AI VIETNAM
Seminar

QLoRa: Efficient Fine-tuning of Quantized LLMs

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer

Link: https://arxiv.org/abs/2305.14314

Nguyen-Thuan Duong
Dinh-Thang Duong

Outline

- ❖ Abstract
- ❖ Introduction
- ❖ Background
- ❖ Methods
- ❖ Evaluation
- ❖ Conclusion
- ❖ Question

Abstract

❖ Content

We present QLoRA, an efficient finetuning approach that reduces memory usage enough to finetune a 65B parameter model on a single 48GB GPU while preserving full 16-bit finetuning task performance. QLoRA backpropagates gradients through a frozen, 4-bit quantized pretrained language model into Low Rank Adapters~(LoRA). Our best model family, which we name Guanaco, outperforms all previous openly released models on the Vicuna benchmark, reaching 99.3% of the performance level of ChatGPT while only requiring 24 hours of finetuning on a single GPU. QLoRA introduces a number of innovations to save memory without sacrificing performance: (a) 4-bit NormalFloat (NF4), a new data type that is information theoretically optimal for normally distributed weights (b) double quantization to reduce the average memory footprint by quantizing the quantization constants, and (c) paged optimziers to manage memory spikes. We use QLoRA to finetune more than 1,000 models, providing a detailed analysis of instruction following and chatbot performance across 8 instruction datasets, multiple model types (LLaMA, T5), and model scales that would be infeasible to run with regular finetuning (e.g. 33B and 65B parameter models). Our results show that QLoRA finetuning on a small high-quality dataset leads to state-of-the-art results, even when using smaller models than the previous SoTA. We provide a detailed analysis of chatbot performance based on both human and GPT-4 evaluations showing that GPT-4 evaluations are a cheap and reasonable alternative to human evaluation. Furthermore, we find that current chatbot benchmarks are not trustworthy to accurately evaluate the performance levels of chatbots. A lemon-picked analysis demonstrates where Guanaco fails compared to ChatGPT. We release all of our models and code, including CUDA kernels for 4-bit training.

Introduction

- ❖ Getting Started
- ❖ Applications
- ❖ Challenges

Introduction

❖ What is LLMs?

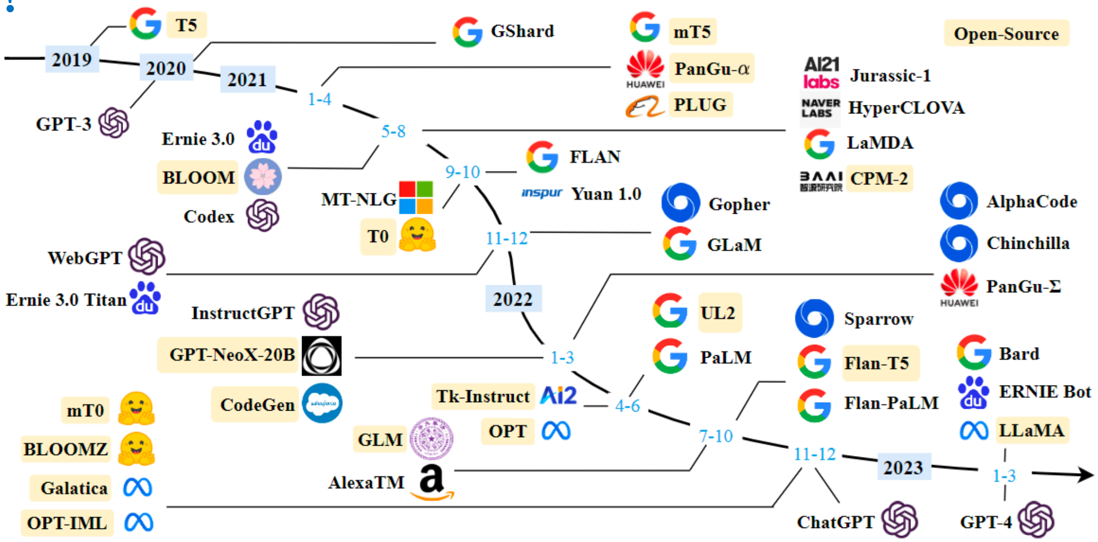


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

LLMs (Large Language Models): Language models that were trained on a very large corpus of text. This made them capable of performing various NLP tasks with high precision.

Image Reference: https://wandb.ai/vincenttu/blog_posts/reports/A-Survey-of-Large-Language-Models--VmlldzozOTY2MDM1

Introduction

❖ What is LLMs?

LLMs are often pretrained on a vast majority of data and designed to be adaptable to a wide variety of tasks (Foundation models).

