# Vehicle Servicing Chatbot

Internship Report

7/18/2024

Tata Motors Limited

DRAKSHA

Under the guidance of Sanket Sir

# Table of Contents

# Introduction

In today's fast-paced world, vehicle maintenance and servicing are essential for ensuring safety, performance, and longevity. However, accessing reliable and timely information on vehicle upkeep can often be challenging for many users. To address this need, we have developed an innovative vehicle chatbot that leverages the capabilities of LLaMA2, a state-of-the-art language model.

This vehicle chatbot is designed to assist users with a wide range of queries related to vehicle maintenance and servicing. By utilizing advanced natural language processing and machine learning techniques, our chatbot provides accurate, context-aware responses to user inquiries, making it easier for vehicle owners to maintain their vehicles efficiently.

The development of this chatbot involved integrating LLaMA2 due to its robust performance and adaptability, ensuring that it can handle the complex and varied nature of vehicle-related queries. From routine maintenance schedules to troubleshooting common issues, the chatbot serves as a comprehensive resource for vehicle owners, offering guidance and support at their fingertips.

In this report, we will delve into the technical aspects of the chatbot's development, including the architecture, training process, and implementation details. Additionally, we will highlight the key features and functionalities that make this chatbot a valuable tool for vehicle maintenance and servicing.

# Model

Llama2 is a Chat-bot developed by Meta AI also that is known as Large Language Model Meta AI. It uses Natural language processing(NLP) to work on human inputs and it generates text, answers complex questions, and can have natural and engaging conversations with users. It was pretrained on 2 trillion tokens of data from publicly available sources.

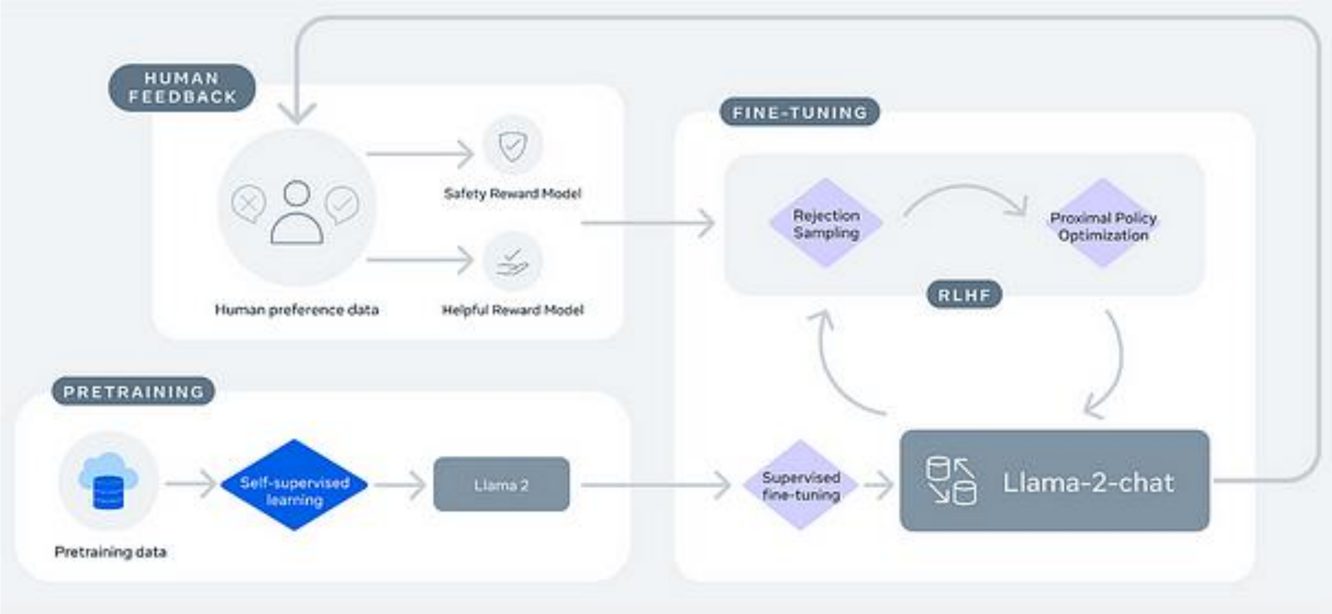
Llama2 is available in various parameter sizes: 7 billion (7B), 13 billion (13B), and 70 billion (70B). It includes both pretrained and fine-tuned variations. As an auto-regressive language model, Llama2 utilizes optimized transformer architecture. The fine-tuned (chat) versions are enhanced through supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF), aligning them with human preferences for helpfulness and safety.

