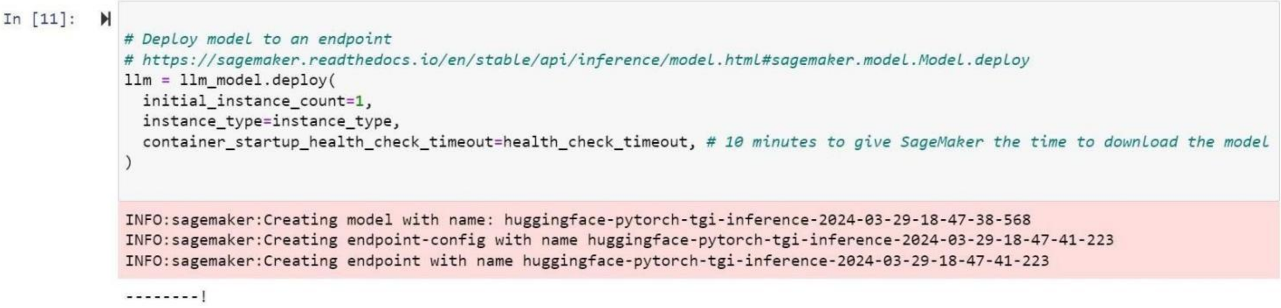


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name ▲	Type	Last modified	Size	Storage class
□	test_dataset.json	json	March 26, 2024, 15:39:10 (UTC+05:30)	625.0 KB	Standard
□	train_dataset.json	json	March 26, 2024, 15:38:57 (UTC+05:30)	2.4 MB	Standard

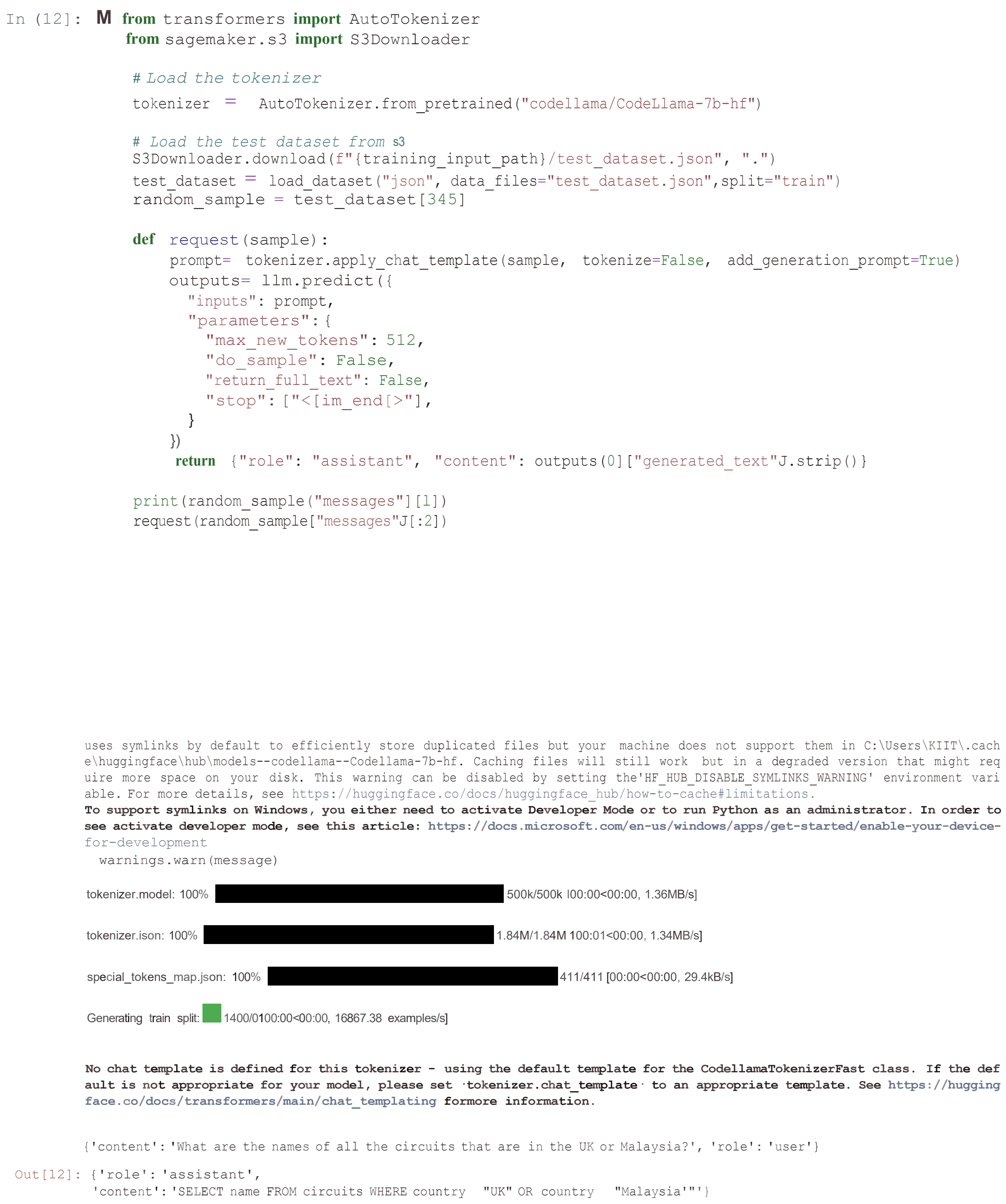
4.4 Deploy & Evaluate LLM on Amazon SageMaker