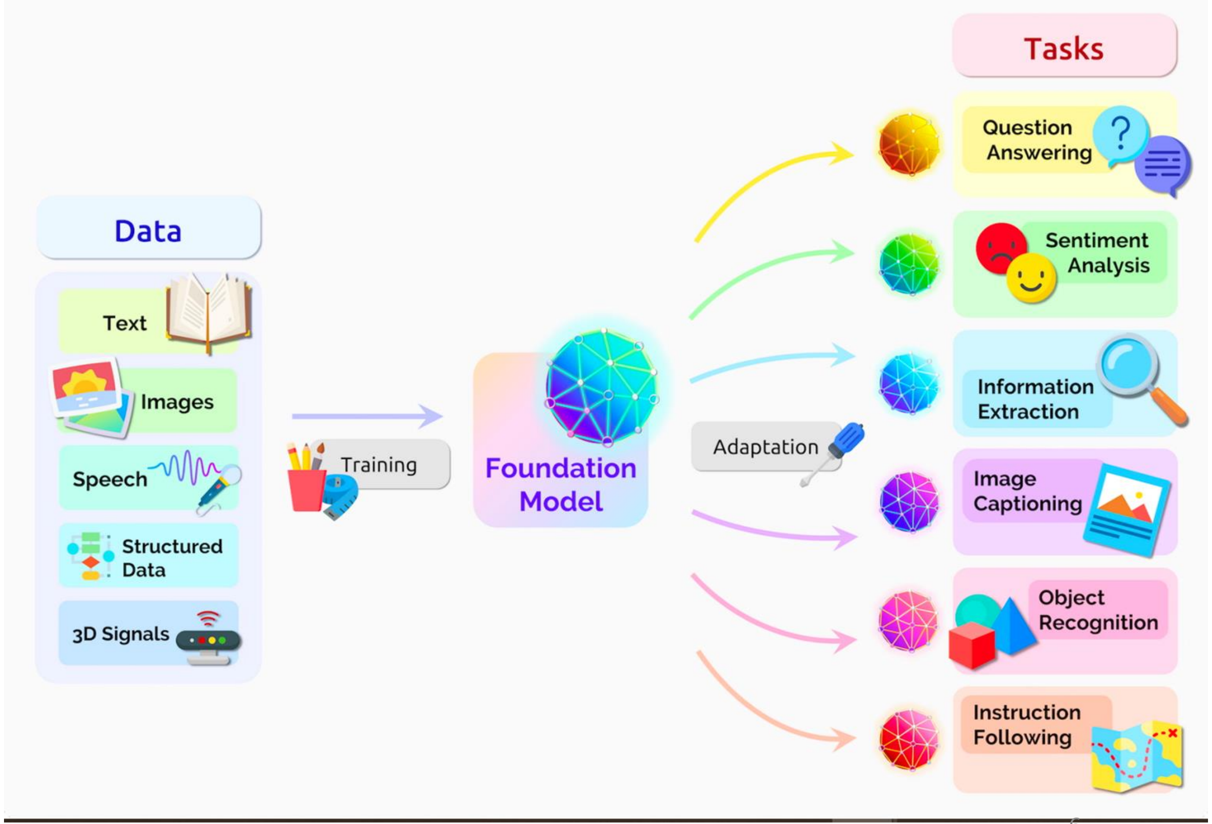
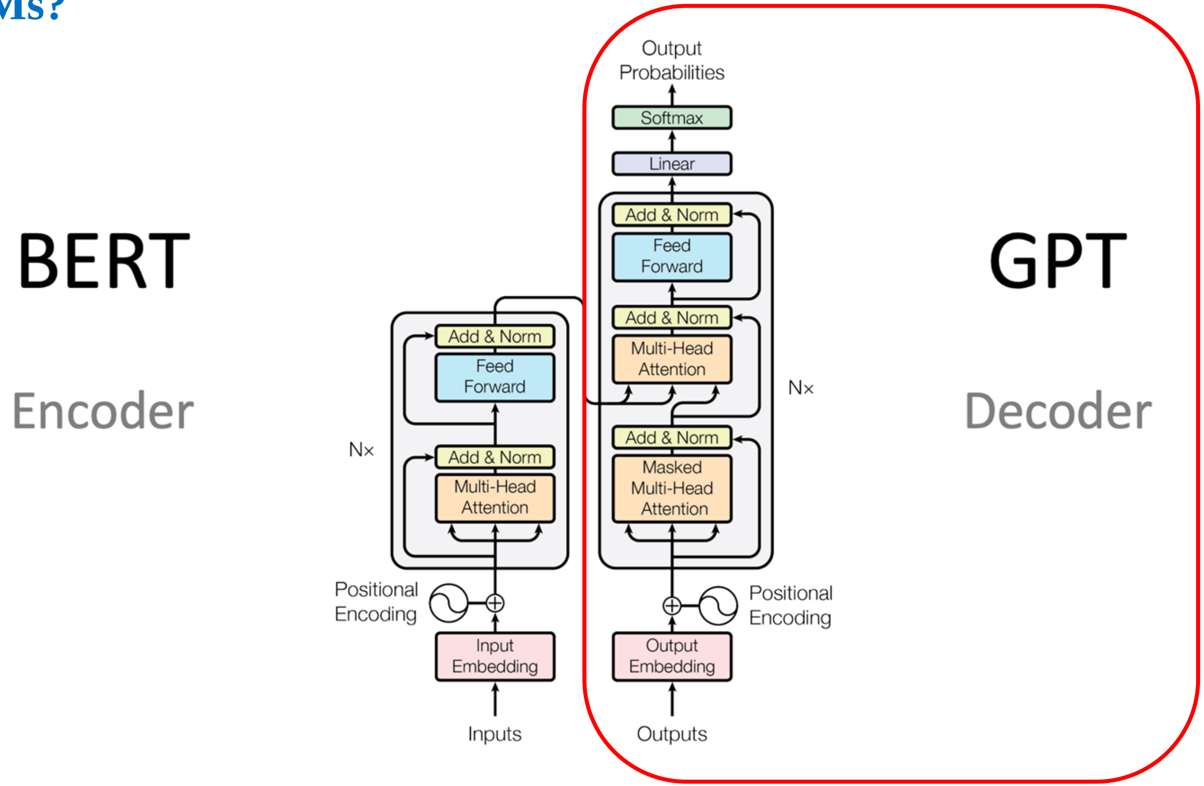


