**REPORT**

**on**

**“Fine-Tuning and Evaluating LLMs using Amazon SageMaker”**

**Submitted by**

**Harshit sinha** 21051895

**Submitted to**

**Mr. Ambika Prasad Rath**

****

**SCHOOL OF COMPUTER ENGINEERING**

**KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY**

**BHUBANESWAR, ODISHA - 751024**

**April 2020**

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/ Literature Review

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 LaMa2

LaMa2, or Language Model for Latent Meaning Analysis 2, is an advanced large language model designed to excel in tasks related to understanding and analyzing the latent meanings within text. Developed by researchers at OpenAI, LaMa2 represents a significant advancement in natural language processing (NLP) technology.

LaMa2 is built upon the Transformer architecture, a powerful framework for processing sequential data. Like other large language models, LaMa2 has been pre-trained on vast amounts of text data to learn the intricacies of human language. Through self-supervised learning techniques, LaMa2 has developed a deep understanding of language semantics, syntax, and context.

2.3 Fine-tuning and evaluation

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.

Evaluation of Fine-Tuned LaMa2:

Once fine-tuning is complete, it's crucial to evaluate the performance of the fine-tuned model. Evaluation metrics vary depending on the task, but common evaluation techniques include:

**Task-Specific Metrics:** Use metrics relevant to the fine-tuning task. For instance, accuracy, precision, recall, F1-score for classification tasks; BLEU score for language translation; and perplexity for language modeling tasks.

**Cross-Validation:** Split the dataset into training and validation sets or employ cross-validation techniques to assess the model's performance across different subsets of the data.

**Human Evaluation:** In addition to automated metrics, consider conducting human evaluations to assess the quality of the model's output, especially for tasks involving language generation or understanding.

**Generalization Testing:** Test the fine-tuned LaMa2 on unseen or held-out test data to evaluate its ability to generalize to new samples.

**Comparison with Baselines:** Compare the performance of the fine-tuned LaMa2 with baseline models or previous state-of-the-art approaches to gauge its effectiveness.

3. Problem Statement

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

1. [Setup development environment](https://www.philschmid.de/sagemaker-train-evalaute-llms-2024#1-setup-development-environment)
2. [Create and prepare the dataset](https://www.philschmid.de/sagemaker-train-evalaute-llms-2024#2-create-and-prepare-the-dataset)
3. [Fine-tune LLM using **trl**on Amazon SageMaker](https://www.philschmid.de/sagemaker-train-evalaute-llms-2024#3-fine-tune-mistral-7b-with-qlora-on-amazon-sagemaker)
4. [Deploy & Evaluate LLM on Amazon SageMaker](https://www.philschmid.de/sagemaker-train-evalaute-llms-2024#4-deploy--evaluate-llm-on-amazon-sagemaker)

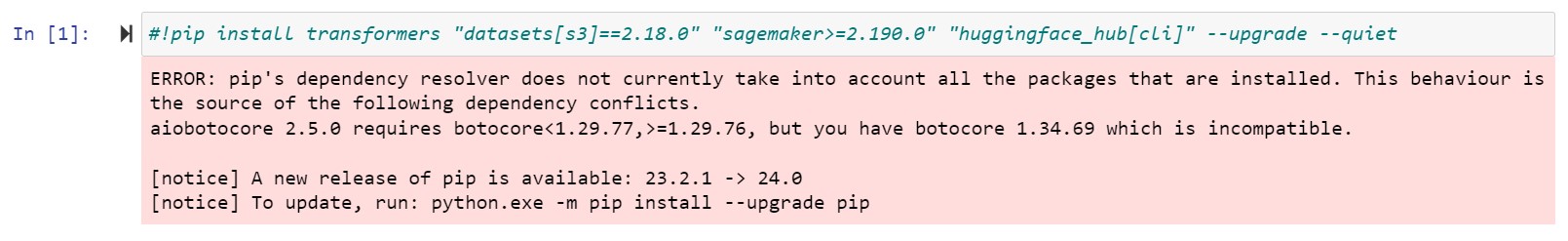
Requirement Specifications-

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

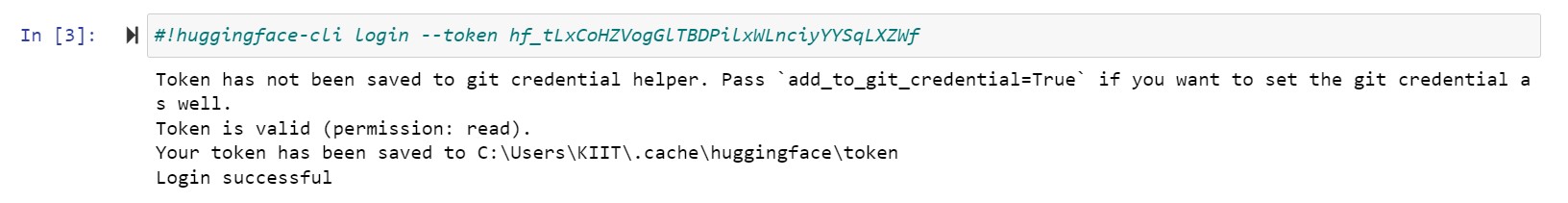
4. Implementation

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.

