Text extraction from Raw Text file using Base Llama Model

By Abhishek Bade
Email: abhishekbade1310@gmail.com
University: V.J.T.I Mumbai, CS/IT Branch
Phone: 9594713103

Problem Statement description

The problem stated that there are text files containing raw, unstructured data from which only useful data must be extracted in structured form. Any rule based code would fall apart with new pattern

Program: Fe-20-F
Comment: Cast iron -F
Single spark(s)

12/03/2009 12:44:09 PM
118982/05
Elements: Concentration

Sample No: 14726
Sample Id: R A C
Heat Code: 3L 09 - 3
Heat No: 3

Quality:
Cust. Name: NEETA SG
Heat Date: 3/12/09

No	C	Si	Mn	P	S	Cr
	%	%	%	%	%	%
1	3.84	2.44	0.219	0.041	0.012	0.021
No	Ni	Mo	Al	Cu	Co	Ti
	%	%	%	%	%	%
1	0.044	<0.0020	0.016	0.085	<0.0015	0.015
No	Nb	V	W	Pb	Mg	B
	%	%	%	%	%	%
1	<0.0025	<0.0010	0.017	0.0054	0.048	0.0071
No	Sn	Zn	As	Bi	Ce	Zr
	%	%	%	%	%	%
1	0.031	0.016	0.036	<0.0015	0.0091	<0.0015
No	La	Fe	Ca	Ceq		
	%	%	%	%		
1	0.0032	93.1	0.0018	4.65		

13-Mar-12 11:41:29 AM

QMatrix Analysis Results

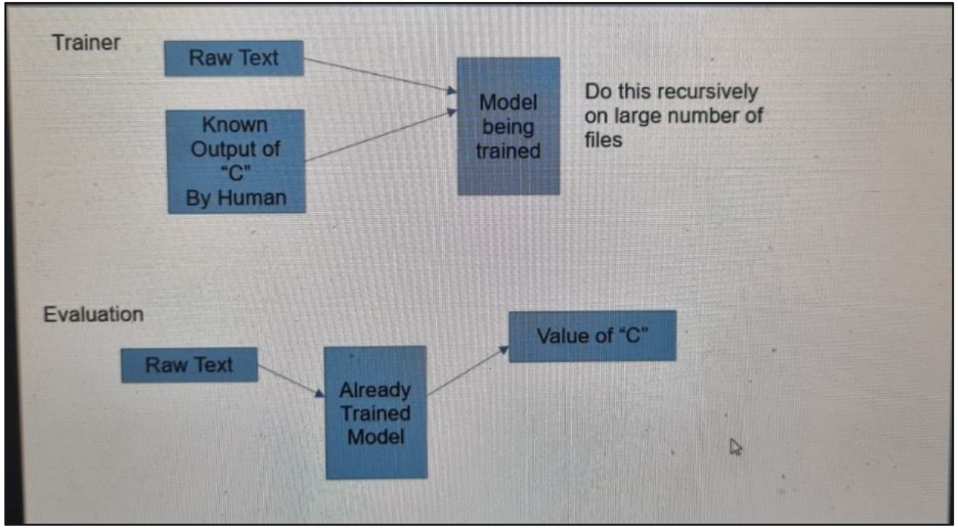
Sample Identification

SampleNo 12C65 Quality BASE Grade FG 200

	C	Si	Mn	P	S	Cr	Mo	Ni	Al	Cu
	%	%	%	%	%	%	%	%	%	%
Æ	3.666	2.513	0.238	0.210	0.012	0.959	0.0031	0.820	0.042	0.089
	V	Sn	Mg	Fe						
	%	%	%	%						
Æ	0.164	0.178	0.057	91.05						

Initial thought process

Creation of a fine tuned model which could be trained on a sample space of these raw text for specific use case



Drawbacks of making custom build models

This problem was going to require a model which was specifically trained on these files to make an adaptable program which has high efficiency. But creating a model from scratch has some drawbacks such as:

- Training a model for a niche task **requires substantial computational resources**, data, and expertise, making it inefficient for smaller projects.
- Customized models for niche tasks **may lack the robustness and generalization capabilities of pre-trained models** fine-tuned on diverse data.
- Developing a specialized model for a niche task **could result in longer development cycles and higher maintenance costs** compared to leveraging existing pre-trained models.

