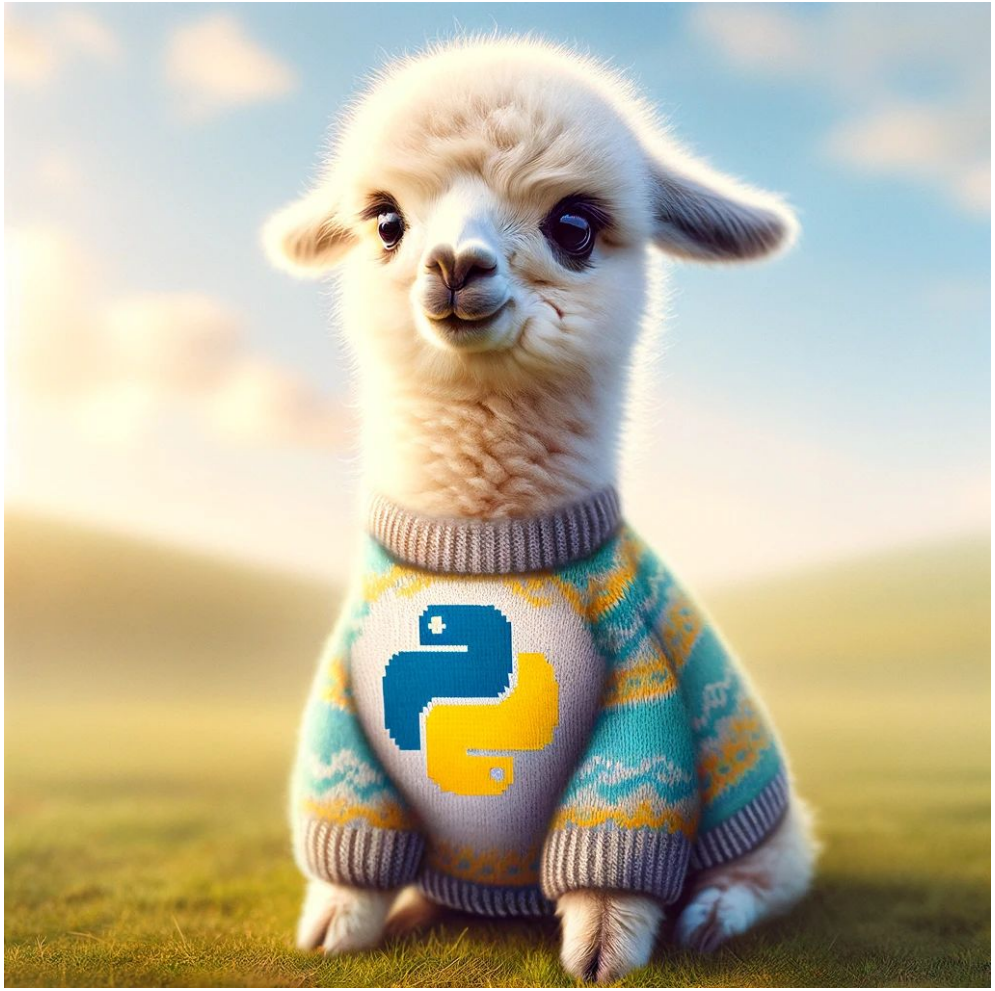
Pythonized LLaMA

Muhammed Saneens &
Pratiksha Pai



Motivation

- A local interface to help with python programming!

Problem:

- Github Copilot and other services rely on OpenAI APIs and stable internet connection to help you while coding
- They are not customizable
- Building on top of paid service like OpenAI gets very costly very soon
- Hence we wanted a local interface built on open-source LLMs that could help us as a pair programmer

Approach: Base Model

- Our model is built on top of SOTA **LLaMA-2 7 B**illion parameter base model.
- Llama 2 by *Meta*
  - LLM based on transformer architecture
  - Compressed internet data
  - Auto-regressive in nature
- Llama 2 base model is a document completer.
- For our problem we use the base model that is adapted for **chat** and **hugging face transformer's** library by *NousResearch*: NousResearch/Llama-2-7b-hf

LLaMa2 - https://ai.meta.com/llama/
NousResarch - https://huggingface.co/NousResearch

Approach: Dataset & SFT

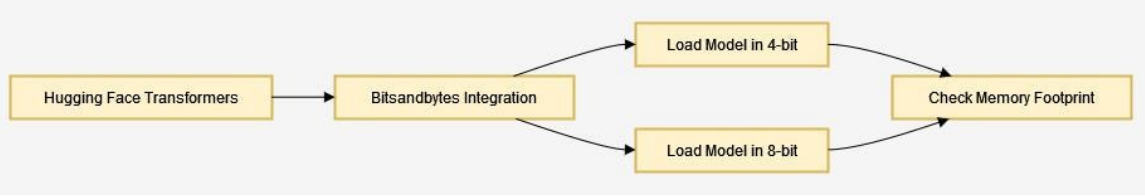
- We use **instruction based supervised fine tuning (SFT)** on the model using the dataset *python_code_instructions_18k_alpaca* on 🤗