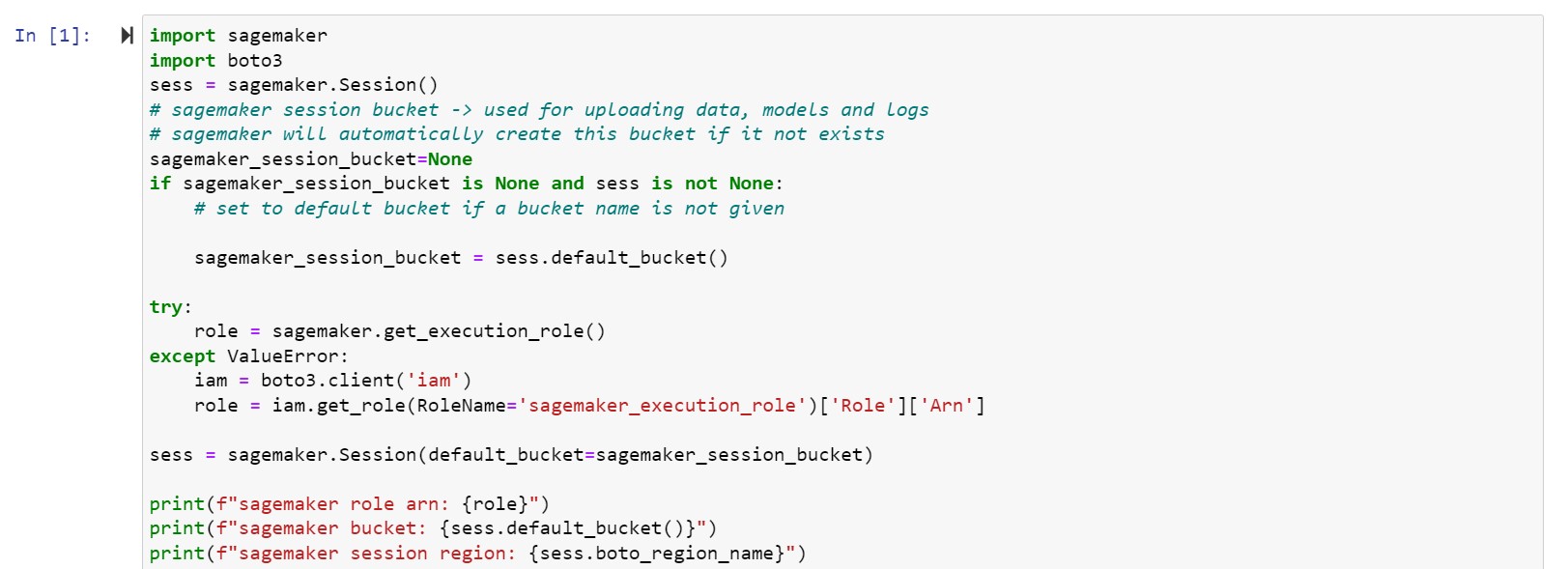


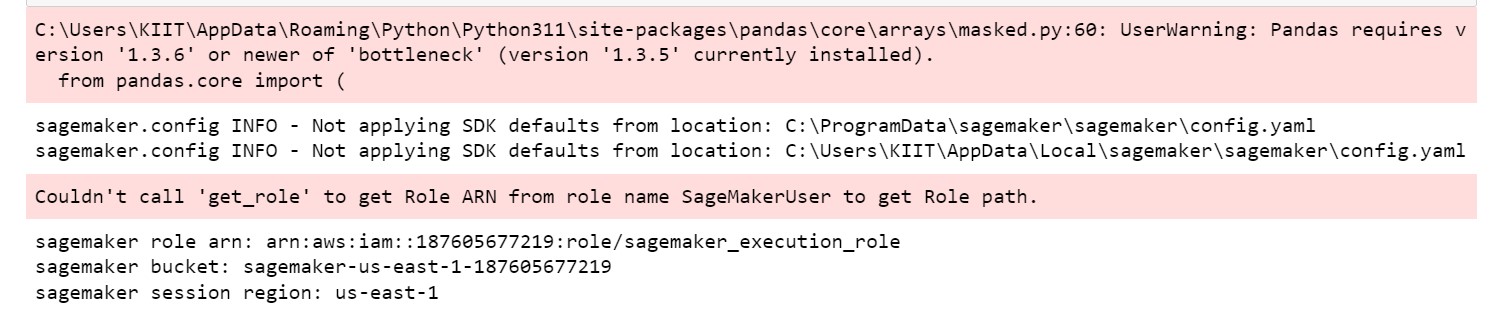
And as we will be using Llama 2 so, we need to login into our Hugging face account, for accessing the gated libraries.