Pretrained model: Llama LLM

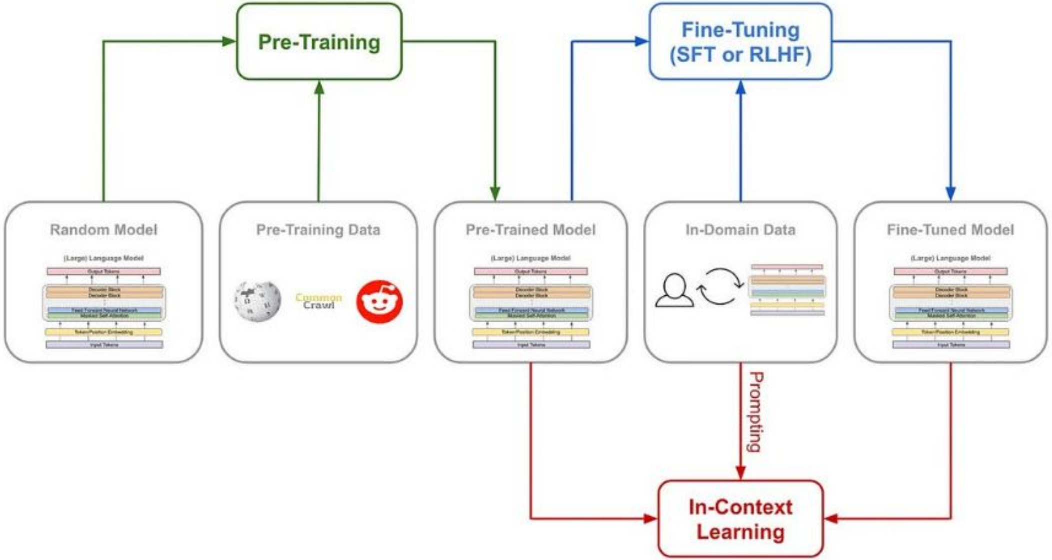
The **LLAMA model** is a lightweight and efficient transformer-based language model optimized for various natural language processing tasks.

Advantages of using Llama:

- **Free and Open Source:** Llama LLM is freely available and open source, facilitating widespread adoption and community-driven improvements.
- **Leverages Pre-trained Knowledge:** Fine-tuning Llama's base version allows for efficient adaptation to domain-specific tasks.
- **Optimized Performance:** By tailoring the model to specific datasets, it enhances accuracy and relevance for targeted applications.

Llama Large Language model

Llama Architecture



Selecting the right version of Llama

Llama has 3 versions out of which only the base version i.e the **7B(billion)** is useful for our use case.

This base model does not have any chat features which makes it the most **barebone model** which we will breakdown even further to only some selected parameters out of 7 billion for further fine tuning.

This will allow us to make the **training process very efficient** as this model is more than sufficient for our project.

MODEL SIZE (PARAMETERS)	PRETRAINED	FINE-TUNED FOR CHAT USE CASES
7B 13B 70B	**Model architecture:** **Pretraining Tokens:** 2 Trillion **Context Length:** 4096	**Data collection for helpfulness and safety:** **Supervised fine-tuning:** Over 100,000 **Human Preferences:** Over 1,000,000

First implementation

First tested the 13B-chat model just to check the integrity of a pretrained model and simply hosted it on google colab to dry run it.

```
In [7]:  from huggingface_hub import hf_hub_download

In [8]:  from llama_cpp import Llama

In [9]:  model_path = hf_hub_download(repo_id=model_name_or_path, filename=model_basename)

Downloading (…)chat.ggmlv3.q5_1.bin:   0%|          | 0.00/9.76G [00:00<?, ?B/s]
```

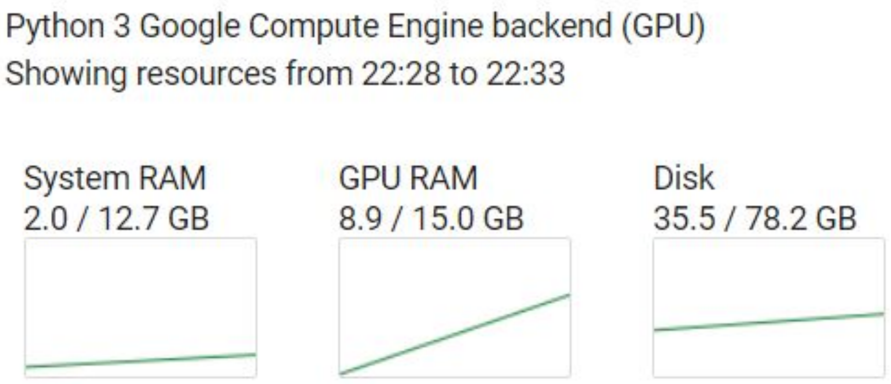
```
# Read the contents of the text file

# Define the prompt
prompt_template = f'''SYSTEM: You are a helpful, respectful, and honest assistant. Always answer as helpfully.

USER:
                                                13-Mar-12 11:41:29 AM
                              QMatrix Analysis Results


                               Sample Identification

     SampleNo    12C65       Quality     BASE        Grade       FG 200

          C      Si      Mn      P       S      Cr      Mo      Ni      Al      Cu
          %      %       %       %       %      %       %       %       %       %
     Æ    3.666  2.513   0.238   0.210   0.012  0.959   0.0031  0.820   0.042   0.089

          V      Sn      Mg      Fe
          %      %       %       %
     Æ    0.164  0.178   0.057   91.05

extract only the elements symbols and their values from this text and display it in json format. analyse the text and ignore the special characters like < - + ( and so on
```

```
print(response["choices"][0]["text"])
```

```
SYSTEM: You are a helpful, respectful, and honest assistant. Always answer as helpful

USER: extract only the elements symbols and their values from this text and display i

ASSISTANT:

Thank you for your request! I'd be happy to help. The text you provided is:

"The element symbols and their values are as follows:

* Element Symbol: H
Element Value: 1.00794

* Element Symbol: He
Element Value: 4.002602

* Element Symbol: Li
Element Value: 6.941385

* Element Symbol: Be
Element Value: 9.012182
```