**Why not Llama 3?**

Meta AI has already introduced Llama 3 which is more advanced and faster than its previous versions, but during the development of our vehicle chat-bot, LLaMA2 was chosen over LLaMA3 as LLaMA3 requires advanced configurations and additional computational power that exceed the capabilities of my current laptop. LLaMA2, on the other hand, is well-suited to our existing setup, ensuring stable and reliable performance. This choice allows us to focus on refining our chat-bot's functionality without the complexities and demands of the latest model.

**Why Llama 2 chat version?**

1. Fine-tuned for dialogue: The Llama 2 chat model is specifically fine-tuned for dialogue-driven use cases, making it more suitable for chat-bots and virtual agents.
2. Improved response generation: The chat model is trained to generate responses in the format expected by users, making it more effective at responding to user queries.
3. Enhanced safety: The chat model is trained with a focus on safety, avoiding toxic and hateful responses.
4. Better alignment with human preferences: The chat model is trained using reinforcement learning with human feedback, aligning it better with human preferences.
5. Fewer parameters: The chat model has fewer parameters than the regular Llama 2 model, making it more efficient and easier to deploy.




## How to get the model

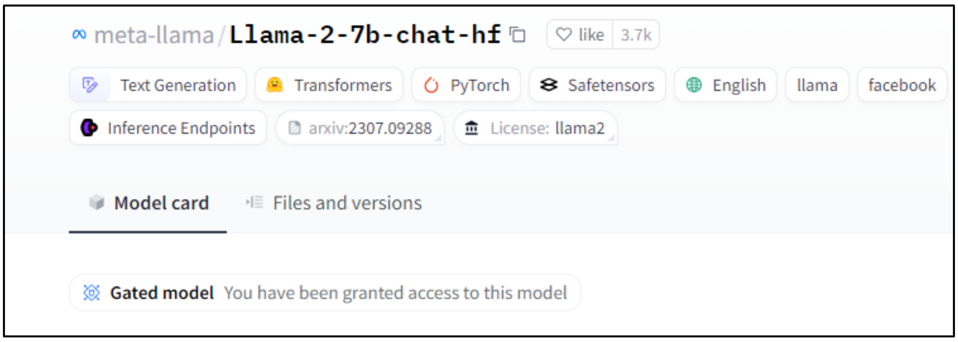
Step 1: Go to the website [https://huggingface.co/meta-llama/Llama-2-7b-chat-hf](https://huggingface.co/meta-llama/Llama-2-7b-chat-hf)

Step 2: It is a gated model, So request for access. It will direct you to Meta's request form.

Step 2:  Fill out Meta's form to request access. Make sure this is from the same email as you Huggingface account

Step 3: Confirm your email

Step 4: Your request might take some time to get approved. Once it is approved it will show



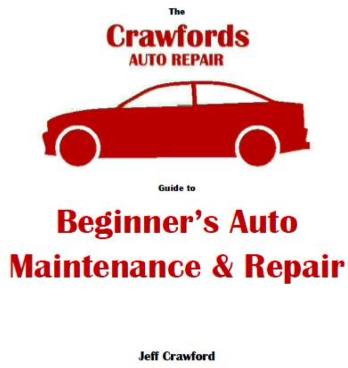

Step 5:  Go to Settings > Access Tokens

Step 6: Create a new Token (Type=fine-grained)

Step 7: Copy the Token. It will be used to load the model.