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.





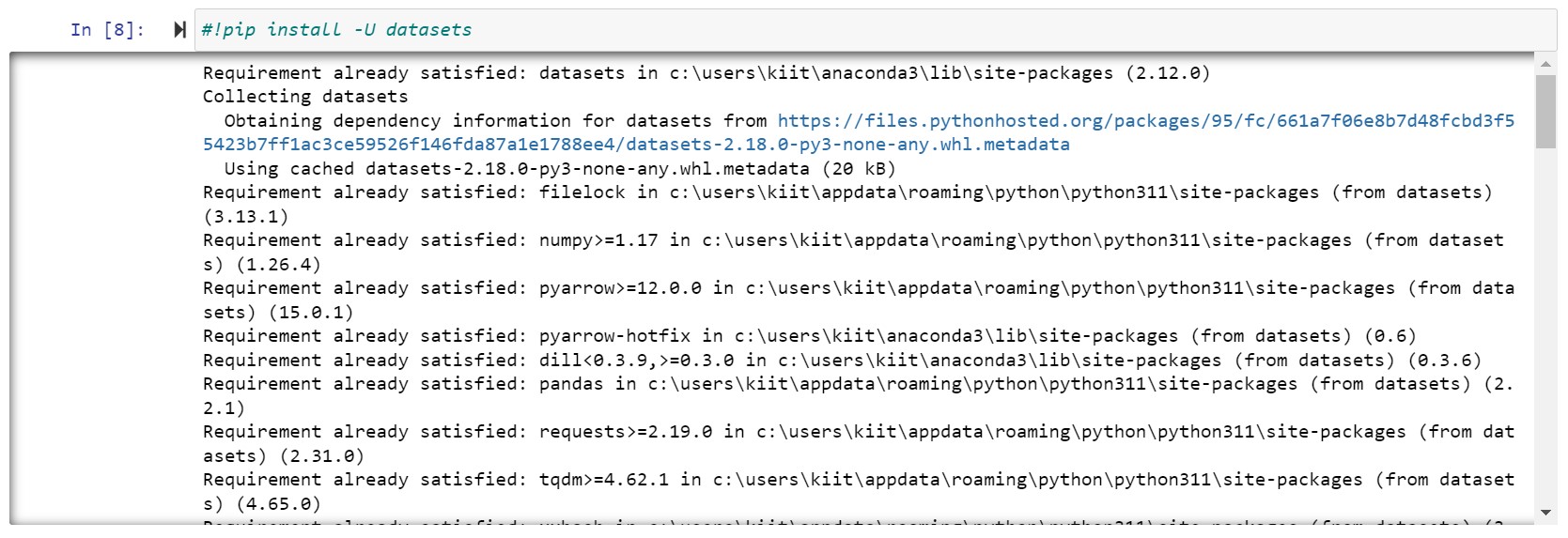
4.2 Create and prepare the dataset

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.

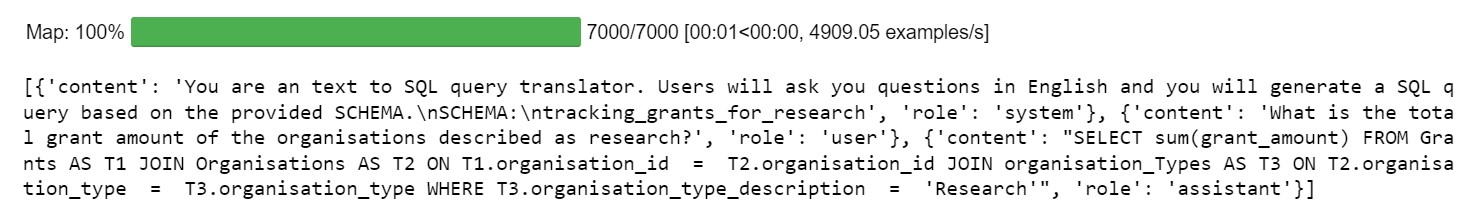
We are going to use trl for fine-tuning, which supports popular instruction and conversation dataset formats. This means we only need to convert our dataset to one of the supported formats and trl will take care of the rest.