We are now ready to fine-tune our model. We will use the SFTTrainer from trl to fine-tune our model. The SFTTrainer makes it straightforward to supervise fine-tune open LLMs.



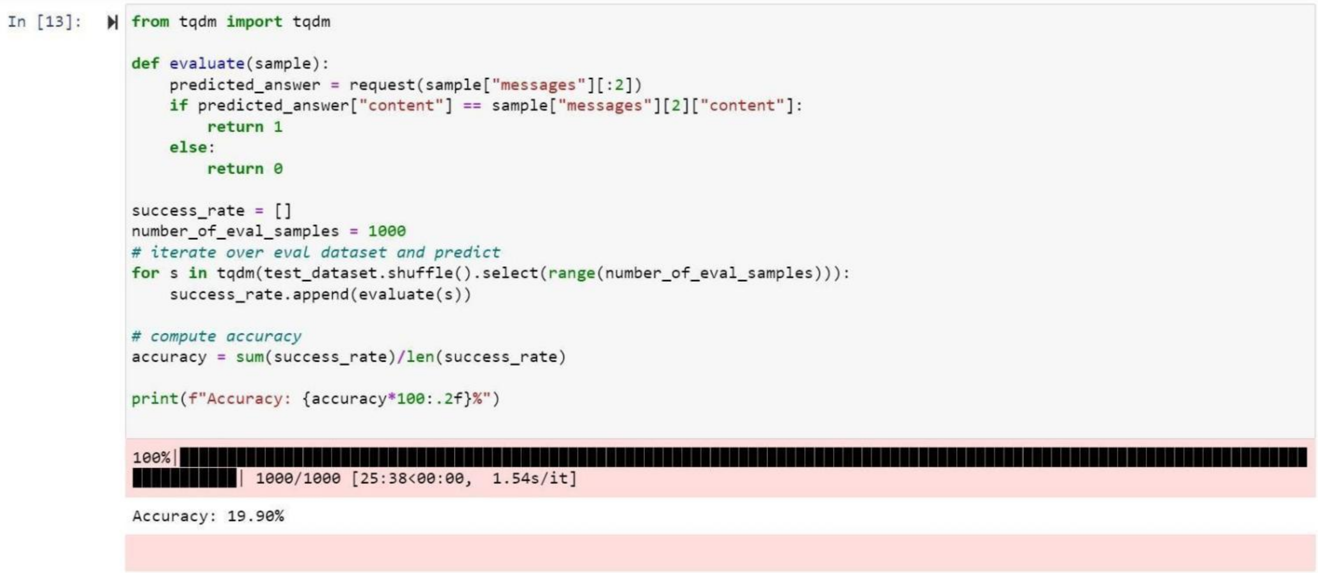
```
In [11]:

# Deploy model to an endpoint
# https://sagemaker.readthedocs.io/en/stable/api/inference/model.html#sagemaker.model.Model.deploy
llm = llm_model.deploy(
  initial_instance_count=1,
  instance_type=instance_type,
  container_startup_health_check_timeout=health_check_timeout, # 10 minutes to give SageMaker the time to download the model
)
```

```
INFO:sagemaker:Creating model with name: huggingface-pytorch-tgi-inference-2024-03-29-18-47-38-568
INFO:sagemaker:Creating endpoint-config with name huggingface-pytorch-tgi-inference-2024-03-29-18-47-41-223
INFO:sagemaker:Creating endpoint with name huggingface-pytorch-tgi-inference-2024-03-29-18-47-41-223

--------!
```



```
In (12]:  M  from transformers import AutoTokenizer
              from sagemaker.s3 import S3Downloader

              # Load the tokenizer
              tokenizer  =  AutoTokenizer.from_pretrained("codellama/CodeLlama-7b-hf")

              # Load the test dataset from s3
              S3Downloader.download(f"{training_input_path}/test_dataset.json", ".")
              test_dataset = load_dataset("json", data_files="test_dataset.json",split="train")
              random_sample = test_dataset[345]

              def request(sample):
                  prompt= tokenizer.apply_chat_template(sample,  tokenize=False,  add_generation_prompt=True)
                  outputs= llm.predict({
                    "inputs": prompt,
                    "parameters":{
                      "max_new_tokens": 512,
                      "do_sample": False,
                      "return_full_text": False,
                      "stop": ["<[im_end[>"],
                    }
                  })
                   return  {"role": "assistant",  "content": outputs(0]["generated_text"J.strip()}

              print(random_sample("messages"][l])
              request(random_sample["messages"J[:2])
```