Image Reference: https://blogs.nvidia.com/blog/what-are-foundation-models/

Introduction

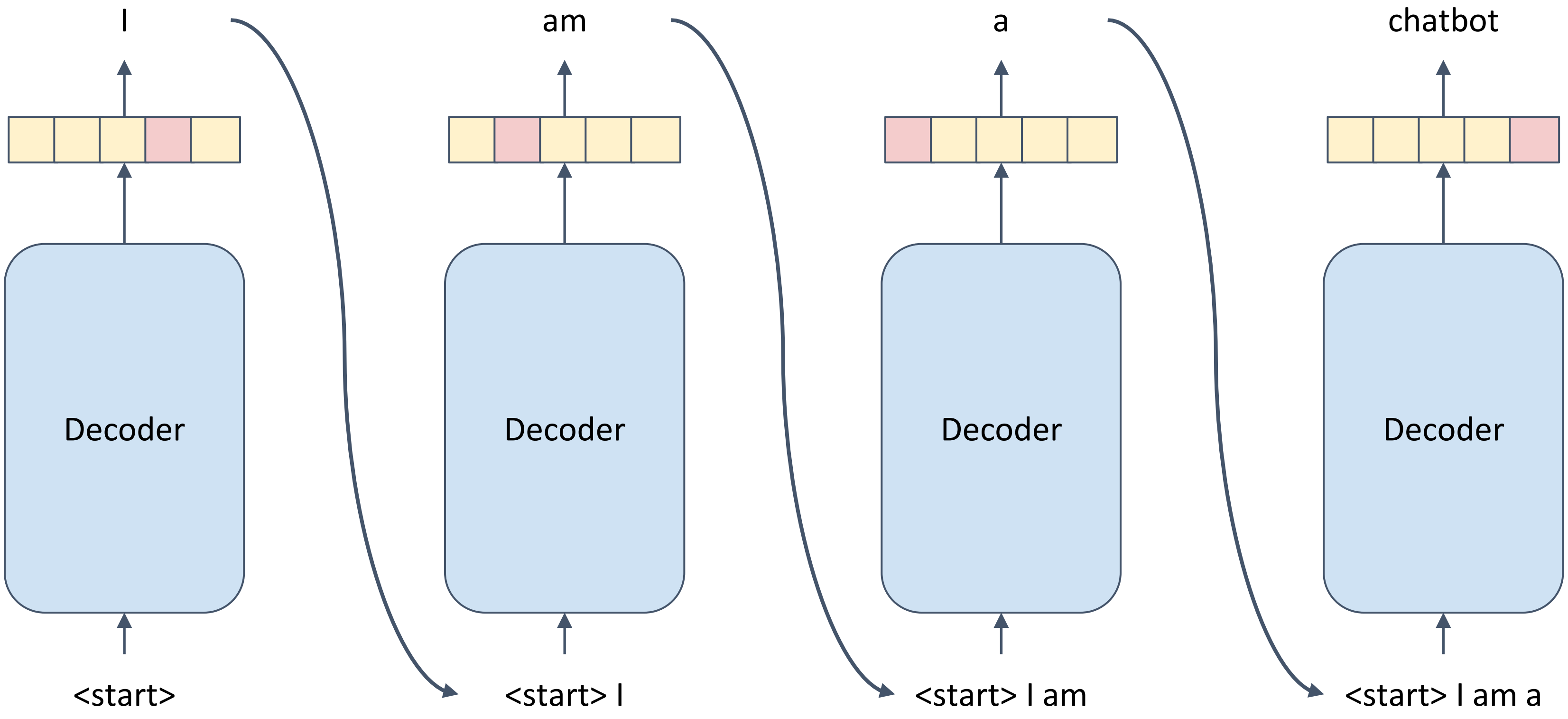
❖ What is LLMs?