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

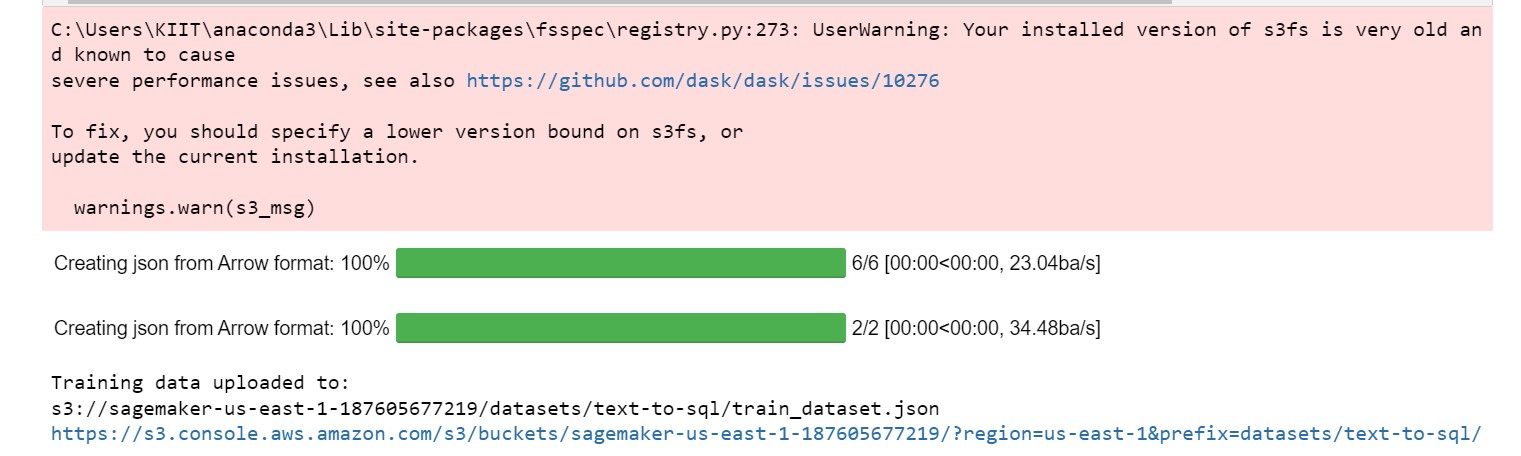


Now we will load our open-source dataset using the Hugging Face Datasets library and then convert it into the conversational format, where we include the schema definition in the system message for our assistant.





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.

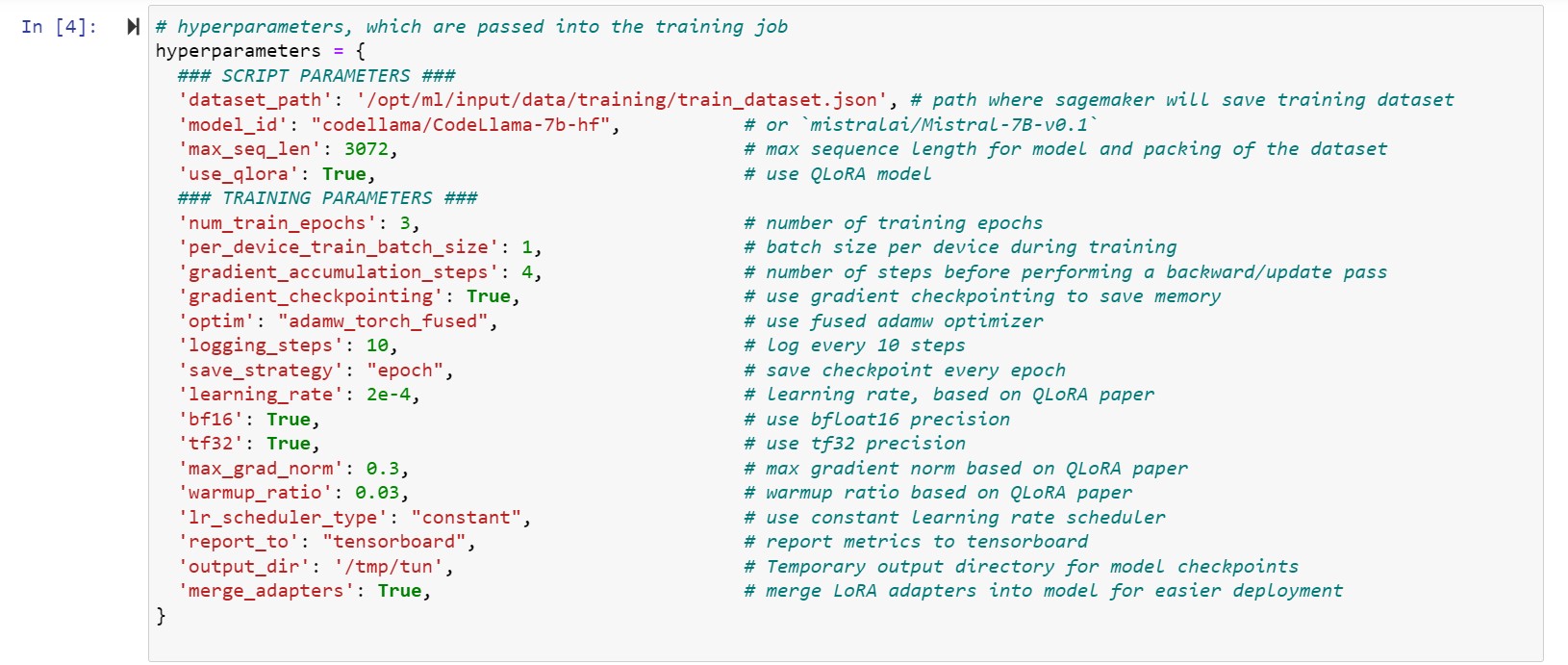


4.3 Fine-tune LLM using **trl** on Amazon SageMaker

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)