# Dataset

The data to fine tune this model has been taken from:
"The Crawford's Auto Repair Guide to Beginner's Auto Maintenance & Repair", By Jeff Crawford,
Contributing Author: Rex Kimball, Mirex Marketing




## Formatting Dataset

**Step 1:**

Several Question-Answer pair was created from this book manually and was stored in the vehicle_dataset.xlsx file. The name of the sheet is "Samples".

| | A | B |
|---|---|---|
| 1 | **Question** | **Answer** |
| 2 | How to check Tyre Pressure? | Follow these steps to check and correct the tire pressure: Remove the valve cap of each tire, Align the gauge up to the valve , Press the gauge onto the valve with firm direct pressure and then release. Now you will see the measuring stick get pushed out of the other side of the gauge (on a pen gauge). Fill the tire with air, recheck the pressure, then repeat as needed until the desired pressure is obtained. Replace the valve cap |
| 3 | Why engine oil is necessary? | The purpose of engine oil is to form a film of lubrication between all moving parts of an internal combustion engine to reduce friction and wear. Choosing the right engine oil for your vehicle and changing the oil during regularly-scheduled maintenance intervals will keep the engine running smoothly over time. |
| 4 | How to flush the cooling system? | Refer to the owner's manual for locations of each part of the cooling system. Open the radiator drain to drain the cooling system and collect the drainage into a container (you can take this to your local auto parts store for recycling). Remove the thermostat. The thermostat is typically on the engine side of the upper radiator hose, but in some cases its on the lower hose. Disconnect the lower radiator hose, force coolant through the thermostat housing (upper hose and engine block) with a garden hose until the water runs clear. Connect the garden hose to the radiator, flush water through the radiator until the water runs clear. Completely drain the cooling system of all the water. Reinstall the thermostat. Disconnect the reservoir and flush with the garden hose until water runs clear.Completely drain the reservoir. Connect the hoses and close the drains. Fill the cooling system with new coolant to manufacturer's specification and distilled water. Use a 50/50 mixture of coolant and distilled water or pre-mixed coolant. |

**Step 2:**

To fine tune the llama chat model, dataset in the vehicle_dataset.xlsx should be converted in the given format.

```
<s>[INST] Question [/INST] Answer </s>
```

- To convert vehicle_dataset in a proper format we run format_dataset.py file

```
Python format_dataset.py
```

```python
from openpyxl import load_workbook
import csv

# Load Excel workbook
wb = load_workbook("vehicle_dataset.xlsx")
```

```python
# Choose a specific sheet
sheet = wb["Samples"]

# Find the number of rows and columns(2) in the sheet
rows = sheet.max_row
columns = sheet.max_column
print(f"Dataset has {rows} rows and {columns} columns")

# List to store all rows in the llama2 format
formatted_data = []

with open("vehicle_dataset.csv", 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    # Iterate over rows and columns to extract data
    column_name="text"
    writer.writerow([column_name])
    for i in range(2, rows+1):
        prompt = "<s>[INST] "+sheet.cell(row=i, column=1).value+" [/INST] "+sheet.cell(row=i, column=2).value+" </s>"
        writer.writerow([prompt])
print("data formatting done")
```

- After running the command vehicle_dataset.csv is created which we will use to fine tune our llama chat model