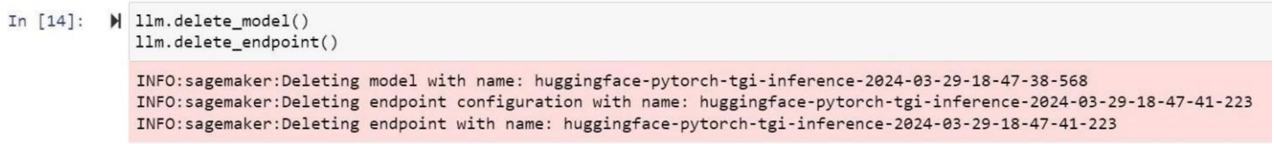
```
uses symlinks by default to efficiently store duplicated files but your  machine does not support them in C:\Users\KIIT\.cach
e\huggingface\hub\models--codellama--Codellama-7b-hf. Caching files will still work  but in a degraded version that might req
uire more space on your disk. This warning can be disabled by setting the'HF_HUB_DISABLE_SYMLINKS_WARNING' environment vari
able. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to
see activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-
for-development
  warnings.warn(message)

tokenizer.model: 100%  500k/500k I00:00<00:00, 1.36MB/s]

tokenizer.ison: 100%  1.84M/1.84M 100:01<00:00, 1.34MB/s]

special_tokens_map.json:  100%  411/411 [00:00<00:00, 29.4kB/s]

Generating  train  split:  1400/0100:00<00:00, 16867.38  examples/s]


No chat template is defined for this tokenizer - using the default template for the CodellamaTokenizerFast class. If the def
ault is not appropriate for your model, please set ·tokenizer.chat_template· to an appropriate template. See https://hugging
face.co/docs/transformers/main/chat_templating formore information.


{'content':'What are the names of all the circuits that are in the UK or Malaysia?', 'role':'user'}

Out[12]: {'role':'assistant',
          'content':'SELECT name FROM circuits WHERE country   "UK" OR country   "Malaysia'"'}
```



```
In [13]:  from tqdm import tqdm

           def evaluate(sample):
               predicted_answer = request(sample["messages"][:2])
               if predicted_answer["content"] == sample["messages"][2]["content"]:
                   return 1
               else:
                   return 0

           success_rate = []
           number_of_eval_samples = 1000
           # iterate over eval dataset and predict
           for s in tqdm(test_dataset.shuffle().select(range(number_of_eval_samples))):
               success_rate.append(evaluate(s))

           # compute accuracy
           accuracy = sum(success_rate)/len(success_rate)

           print(f"Accuracy: {accuracy*100:.2f}%")
```

```
100%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████
███████████| 1000/1000 [25:38<00:00,  1.54s/it]

Accuracy: 19.90%
```

Now we will delete the end points as these adds on to the AWS bill.