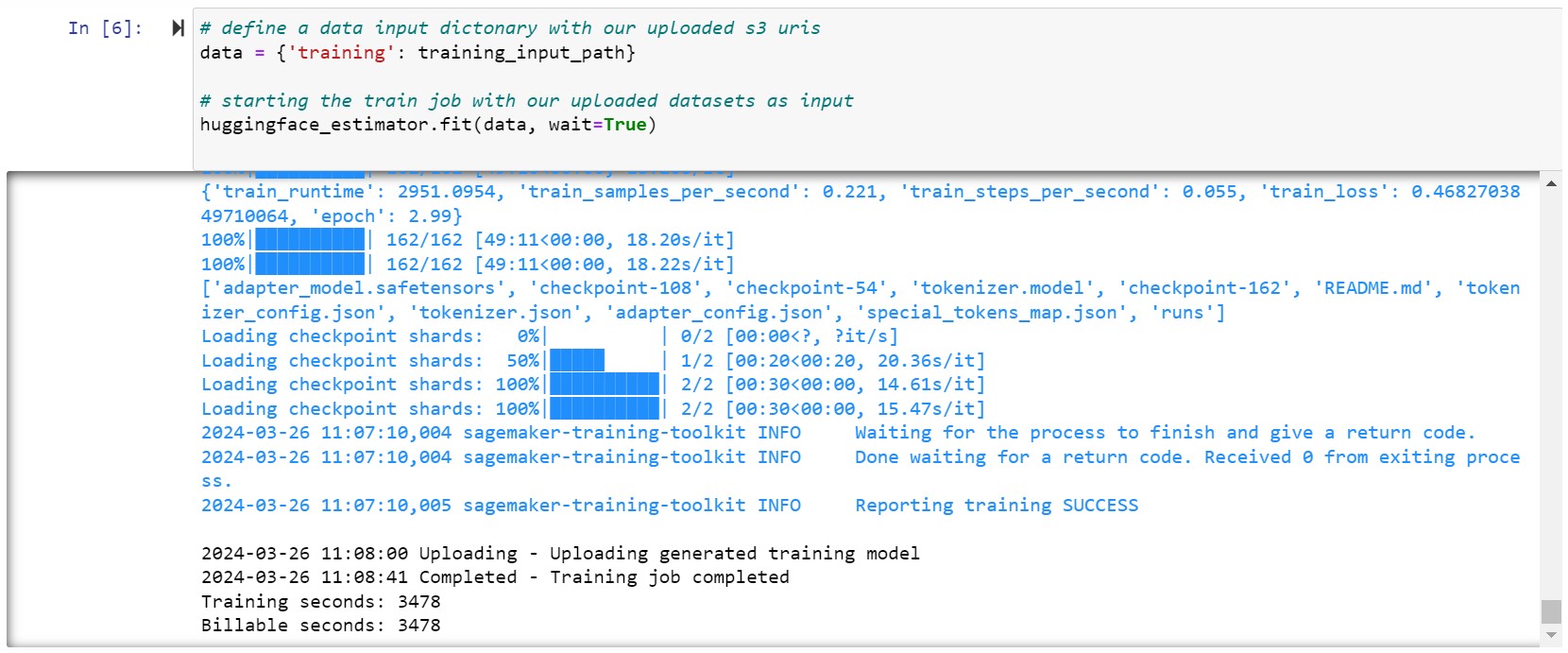
In order to create a sagemaker training job we need a HuggingFace Estimator. The Estimator handles end-to-end Amazon SageMaker training and deployment tasks. The Estimator manages the infrastructure use.

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.

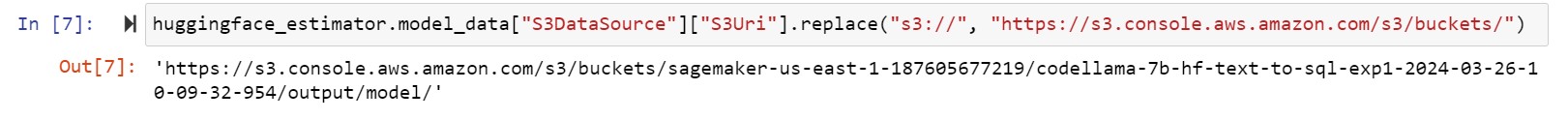


We can now start our training job, with the **.fit()** method passing our S3 path to the training script.