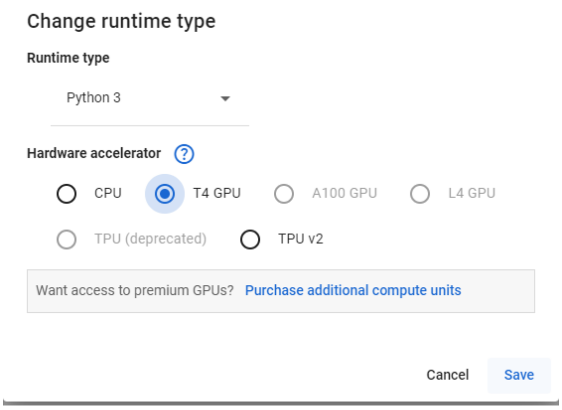
| | A |
|---|---|
| 1 | text |
| 2 | <s>[INST] How to check Tyre Pressure? [/INST] Follow these steps to check and correct the tire pressure: Remove the valve cap of each tire, Align the gauge up to the valve , Press the gauge onto the valve with firm direct pressure and then release. Now you will see the measuring stick get pushed out of the other side of the gauge (on a pen gauge). Fill the tire with air, recheck the pressure, then repeat as needed until the desired pressure is obtained. Replace the valve cap </s> |
| 3 | <s>[INST] Why engine oil is necessary? [/INST] The purpose of engine oil is to form a film of lubrication between all moving parts of an internal combustion engine to reduce friction and wear. Choosing the right engine oil for your vehicle and changing the oil during regularly-scheduled maintenance intervals will keep the engine running smoothly over time. </s> |
| 4 | <s>[INST] How to flush the cooling system? [/INST] Refer to the owner’s manual for locations of each part of the cooling system. Open the radiator drain to drain the cooling system and collect the drainage into a container (you can take this to your local auto parts store for recycling). Remove the thermostat. The thermostat is typically on the engine side of the upper radiator hose, but in some cases its on the lower hose. Disconnect the lower radiator hose, force coolant through the thermostat housing (upper hose and engine block) with a garden hose until the water runs clear. Connect the garden hose to the radiator, flush water through the radiator until the water runs clear. Completely drain the cooling system of all the water. Reinstall the thermostat. Disconnect the reservoir and flush with the garden hose until water runs clear.Completely drain the reservoir. Connect the hoses and close the drains. Fill the cooling system with new coolant to manufacturer's specification and distilled water. Use a 50/50 mixture of coolant and distilled water or pre- |

# Fine tuning the model

The Colab T4 GPU, with its limited 16 GB of VRAM, can barely accommodate the weights of Llama 2–7b, making full fine-tuning unfeasible. To address this, we will employ parameter-efficient fine-tuning techniques such as Lora or Qlora. Specifically, we'll use the Qlora technique to fine-tune the model in 4-bit precision, optimizing VRAM usage. For this, we will leverage the Hugging Face ecosystem of libraries, including transformers, accelerate, peft, trl, and bitsandbytes.

## Setting up the environment

A file meta_finetune.ipynb is created in colab and the runtime type is changed to T4 GPU.




**Step 1:** Python environment is set up with the following tools and libraries necessary for efficient and accelerated training and fine-tuning, with support for optimization and reinforcement learning.

```
%%capture
%pip install accelerate peft bitsandbytes transformers trl
```

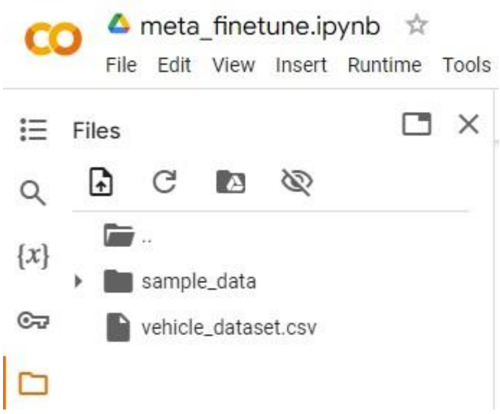
- %%capture: a Jupyter Notebook magic command that will help to install these libraries and their dependencies without printing any output
- accelerate: useful for accelerating the training of machine learning models, making use of multiple GPUs, or even CPUs in a distributed fashion.
- peft: a library that provides various techniques for fine-tuning large language models in a parameter efficient way without the need for extensive computational resources by tuning only a subset of parameters.
- bitsandbytes: provides efficient memory and computation optimizations, such as 8-bit optimizers, which can significantly reduce the memory footprint and speed up training.

- transformers: popular library for natural language processing (NLP) that provides pretrained models and tools for fine-tuning and deploying them.
- Trl (Transformers Reinforcement Learning) : used to fine-tune transformers using reinforcement learning techniques, it is used since supervised learning data is scarce.