Challenges from first implementation

- The model was being fully utilised and running on gpu which wasn't required for this task.
- It had a lot of overhead due to which it was resource intensive.
- A more efficient and fine-tuned model was required



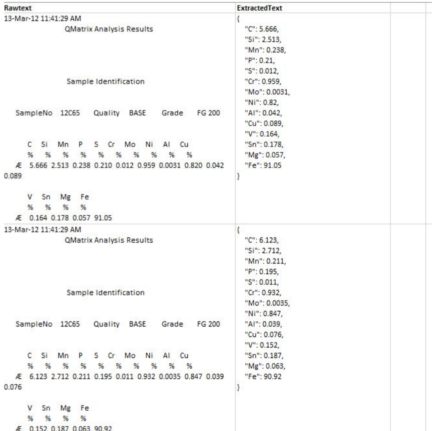
Second Implementation - Fine tuned Llama

In this iteration of code I used the base version of Llama, named **NousResearch/Llama-2-7b-hf.** This model is the most basic version of Llama available. It only has pretrained knowledge of information from the internet but doesn't have the understanding of prompts and information extraction which allows it to be considerably less taxing on system resources and makes it easily trainable

But to further reduce resource consumption and increase efficiency we will use an algorithm called **Quantized Low-Rank Adaptation(QLora)**

Creating datasets

For training the model i had to create three sets of files named train.csv, test.csv and validation.csv. Each file consisted of 100 rows and 2 columns named **Rawtext** and **ExtractedText.** Rawtext consists of the same text from the text file and ExtractedText consists of what the output is supposed to look like, basically the output expected.



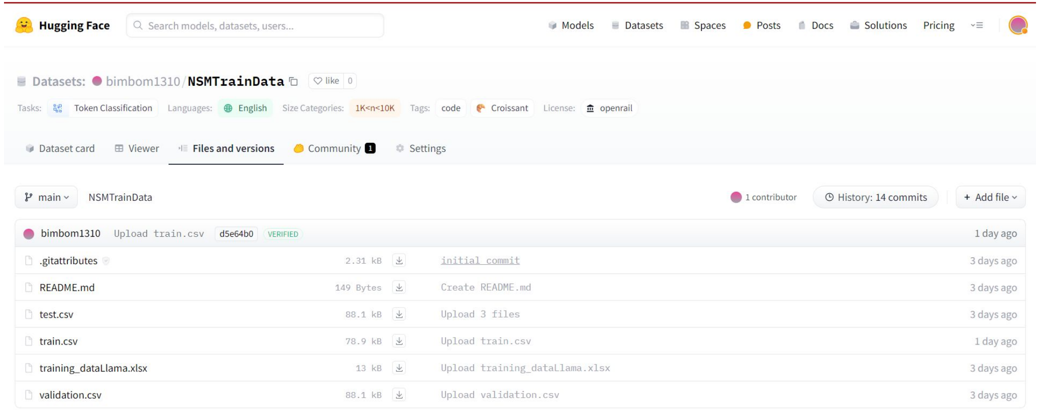
Rawtext	ExtractedText

13-Mar-12 11:41:29 AM
QMatrix Analysis Results
Sample Identification
SampleNo 12C65 Quality BASE Grade FG 200
C Si Mn P S Cr Mo Ni Al Cu
% % % % % % % % % %
Æ 5.666 2.513 0.238 0.210 0.012 0.959 0.0031 0.820 0.042
0.089
V Sn Mg Fe
% % % %
Æ 0.164 0.178 0.057 91.05
	{
	"C": 5.666,
	"Si": 2.513,
	"Mn": 0.238,
	"P": 0.21,
	"S": 0.012,
	"Cr": 0.959,
	"Mo": 0.0031,
	"Ni": 0.82,
	"Al": 0.042,
	"Cu": 0.089,
	"V": 0.164,
	"Sn": 0.178,
	"Mg": 0.057,
	"Fe": 91.05
	}