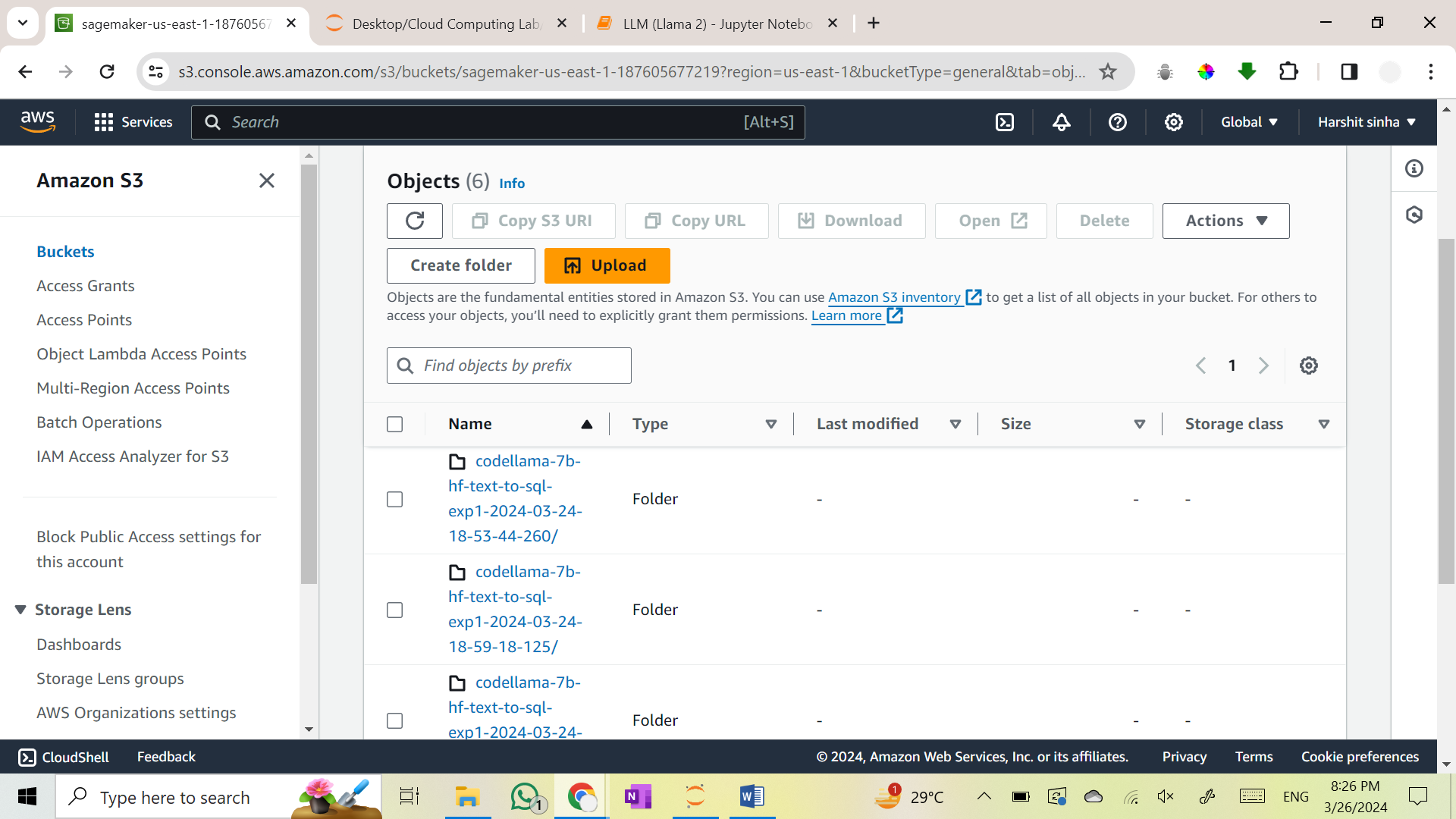


In our case for CodeLlama 7B, the SageMaker training job took 3478 seconds, which is about 0.96611 hour (~ 1hour). The ml.g5.2xlarge instance we used costs $1.15 per hour for on-demand usage. As a result, the total cost for training our fine-tuned Code LLama model was only ~$1.76.