**Step 2:** necessary modules from these libraries are loaded and an environment is set for fine-tuning and deploying a transformer-based language model using several advanced libraries.

```
import os
import torch
from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    TrainingArguments,
    pipeline,
    logging,
)
from peft import LoraConfig
from trl import SFTTrainer
```

**Step 3:** We will load the formatted data file vehicle_dataset.csv in colab.




```
import pandas as pd
from datasets import Dataset
dataset = pd.read_csv("/content/vehicle_dataset.csv",encoding='latin-1')
```

```
dataset = Dataset.from_pandas(dataset)
```

## Quantization of the model

**Step 4:** quantization setup is configured for optimizing the memory usage and computational efficiency of a neural network model using the BitsAndBytesConfig from the transformers library. We have sets up a 4-bit quantization scheme with Normal Float 4 (NF4) quantization type specifying that computations should be done in 16-bit floating point precision. Double quantization is disabled to avoid added complexity. We used this setup as it is useful for deploying large language models on resource-constrained hardware.

```
compute_dtype = getattr(torch, "float16")

quant_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=False,
)
```

- compute_dtype variable is set to torch.float16. By using the getattr function, the float16 attribute from the torch module is dynamically retrieved. The float16 data type is a 16-bit floating point, less precise than the 32-bit floating point. However, float16 offers advantages in reducing memory usage and speeding up computations.
- load_in_4bit = True : flag parameter that indicates that the model should be loaded using 4-bit precision. It reduces the precision of model's weights and activations, which compresses the model and reduces the memory usage and improve inference speed.
- bnb_4bit_quant_type="nf4" : specifies the type of 4-bit quantization to use, "nf4"(Normal Float 4) which is a quantization scheme designed to maintain accuracy even with low precision of model's weights.
- bnb_4bit_compute_dtype=compute_dtype : This sets the computation data type for the quantized model to torch.float16, meaning that even though the model weights are stored in 4-bit precision, computations during inference or training will be carried out in 16-bit floating point precision. This helps in maintaining a balance between performance (speed) and accuracy.
- bnb_4bit_use_double_quant=False : Double quantization involves an additional quantization step, which can help in further reducing the model size and improving performance but it also requires

more memory and computational resources. We have prioritized memory efficiency over extra accuracy.

**Step 5:** Loading tokenizer

**tokenizer should be loaded before the model (llama2) as it may sometimes show error due to the directory name of the model.

**The token copied from huggingface is stored in a variable Access_Token.

```
tokenizer = AutoTokenizer.from_pretrained(
    "meta-llama/Llama-2-7b-chat-hf",
    token = Access_Token,
    trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

- AutoTokenizer.from_pretrained : automatically select the appropriate tokenizer class for a given pre-trained model.
- trust_remote_code=True : allows the loading process to trust and execute any custom code that might be provided by the model repository.
- tokenizer.pad_token = tokenizer.eos_token: sets the padding token of the tokenizer to be the same as the end-of-sequence token. By doing this, we ensure that padding sequences are properly handled by the model.
- tokenizer.padding_side = "right" : if a sequence is shorter than the maximum sequence length in a batch, the padding tokens will be added to the end of the sequence rather than the beginning.

**Step 6:** Loading Llama 2 model

```
llama_model = AutoModelForCausalLM.from_pretrained(
    "meta-llama/Llama-2-7b-chat-hf",
    token = Access_Token,
    quantization_config=quant_config,
    device_map={"": 0}
)
llama_model.config.use_cache = False
llama_model.config.pretraining_tp = 1
```

- Here 7-billion parameter version of the Llama (Large Language Model Meta AI) model, fine-tuned for chat applications is loaded from Hugging Face.
- device_map={"": 0} : specifies that the model should be loaded onto the first device (0). It is used to distribute model layers across multiple devices.
- llama_model.config.use_cache = False : By default, transformers might cache the past key values during generation to speed up subsequent token generation. Disabling the cache might be useful in certain scenarios where you want to save memory or are not generating long sequences.
- llama_model.config.pretraining_tp = 1 : value different than 1 will activate the more accurate but slower computation of the linear layers.

```
# before fine tuning
prompt = "How to stop vehicle Noises?"
pipe = pipeline(task="text-generation", model=llama_model, tokenizer=tokenizer, max_length=200)
result = pipe(f"<s>[INST] {prompt} [/INST]")
print("before fine tuning",result[0]['generated_text'])
```

We set up a text generation pipeline using a pre-trained Llama model and tokenizer, and then use it to generate text based on a given prompt. The pipeline is configured to handle the text generation task, with a maximum length of 200 tokens for the generated output. The prompt is wrapped in special instruction tokens to help the model understand the input format. The generated text is then printed out, showing the original model's response to the prompt before any fine-tuning.

## Setting up parameters for fine tuning

We have used parameter-efficient fine-tuning (PEFT) using the Low-Rank Adaptation (Lora) technique. The LoraConfig class is part of the peft library, which allows efficient fine-tuning of large language models by updating a small subset of model parameters.

**Step 7:** PEFT parameters