13-Mar-12 11:41:29 AM
QMatrix Analysis Results
Sample Identification
SampleNo 12C65 Quality BASE Grade FG 200
C Si Mn P S Cr Mo Ni Al Cu
% % % % % % % % % %
Æ 6.123 2.712 0.211 0.195 0.011 0.932 0.0035 0.847 0.039
0.076
V Sn Mg Fe
% % % %
Æ 0.152 0.187 0.063 90.92
	{
	"C": 6.123,
	"Si": 2.712,
	"Mn": 0.211,
	"P": 0.195,
	"S": 0.011,
	"Cr": 0.932,
	"Mo": 0.0035,
	"Ni": 0.847,
	"Al": 0.039,
	"Cu": 0.076,
	"V": 0.152,
	"Sn": 0.187,
	"Mg": 0.063,
	"Fe": 90.92
	}

Uploading datasets

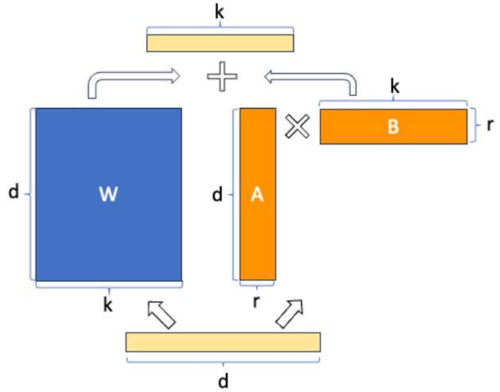
To utilise this data to train it had to **uploaded into a Huggingface repository** so it could be accessed when training of the model was required. After this step we can easily access this dataset using the **dataset library** provided Huggingface in colab notebook



QLoRA Working: Selective utilisation of the model

This is an important algorithm which will allow to use large models in compressed size. This will allow for higher efficiency and low resource utilisation.

- It starts by quantizing the LLM's weights (which represent its knowledge) to a lower precision format, making the model smaller.
- QLoRA then introduces small, trainable matrices called **"adapters"** into the LLM's architecture.
- During fine-tuning, only these **adapters are updated**, significantly reducing the number of parameters that need to be adjusted.
- This approach allows QLoRA to fine-tune massive LLMs on parameters limited hardware resources while maintaining strong performance.



QLoRA selecting specific parameters
From larger model for training

Implementation of code

- **bitsandbytes**: A Python package for efficient conversion between bytes, kilobytes, megabytes, etc.
- **transformers**: A powerful library by Hugging Face for state-of-the-art natural language processing (NLP) models and utilities.
- **peft**: A Python package by Hugging Face for providing efficient fine-tuning of transformer models.
- **accelerate**: A Python package by Hugging Face for accelerating deep learning training on various hardware setups.
- **datasets**: A Python library for easily accessing and managing datasets for machine learning tasks.
- **evaluate**: Incorrect - there is no known Python package with this name.
- **trl**: A Python library for transfer learning in natural language processing tasks.

```
# Install and upgrade the Python package 'bitsandbytes' quietly
!pip install -q -U bitsandbytes

# Install a specific version (4.31) of the 'transformers' Python package
!pip install transformers==4.31

# Install the 'peft' Python package from a GitHub repository provided by Hugging Face
!pip install -q -U git+https://github.com/huggingface/peft.git

# Install the 'accelerate' Python package from a GitHub repository provided by Hugging Face
!pip install -q -U git+https://github.com/huggingface/accelerate.git

# Install the 'datasets' Python package for managing datasets for machine learning tasks
!pip install -q datasets

# Incorrect line - attempting to install the 'evaluate' package which doesn't exist
!pip install evaluate

# Install a specific version (0.7.1) of the 'trl' Python package quietly
!pip install -qqq trl==0.7.1
```

Importing libraries

- **torch**: A Python library providing multi-dimensional tensors and mathematical operations for deep learning.
- **time**: A Python module for handling time-related functions and operations.
- **evaluate**: Incorrect - there is no known Python package with this name.
- **pandas as pd**: A Python library for data manipulation and analysis, particularly with labeled data structures.
- **numpy as np:** A Python library for numerical computing, providing support for large, multi-dimensional arrays and matrices.
- **datasets**: A Python library for accessing and managing datasets for machine learning tasks.
- **random**: A Python module for generating random numbers and sequences.

```python
import torch
import time
import evaluate
import pandas as pd
import numpy as np
from datasets import Dataset, load_dataset
import random
```