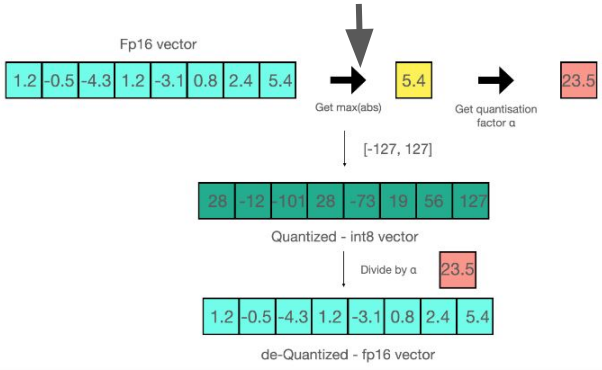
```
Prompt:
### Instruction:
Use the Task below and the Input given to write the Response, which is a programming code that can solve the Task.

### Task:
Write a Python function to display the first and last elements of a list.

### Input:


### Response:


Generated instruction:
def first_last(list):
    return list[0], list[-1]
```

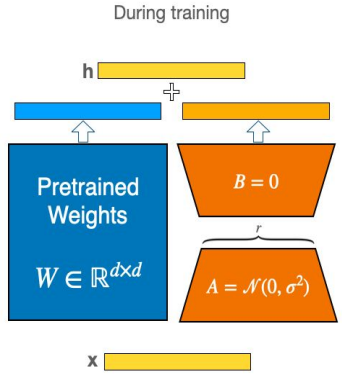
Dataset: https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca

Approach: Loading Pretrained Model + PEFT

- Download & load the model into memory using **4 bit quantization**
- Fine tuning = Pre-trained model + weight updates for **our dataset**
- **PEFT** - Parameter efficient Fine Tuning - adjust small number of parameters/**data**

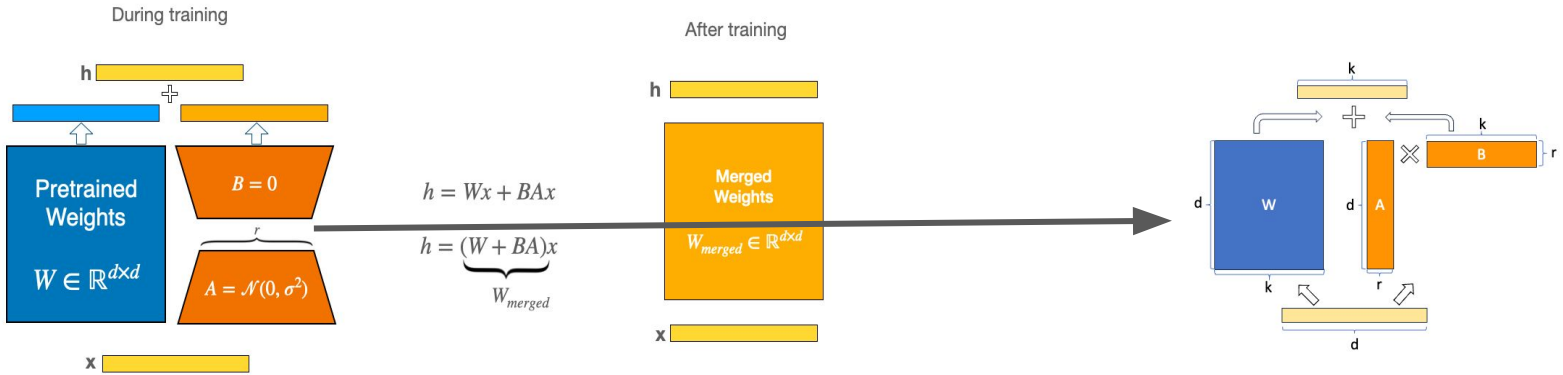






Approach: QLoRA (Quantized Low-Rank Adaptation)

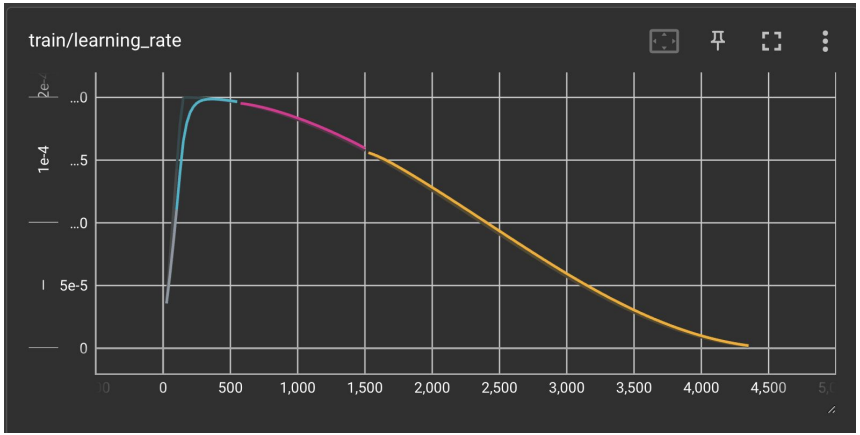
- Injects trainable matrices into Transformer layers, reducing the number of parameters.
- Reduces trainable parameters by 10,000 times compared to full fine tuning.
- 4-bit Normal Float (NF4): A new data type optimized for normally distributed weights, enhancing memory efficiency.