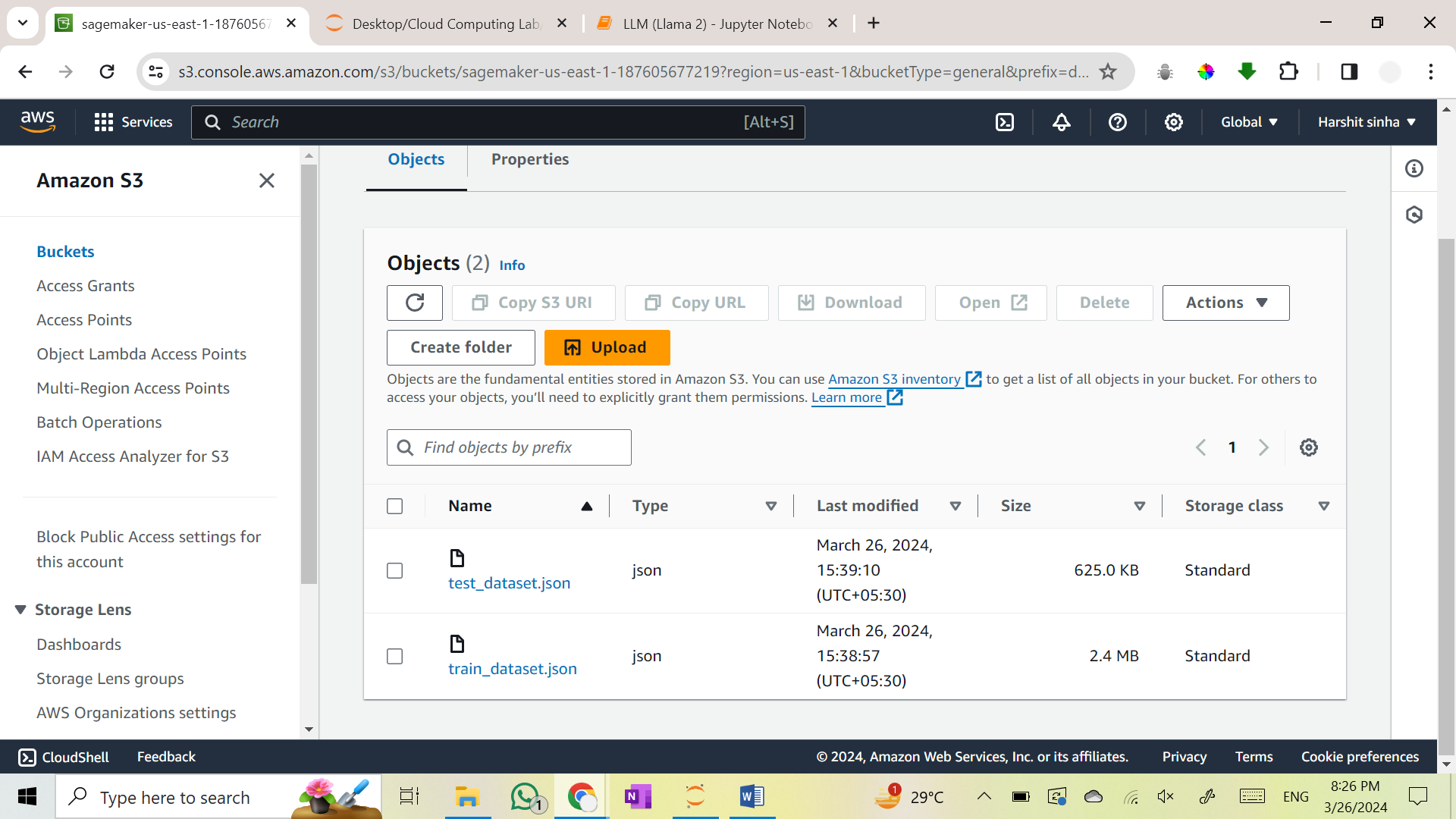
Now make sure SageMaker has successfully uploaded the model to S3. We can use the model\_data property of the estimator to get the S3 path to the model. Since we used merge\_weights=True and disable\_output\_compression=True the model is stored as raw files in the S3 bucket.



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

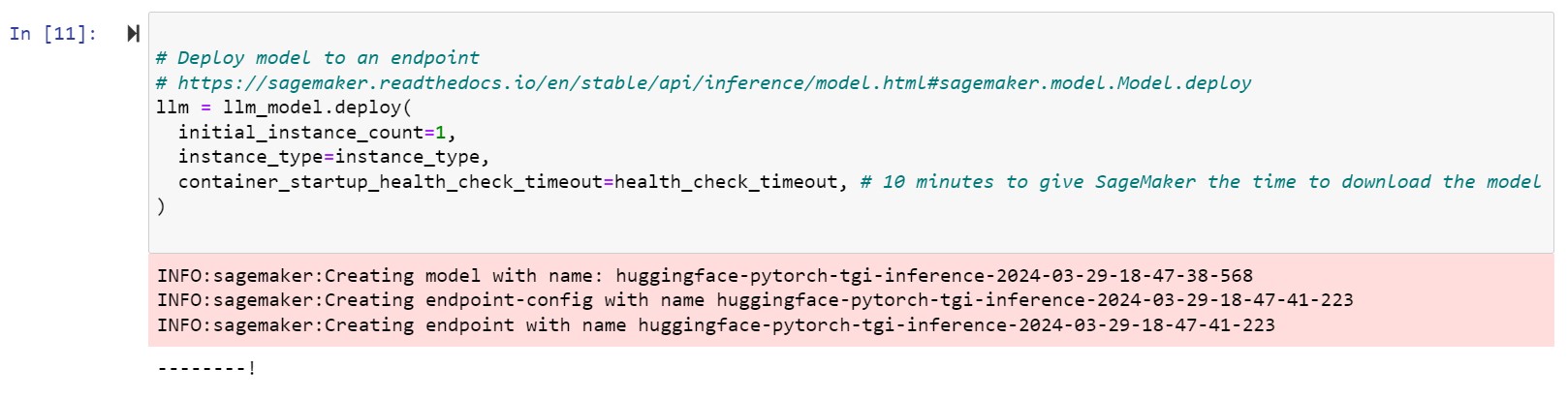


And also the training & testing dataset JSON files.