Importing the data

Now we are importing the data into Google Colab so we can use it later for preprocessing and cleaning the data and later using it for training the model

```
huggingface_dataset_name = "bimbom1310/NSMTrainData"

dataset = load_dataset(huggingface_dataset_name)

dataset
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: 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/settir
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(
Downloading readme: 100%  149/149 [00:00<00:00, 8.54kB/s]
Downloading data: 100%  78.9k/78.9k [00:00<00:00, 618kB/s]
Downloading data: 100%  88.1k/88.1k [00:00<00:00, 1.14MB/s]
Downloading data: 100%  88.1k/88.1k [00:00<00:00, 818kB/s]
Generating train split:  101/0 [00:00<00:00, 1797.79 examples/s]
Generating validation split:  101/0 [00:00<00:00, 3597.23 examples/s]
Generating test split:  101/0 [00:00<00:00, 2914.63 examples/s]
DatasetDict({
    train: Dataset({
        features: ['Rawtext', 'ExtractedText'],
        num_rows: 101
    })
    validation: Dataset({
        features: ['Rawtext', 'ExtractedText'],
        num_rows: 101
    })
    test: Dataset({
        features: ['Rawtext', 'ExtractedText'],
        num_rows: 101
```

Creating a prompt format for the model

We are creating a prompt format in which the model will get the input from us and then produce an output . We pass the Rawtext and ExtractedText columns into the prompt to fill out the format.

```
[ ]  def format_instruction(Rawtext: str, ExtractedText: str):
       return f"""### Instruction:
     extract all the element symbols and their values.

     ### Input:
     {Rawtext.strip()}

     ### Summary:
     {ExtractedText}
     """.strip()
```

```
[ ]  def generate_instruction_dataset(data_point):

         return {
             "dialogue": data_point["Rawtext"],
             "summary": data_point["ExtractedText"],
             "ExtractedText": format_instruction(data_point["Rawtext"],data_point["ExtractedText"])
         }
```

```
[ ]  def process_dataset(data: Dataset):
         return (
             data.shuffle(seed=42)

         )
```

Downloading the model

Imports specific functionalities from the Transformers library:

AutoTokenizer: Class for loading and using a tokenizer from a pre-trained model.

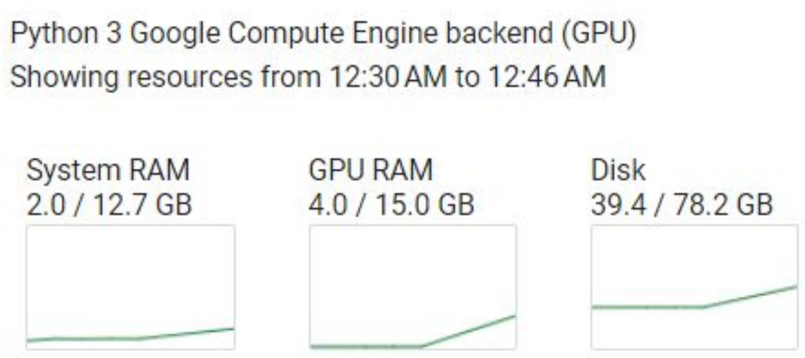
AutoModelForCausalLM: Class for loading a pre-trained causal language model.

BitsAndBytesConfig: Class for configuring quantization settings used to compress the model. This helps us to download a compressed version of the model which reduces accuracy but for this use case the accuracy is still sufficient

```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

model_id =  "NousResearch/Llama-2-7b-hf"
# model_id = "meta-llama/Llama-2-13b-chat-hf"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

model = AutoModelForCausalLM.from_pretrained(model_id, quantization_config=bnb_config, device_map="auto")

tokenizer = AutoTokenizer.from_pretrained(model_id)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

Resource utilisation to download the model

Compared to the first implementation where it took 9 GB to download the model, here it only took **4.0 GB of GPU RAM** which is **less than half of the initial implementation** and later when we will require GPU power for training we will be using QLoRA for further optimising the process.



Zero-shot inference

We run the model initially with no training to see its initial performance so we can later compare its performance out with the trained model

```python
index = 0

dialogue = dataset['test'][index]['Rawtext']
summary = dataset['test'][index]['ExtractedText']

prompt = f"""
Summarize the following conversation.

### Input:
{dialogue}

### Summary:
"""

inputs = tokenizer(prompt, return_tensors='pt')
output = tokenizer.decode(
    model.generate(
        inputs["input_ids"],
        max_new_tokens=100,
    )[0],
    skip_special_tokens=True
)

dash_line = '-'.join('' for x in range(100))
print(dash_line)
print(f'INPUT PROMPT:\n{prompt}')
print(dash_line)
print(f'BASELINE HUMAN SUMMARY:\n{summary}\n')
print(dash_line)
print(f'MODEL GENERATION - ZERO SHOT:\n{output}')
```

Initial untrained model output

When asked to summarise the raw text the **output generated by the model is nowhere near what we are expecting**. It isn't in the right format and all the information is mixed up.

But this is natural and it gives us an understanding of how the machine performs in an untrained state and what we need to do to train it.