```
peft_params = LoraConfig(
    lora_alpha=16,
    lora_dropout=0.1,
    r=64,
    bias="none",
    task_type="CAUSAL_LM",
)
```

- lora_alpha=16: This parameter controls the scaling factor for the low-rank updates applied during fine-tuning. A higher lora_alpha value increases the influence of the low-rank adaptations on the model's output. We have set to 16, indicating that low-rank matrix will be scaled by a factor of 16 before being added to model's weights. It is a tradeoff between general knowledge and task specific adaptation.
- lora_dropout=0.1: Dropout is a regularization technique that helps prevent overfitting and to learn generalizable features by randomly setting a fraction of the neurons to zero during training. Here, it is set to 0.1, meaning 10% of the neurons in the low-rank layers will be dropped during each training step.
- r=64: A higher rank allows the model to learn more complex adaptations, but it also increases the computational cost. 64 indicate a relatively small rank which can reduce risk of overfitting.
- bias="none" : Setting bias to "none" means that no bias terms will be adapted during fine-tuning that means that only the weights of the low-rank adaptation layers will be updated, not the biases.
- task_type="CAUSAL_LM" : This parameter indicates the type of task for which the model is being fine-tuned. "CAUSAL_LM" involves generating text in an autoregressive manner (i.e., predicting the next word given the previous words). This is suitable for tasks like text generation, where the model generates text one token at a time.

**Step 8:** Setting up the Training parameters

```
training_params = TrainingArguments(
    output_dir="./results",
    num_train_epochs=2,
    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"
)
```

- output_dir="./results" : Specifies the directory where the model checkpoints and training logs and evaluation result will be saved. In this case, the directory is ./results.
- num_train_epochs=2 : Indicates that the entire training dataset will be passed through the model 2 times. Value is relatively small as our dataset is relatively small. If value is large then model may over fit.
- per_device_train_batch_size=4 : specifies that each device will process 4 samples parallel per batch during training since we have limited GPU.
- gradient_accumulation_steps=1 : gradients will be updated every step without accumulation therefore overhead of weight updates is reduced.
- optim="paged_adamw_32bit" : Specifies the optimizer to use for training. paged_adamw_32bit is a variant of the AdamW optimizer that operates in 32-bit precision. AdamW is an Adam optimizer variant with weight decay regularization.
- save_steps=25 : Here, the model will save model's weight and training state every 25 steps(batches).
- logging_steps=25 : number of training steps between each logging event is 25. Training logs, metrics and other information will be written to the output directory every 25 batches. Logging too frequently will increase output file size and too infrequently may mean missing important training insights.
- learning_rate=2e-4 : it is set to 0.0002. Smaller learning rate means smaller updates to model's weights and gradual learning. Less risk of overshooting optimal weights.
- weight_decay=0.001 : Specifies the weight decay (L2 regularization) applied to the model's weights to prevent overfitting. Here it is set to 0.001a moderate value striking a balance between regularization and gradient damping.
- fp16=False : Indicates model will use 32-bit precision for weights and computation ensuring accuracy and stability.
- bf16=False : Indicates whether to use 16-bit brain floating-point precision. It is also set to False.
- max_grad_norm=0.3 : Gradient clipping helps prevent exceeding gradients by scaling the gradients to a maximum norm of 0.3.

- max_steps=-1 : Setting this to -1 means there is no limit on the number of training steps. Training will proceed for the specified number of epochs regardless of number of batches.
- warmup_ratio=0.03 : Defines the fraction of total training steps used for the learning rate warm-up. Here, 3% of the total steps will be used to linearly increase the learning rate from 0 to the specified learning rate. It will gradually adjust the model to training process.
- group_by_length=True : Indicates to group training samples of similar lengths together. Grouping by length can improve efficiency by reducing padding hence reducing computational waste.
- lr_scheduler_type="constant" : Specifies learning rate will remain constant throughout the training without any changes or adjustments.
- report_to="tensorboard" : Indicates that training metrics and logs will be reported to TensorBoard, a tool for visualizing and tracking machine learning experiments.

**Step 9:** Model Fine tuning using the SFTTrainer class from the trl library

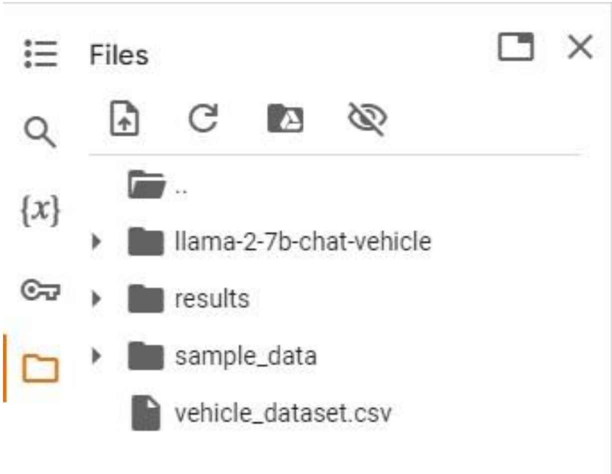
```
trainer = SFTTrainer(
    model=llama_model,
    train_dataset=dataset,
    peft_config=peft_params,
    dataset_text_field="text",
    max_seq_length=None,
    tokenizer=tokenizer,
    args=training_params,
    packing=False,
)
trainer.train()
```

- dataset_text_field="text" : Specifies the field(column) in the dataset that contains the sample data to be used for training.
- max_seq_length=None : Defines that sequence length will be determined dynamically based on the data.
- packing=False : Setting this to False means that each sequence will be processed individually without packing.

## Saving the fine-tuned model

**Step 10**: Save the model adopter and tokenizers to the specified directory.