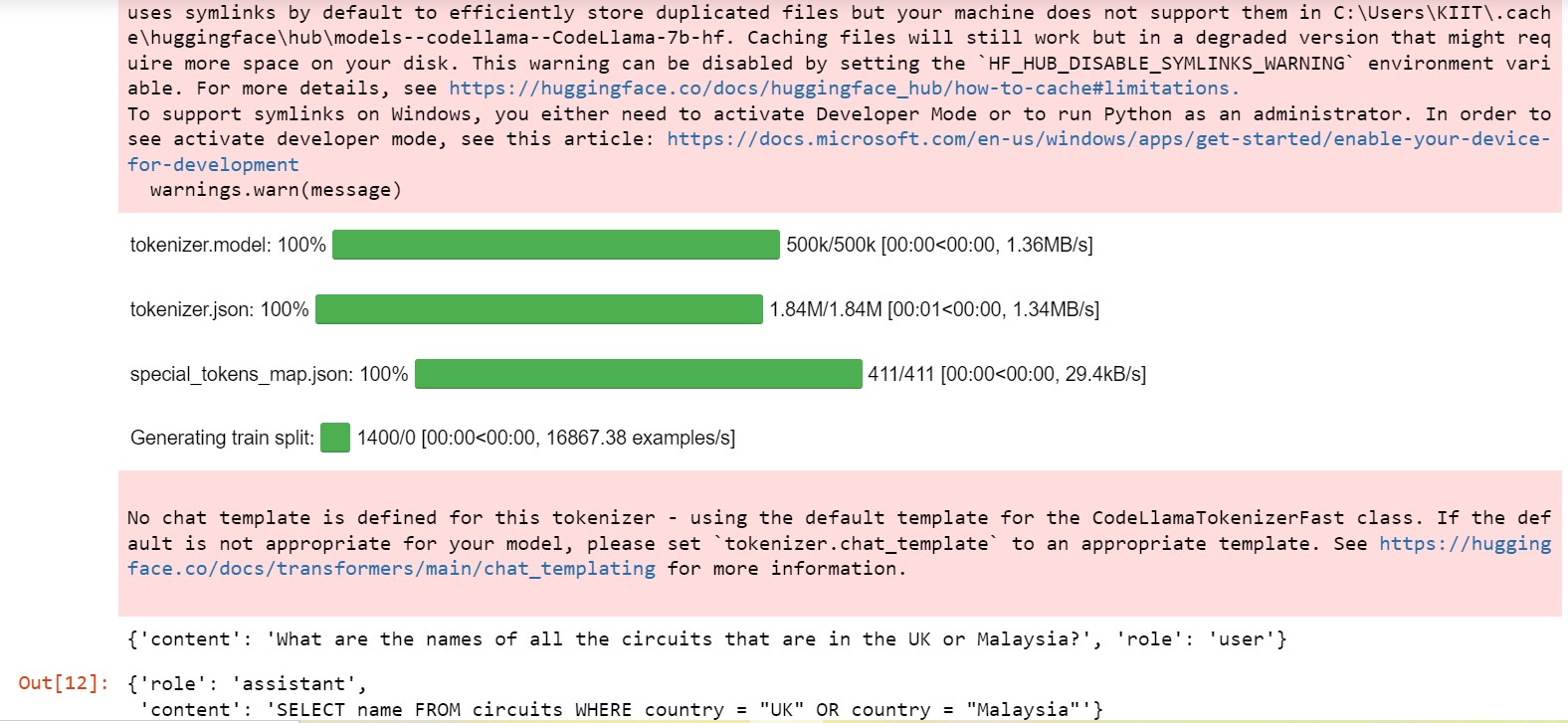


4.4 Deploy & Evaluate LLM on Amazon SageMaker

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.

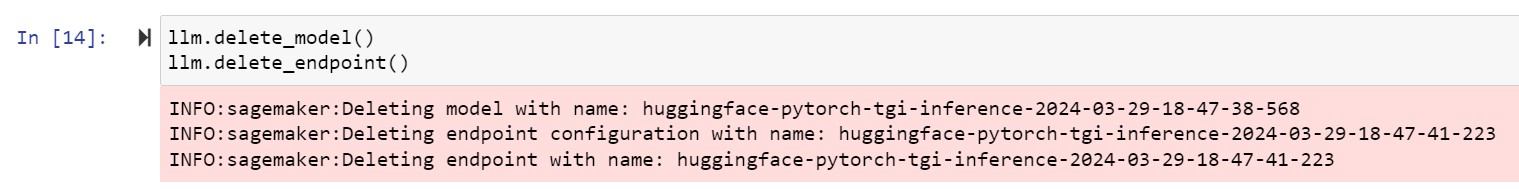








Now we will delete the end points as these adds on to the AWS bill.



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.