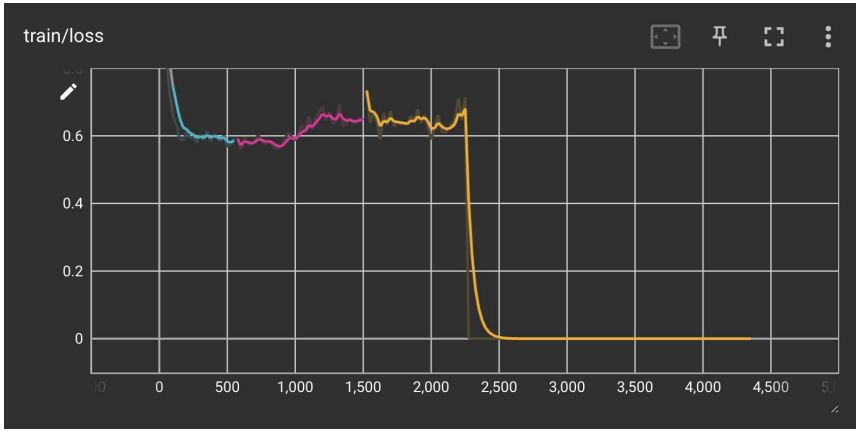


LoRA: Low-rank adaptation of Large Language models
https://arxiv.org/abs/2106.09685

Experiments: Training & Hyperparameters

- The fine tuning of the model took around 8 hours on single NVIDIA A100 GPU
- After this stage we **merge** the pretrained model + PEFT updates to get the final model
- This is then pushed to 🤗





Experiments: Inference

- After having published it to HF, we can simply download it, for inference
- We write a small python script to infer this model for various tasks, locally
- Some of the examples from the model

Example 1: Task without any inputs

```
Prompt:
### Instruction:
Use the Task below and the Input given to write the Response, which is a programming code that can solve the Task.

### Task:
Write a Python function to display the first element of the list.

### Input:


### Response:


Generated instruction:
def first_element(list):
    return list[0]
```

Experiments: Inference

Example 2: A task with inputs

```
Prompt:
### Instruction:
Use the Task below and the Input given to write the Response, which is a programming code that can solve the Task.

### Instruction:
### Task:
Generate a Python code for crawling a website for a specific type of data.

### Input:
website: www.example.com data to crawl: 0123456789

### Response:


Generated instruction:
import requests
import lxml.html

def crawl_website(website, data):
    # Get the HTML of the website
    html = requests.get(website).text

    # Parse the HTML
    tree = lxml.html.fromstring(html)

    # Get the data
    data = tree.xpath('//div[@class="data"]/text()')

    # Return the data
    return data
```

```
Prompt:
### Instruction:
Use the Task below and the Input given to write the Response, which is a programming code that can solve the Task.

### Instruction:
### Task:
Generate a Python code for crawling a website for a specific type of data.

### Input:
website: www.example.com data to crawl: 0123456789

### Response:


Generated instruction:
import requests
import bs4

url = 'www.example.com'
data = '0123456789'

def crawl_website(url, data):
    r = requests.get(url)
    soup = bs4.BeautifulSoup(r.text, 'html.parser')
    for link in soup.find_all('a'):
        if link.text == data:
```

Experiments: Inference

Example 3: Multiple Tasks

```
Prompt:
### Instruction:
Use the Task below and the Input given to write the Response, which is a programming code that can solve the Task.

### Instruction:
### Task:
Given the root of a binary tree, return the inorder traversal of its nodes' values. After this code given the root
of a binary tree, return the postorder traversal of its nodes' values

### Input:


### Response:


Generated instruction:
def inorderTraversal(root):
    if root is not None:
        inorderTraversal(root.left)
        print(root.val)
        inorderTraversal(root.right)

def postorderTraversal(root):
    if root is not None:
        postorderTraversal(root.left)
        postorderTraversal(root.right)
        print(root.val)

# Driver Code
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)

inorderTraversal(root)
postorderTraversal(root)
```

Conclusion

- We successfully fine tuned LLaMa 7b model on a single GPU to get a model fine tuned for Python programming
- We learned a ton about Llama and other open-source alternatives to ChatGPT
- Learned QLoRA
- Successfully inferred our model to perform simple python tasks, locally!

Thanks :)