```
trainer.model.save_pretrained("llama-2-7b-chat-vehicle")
trainer.tokenizer.save_pretrained("llama-2-7b-chat-vehicle")
```



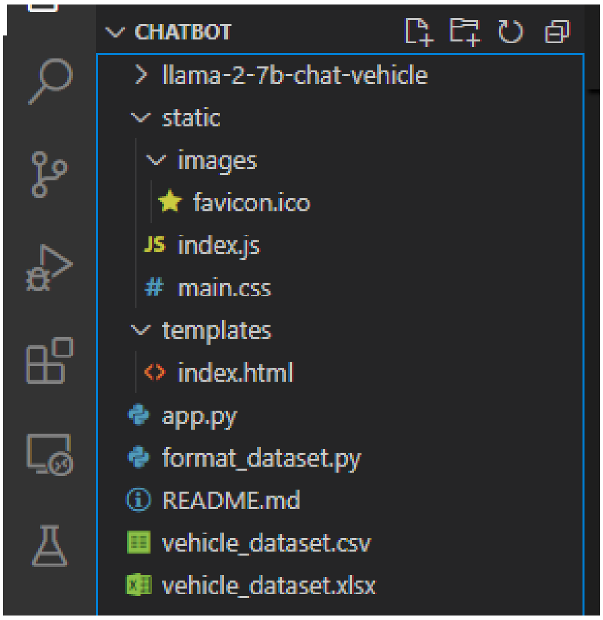

## Using our fine-tuned model

We will load the model again as before, but now instead of original llama model we will load our fine-tuned model

```
model_dir = "/content/llama-2-7b-chat-vehicle"
fine_tuned_model = AutoModelForCausalLM.from_pretrained(
    model_dir, quantization_config=quant_config,
    device_map={"": 0}
    )
fine_tuned_tokenizer = AutoTokenizer.from_pretrained(model_dir)

prompt = "How to stop vehicle Noises?"
pipe = pipeline(task="text-generation", model=fine_tuned_model, tokenizer=fine_tuned_tokenizer,
max_length=200)
result = pipe(f"<s>[INST] {prompt} [/INST]")
print("After fine tuning",result[0]['generated_text'])
```

# Frontend