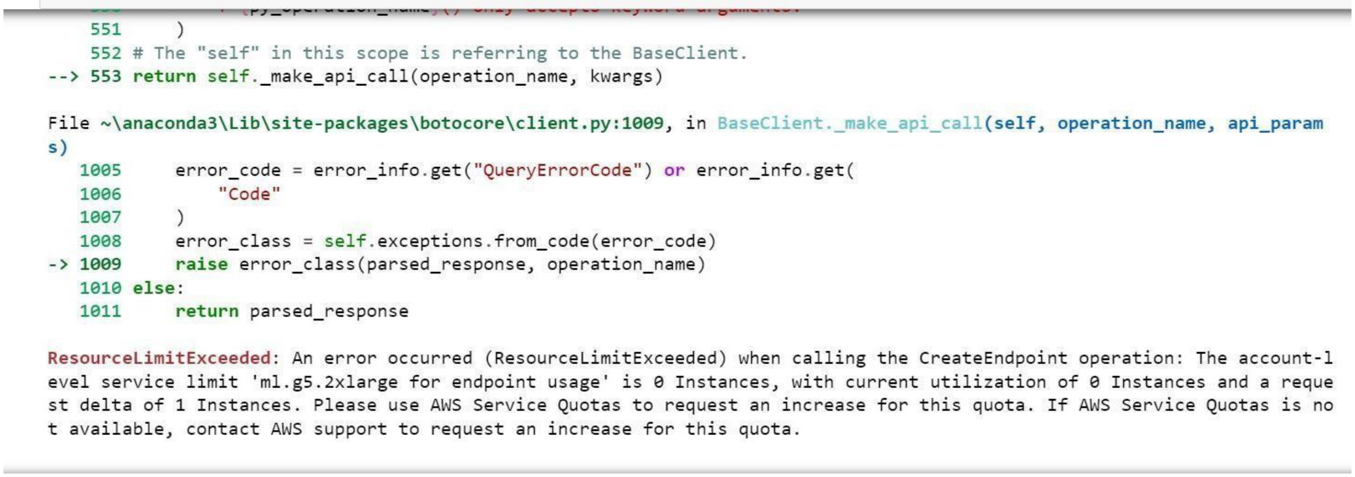


```
In [14]:  llm.delete_model()
           llm.delete_endpoint()
```

```
INFO:sagemaker:Deleting model with name: huggingface-pytorch-tgi-inference-2024-03-29-18-47-38-568
INFO:sagemaker:Deleting endpoint configuration with name: huggingface-pytorch-tgi-inference-2024-03-29-18-47-41-223
INFO:sagemaker:Deleting endpoint with name: huggingface-pytorch-tgi-inference-2024-03-29-18-47-41-223
```

4.5 Errors Encountered in the process and ways to resolve them.

An error regarding the limit of ‘ml.g5.2xlarge for endpoint usage’ was encountered, which was solved by requesting an increase in the limit using the ‘Service Quotas’ present in AWS Console.



```
    551     )
    552 # The "self" in this scope is referring to the BaseClient.
--> 553 return self._make_api_call(operation_name, kwargs)

File ~\anaconda3\Lib\site-packages\botocore\client.py:1009, in BaseClient._make_api_call(self, operation_name, api_param
s)
   1005     error_code = error_info.get("QueryErrorCode") or error_info.get(
   1006         "Code"
   1007     )
   1008     error_class = self.exceptions.from_code(error_code)
-> 1009     raise error_class(parsed_response, operation_name)
   1010 else:
   1011     return parsed_response

ResourceLimitExceeded: An error occurred (ResourceLimitExceeded) when calling the CreateEndpoint operation: The account-l
evel service limit 'ml.g5.2xlarge for endpoint usage' is 0 Instances, with current utilization of 0 Instances and a reque
st delta of 1 Instances. Please use AWS Service Quotas to request an increase for this quota. If AWS Service Quotas is no
t available, contact AWS support to request an increase for this quota.
```

5. Conclusion

In conclusion, the project has provided a comprehensive guide to fine-tuning open LLMs from Hugging Face using Amazon SageMaker. Through the outlined steps of setting up the development environment, creating and preparing the dataset, fine-tuning the LLM using trl, and deploying and evaluating it on Amazon SageMaker, the project offers practical insights into harnessing these sophisticated language models.

The project's emphasis on adaptability, demonstrated by its ability to run on smaller GPU instances while remaining scalable to larger configurations, highlights its versatility and cost-effectiveness. By integrating Hugging Face's models with Amazon SageMaker's infrastructure, the project showcases the seamless integration of cutting-edge natural language processing technologies with scalable cloud computing resources.

6. References

https://www.philschmid.de/sagemaker-train-evalaute-llms-2024

https://llama.meta.com/

https://www.geeksforgeeks.org/what-is-sagemaker-in-aws/