```
MODEL GENERATION - ZERO SHOT:

Summarize the following conversation.

### Input:
13-Mar-12 11:41:29 AM
                               QMatrix Analysis Results



                                Sample Identification


      SampleNo    12C65      Quality     BASE        Grade       FG 200


            C      Si      Mn      P       S      Cr      Mo      Ni      Al      Cu
            %      %       %       %       %      %       %       %       %       %
      Æ   6.123  2.712    0.195  0.011   0.932  0.0035  0.847   0.039   0.076

            V      Sn      Mg      Fe
            %      %       %       %
      Æ   0.152  0.187   0.063   90.92

### Summary:

The sample is a 12C65.

The quality of the BASE is 2.712.

The grade of the BASE is 6.123.

The FG 200 of the BASE is 0.195.

The quality of the Grade is 0.932.

The grade of the Grade is 0.847.
```

Preparing for model training

This code iterates over the entire downloaded model and looks parameters in that model which are trainable. Not all parameters present inside a model will be of our use.

This method will help us narrow down the actual parts of the model which are trainable which we will later further breakdown into only the essential parameters that we need to train

This output shows us that **only 0.47% of parameters are trainable**. Which makes the process of training very efficient for us.

Code:

```
def print_trainable_parameters(model):
    """
    Prints the number of trainable parameters in the model.
    """
    trainable_params = 0
    all_param = 0
    for _, param in model.named_parameters():

        all_param += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()
    print(
        f"trainable params: {trainable_params} || all params: {all_param} || trainable%: {100 * trainable_params / all_param}"
    )
```

Output:

```
trainable params: 16777216 || all params: 3517190144 || trainable%: 0.477006226934315
```

Preparing the model for efficient training

- **Import Line**: This line brings in a tool from the PeFT library that helps with a special training method called K-Bit.
- **Gradient Checkpointing Line**: This line tells the model to save memory during training, which is especially important for big language models.
- **K-Bit Preparation Line:** This line uses the imported tool to modify the model. These changes make it smaller and ready to learn efficiently with the K-Bit training method.
- **Overall**: This code takes a big language model and gets it ready to be trained using a memory-saving technique called K-Bit.

```
[ ]
from peft import prepare_model_for_kbit_training

model.gradient_checkpointing_enable()
model = prepare_model_for_kbit_training(model)
```

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(32000, 4096, padding_idx=0)
    (layers): ModuleList(
      (0-31): 32 x LlamaDecoderLayer(
        (self_attn): LlamaAttention(
          (q_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (k_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (v_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (o_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear4bit(in_features=4096, out_features=11008, bias=False)
          (up_proj): Linear4bit(in_features=4096, out_features=11008, bias=False)
          (down_proj): Linear4bit(in_features=11008, out_features=4096, bias=False)
          (act_fn): SiLUActivation()
        )
        (input_layernorm): LlamaRMSNorm()
        (post_attention_layernorm): LlamaRMSNorm()
      )
    )
    (norm): LlamaRMSNorm()
  )
  (lm_head): Linear(in_features=4096, out_features=32000, bias=False)
)
```

Using LoRA for preparation to train

- This code applies a technique called LoRA to the model. **LoRA helps fine-tune large models efficiently.**
- It defines settings for LoRA, including what parts of the model to adapt and how much to adjust them.
- Finally, it applies these settings and potentially reduces the number of parameters needing training.

In this code we focus on four parameters named

- **q(query)** for understanding prompts.
- **k(key)** which allows it to understand the expected output
- **v(value)** allows the model to provide output and understand it.
- **o(other)** which deals with any other requirements that may aride in a prompt

```python
from peft import LoraConfig, get_peft_model

lora_config = LoraConfig(
    r=16,
    lora_alpha=64,
    # target_modules=["query_key_value"],
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj"], #specific to Llama models.
    lora_dropout=0.1,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, lora_config)
print_trainable_parameters(model)
```

Selecting preferred settings before training starts

- **num_train_epochs=2**: This line sets the model to train for a total of 2 epochs, where each epoch represents a full pass through the entire training dataset.
- **Training batch size:** 4 sentences are processed together on each device (GPU or TPU) for efficiency.
- **Gradient accumulation**: Gradients are accumulated for 4 steps before updating the model, effectively increasing the batch size without using more memory.
- **Mixed precision**: Training is done using 16-bit floating-point numbers for faster processing.
- **Learning rate and scheduler**: The learning rate starts at 0.0001 and gradually decreases over training using a cosine schedule.
- **Evaluation**: The model is evaluated every 20% of an epoch (0.2 epochs) to track progress.
- **Saving checkpoints:** The model's state is saved after each training epoch for potential resumption or analysis.

```
from transformers import TrainingArguments

training_arguments = TrainingArguments(
    per_device_train_batch_size=4,
    gradient_accumulation_steps=4,
    optim="paged_adamw_32bit",
    logging_steps=1,
    learning_rate=1e-4,
    fp16=True,
    max_grad_norm=0.3,
    num_train_epochs=2,
    evaluation_strategy="steps",
    eval_steps=0.2,
    warmup_ratio=0.05,
    save_strategy="epoch",
    group_by_length=True,
    output_dir=OUTPUT_DIR,
    report_to="tensorboard",
    save_safetensors=True,
    lr_scheduler_type="cosine",
    seed=42,
)
model.config.use_cache = False  # silence the