```
#index.html
<div id="chat">
      <p class="botText">Hi! How can I help you?</p>
</div>
<div id="userInput">
      <input id="textInput" type="text" name="msg" placeholder="Enter your query..."
required></input>
      <button id="submit" onclick="getBotResponse()"><i class="fa fa-send" style="font-
size:25px"></i></button></label><br>
</div>
```

JavaScript file is invoked

```
function getBotResponse() {
   var rawText = document.getElementById("textInput").value;
   var userHtml = '<p class="userText">' + rawText + '</p>';
   document.getElementById("chat").innerHTML += userHtml;
   document.getElementById("chat").scrollTop = document.getElementById("chat").scrollHeight;
   document.getElementById("textInput").value = "";
```

```
fetch('/get?msg=rawText')
          .then(response => response.text())
          .then(data => {
            const botHtml = '<p class="botText">' + data + '</p>';
            const chatElement = document.getElementById("chat");
            chatElement.innerHTML += botHtml;
            chatElement.scrollTop = chatElement.scrollHeight;
          })
          .catch(error => console.error('Error fetching data:', error));
}
```

Raw text from user is passed to flask. The answer from the fine-tuned model is sent back to JavaScript. JavaScript appends the answer in the html.

## Flask Connection

```
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    result = pipe(f"<s>[INST] {userText} [/INST]")
    return str(result[0]['generated_text'])

if __name__ == "__main__":
    app.run(debug=True)
```

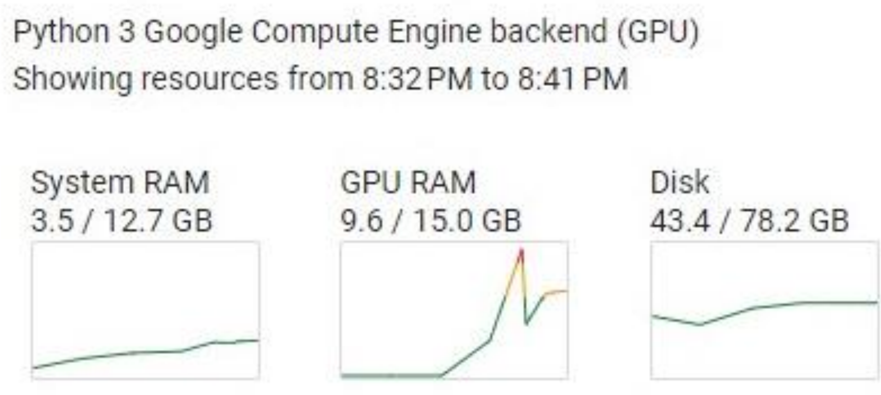
To launch the chatbot in localhost run the command

**python app.py**

# Conclusion

In developing our vehicle chatbot, we have demonstrated the power and versatility of the LLaMA2 model in addressing complex queries related to vehicle maintenance and servicing. By fine-tuning the 7B parameter LLaMA2 model, we were able to create a highly responsive and intelligent assistant that meets the diverse needs of vehicle owners.

To overcome the inherent memory and compute limitations associated with large language models, we employed advanced fine-tuning techniques such as QLoRA, PEFT, and SFT. These methods, in conjunction with Hugging Face libraries like transformers, accelerate, peft, trl, and bitsandbytes, enabled us to optimize the model efficiently on a consumer GPU. This approach not only made the fine-tuning process feasible but also ensured that the chatbot performs at a high level without requiring extensive computational resources.




In conclusion, the integration of LLaMA2 into our vehicle chatbot has significantly enhanced its ability to deliver accurate and contextually relevant responses to users. This project exemplifies the potential of leveraging cutting-edge machine learning techniques to create practical and user-friendly solutions in the field of vehicle maintenance and servicing. We are confident that this chatbot will serve as a reliable tool for vehicle owners, helping them maintain their vehicles more effectively and efficiently.

# References


1. https://medium.com/towards-generative-ai/understanding-llama-2-architecture-its-ginormous-impact-on-genai-e278cb81bd5c
2. https://www.datacamp.com/tutorial/fine-tuning-llama-2
3. https://huggingface.co/docs/trl/main/en/sft_trainer
4. https://huggingface.co/docs/trl/main/en/index
5. https://huggingface.co/docs/peft/index