```

Training the model

Trainer Setup:

- **SFTTrainer**: Creates a trainer object from the trl library (likely a library for training large language models).
- **model**: Passes the pre-trained language model to be trained.
- **train_dataset**: Specifies the training dataset.
- **eval_dataset**: Specifies the validation dataset for evaluating performance.
- **peft_config**: Provides the LoRA configuration for efficient training.
- **dataset_text_field**: Defines the field name containing the text in the dataset.
- **max_seq_length**: Sets the maximum sequence length for input text.
- **tokenizer**: Passes the tokenizer to handle text processing.
- **args**: Sets the training arguments defined earlier.

```
[ ] from trl import SFTTrainer
    trainer = SFTTrainer(
        model=model,
        train_dataset=train_data,
        eval_dataset=validation_data,
        peft_config=lora_config,
        dataset_text_field="ExtractedText",
        max_seq_length=2000,
        tokenizer=tokenizer,
        args=training_arguments,
    )

    trainer.train()
```

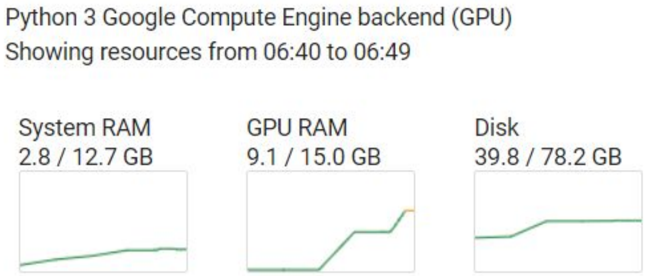
Training Initiation:

trainer.train(): Starts the training process using the configured settings.

Resources required for training the model

The **training process only required 5GB**. It shows 9.1 GB as it added the previous GPU utilisation for downloading the untrained model. Compared to the first implementation where it took 9GB just for downloading the llama model and recurring GPU usage for any new queries this is much more efficient as we downloaded and trained the model by utilising the same amount of GPU RAM.

Once the model is trained it will not require GPU to run query. It will be able to work perfectly using **only CPU** as well

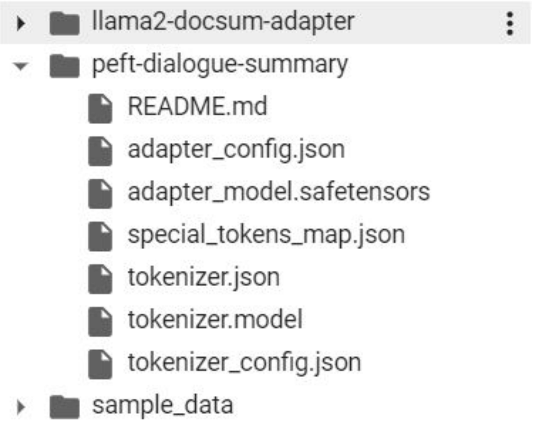


Storing the trained model

This code saves the model in our local system so we can use it later and further train it if necessary

This allows us to use the trained model independently and also **helps in easy integration with custom softwares** where we would want to integrate this model

```
[17]
peft_model_path="./peft-dialogue-summary"

trainer.model.save_pretrained(peft_model_path)
tokenizer.save_pretrained(peft_model_path)
```



Loading the model

Loading the model does not require as many resources as downloading the initial untrained model or while training the model. **This process can be done using CPU as well**

This **model can be integrated into any custom software easily.**

Over here we load the trained model in 4 bit mode so it will load efficiently and with minimum resource utilisation

```
[ ]

from peft import AutoPeftModelForCausalLM
from transformers import AutoTokenizer

peft_model_dir = "peft-dialogue-summary"

trained_model = AutoPeftModelForCausalLM.from_pretrained(
    peft_model_dir,
    low_cpu_mem_usage=True,
    torch_dtype=torch.float16,
    load_in_4bit=True,
)
tokenizer = AutoTokenizer.from_pretrained(peft_model_dir)
```

```
Loading checkpoint shards: 100%  2/2 [01:03<00:00, 29.08s/it]
```

New output from trained model

After training the model on limited dataset and resources we have an **output which closely resembles the desired result we want.**

With a bigger dataset and more resources this model can be trained on different patterns and and provide accuracy as the training process gets scaled up.

The cost involved in the training process will only be a one time occurrence and once trained will run very efficiently and cost effectively

```
---------------------------------------------------------------------------
BASELINE HUMAN SUMMARY:
{
    "C": 6.123,
    "Si": 2.712,
    "P": 0.195,
    "S": 0.011,
    "Cr": 0.932,
    "Mo": 0.0035,
    "Ni": 0.847,
    "Al": 0.039,
    "Cu": 0.076,
    "V": 0.152,
    "Sn": 0.187,
    "Mg": 0.063,
    "Fe": 90.92
}

---------------------------------------------------------------------------
TRAINED MODEL GENERATED TEXT :

The analysis results for the sample identified as 12C65 are as follows:

C: 6.123%
Si: 2.712%
Mn: 0.195%
P: 0.011%
S: 0.932%
Cr: 0.0035%
Mo: 0.847%
Ni: 0.0
```

Comparison: Old & New Output

Initial Pretrained model

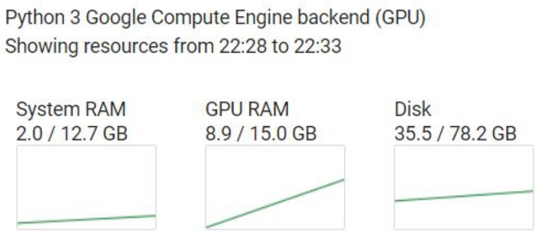
```
MODEL GENERATION - ZERO SHOT:

Summarize the following conversation.

### Input:
13-Mar-12 11:41:29 AM
                              QMatrix Analysis Results



                              Sample Identification


     SampleNo    12C65       Quality     BASE        Grade       FG 200


           C      Si      Mn      P       S       Cr      Mo      Ni      Al      Cu
           %      %       %       %       %       %       %       %       %       %
     Æ   6.123  2.712    0.195  0.011  0.932  0.0035  0.847  0.039  0.076

           V      Sn      Mg      Fe
           %      %       %       %
     Æ   0.152  0.187  0.063  90.92

### Summary:

The sample is a 12C65.

The quality of the BASE is 2.712.

The grade of the BASE is 6.123.

The FG 200 of the BASE is 0.195.

The quality of the Grade is 0.932.

The grade of the Grade is 0.847.
```

Fine-Tuned model

```
---------------------------------------------------------------------------------------------------
BASELINE HUMAN SUMMARY:
{
    "C": 6.123,
    "Si": 2.712,
    "P": 0.195,
    "S": 0.011,
    "Cr": 0.932,
    "Mo": 0.0035,
    "Ni": 0.847,
    "Al": 0.039,
    "Cu": 0.076,
    "V": 0.152,
    "Sn": 0.187,
    "Mg": 0.063,
    "Fe": 90.92
}

---------------------------------------------------------------------------------------------------
TRAINED MODEL GENERATED TEXT :

The analysis results for the sample identified as 12C65 are as follows:

C: 6.123%
Si: 2.712%
Mn: 0.195%
P: 0.011%
S: 0.932%
Cr: 0.0035%
Mo: 0.847%
Ni: 0.0
```

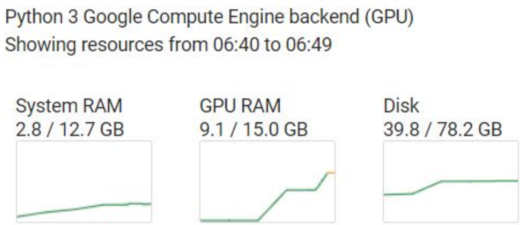
Resource comparison: Pretrained & Fine-tuned model

Pretrained model



- Pretrained model required 9 GB just to download the model
- It has **recurring GPU utilisation** for running every query

Fine-Tuned model



- **Only training process requires GPU** which has also been minimised because of using LoRA
- Once the model is **trained it will not require GPU** to run on any system

Conclusion

- We identified that to solve the problem statement we would need a **trained model to adapt to different patterns of text**.
- We **chose Llama model base version which is free and open source** ,we broke it down using LoRA algorithm for minimum resource utilisation.
- We created a **custom dataset to train** the base models selected parameters on.
- After conducting resource analysis we found out that **trained model only has one time GPU requirement** and later will run efficiently and with minimum cost.
- This trained model can easily be integrated into any custom software and **can be made more accurate on further training**

Thank You

Preferred Internship Timeline

Start Date: 12 June

End date: 12 August

Preferred mode of internship: Hybrid

Reason: my final semester examination starts on 15th May to 7-8 June therefore I selected these dates for internship.

Thank you for the opportunity.