Transformers Paper: https://arxiv.org/abs/1706.03762
Image Reference: https://heidloff.net/article/foundation-models-transformers-bert-and-gpt/

Introduction

❖ What is LLMs?



Introduction

❖ What is LLMs?

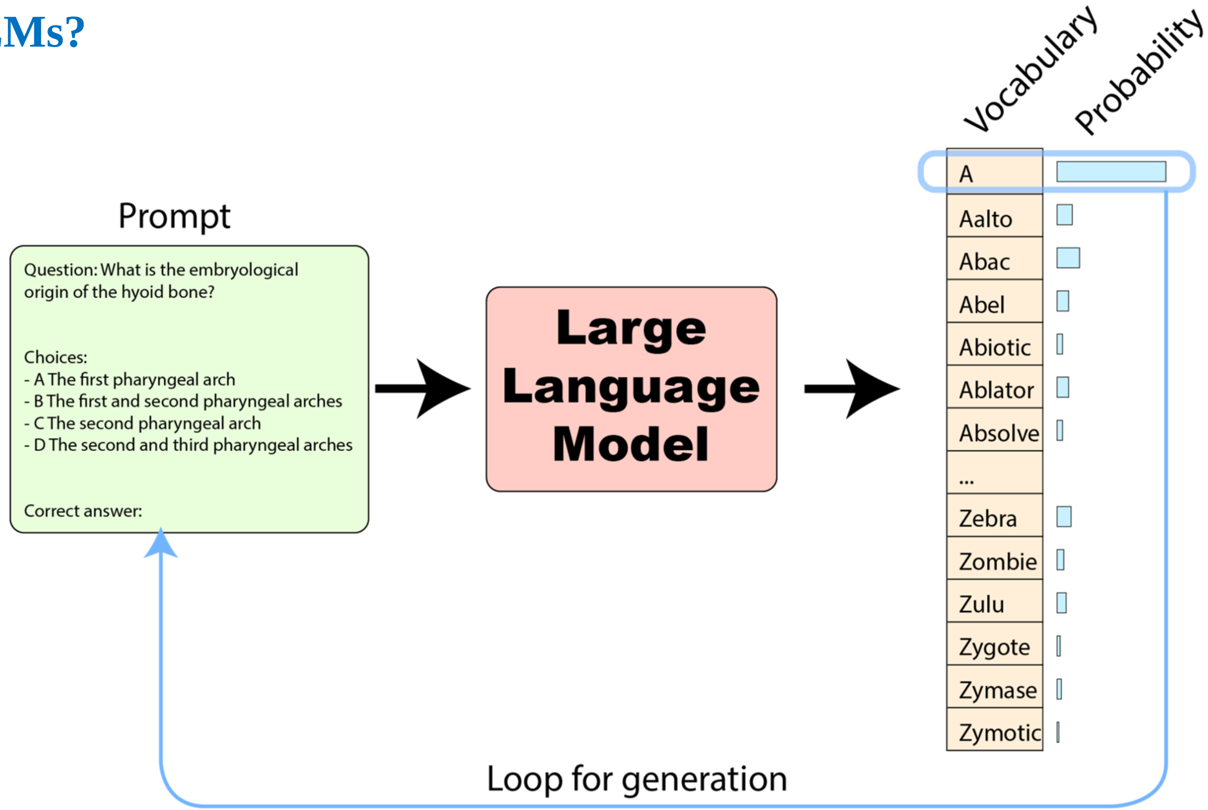


Image Reference: https://huggingface.co/blog/evaluating-mmlu-leaderboard

Introduction

❖ What is LLMs?

Prompt:

Write a python function that receive an image and plot it using matplotlib library.

You

Write a python function that receive an image and plot it using matplotlib library.

ChatGPT

To create a Python function that receives an image and plots it using the Matplotlib library, you can use the following code as an example. You'll need to have Matplotlib and a library like PIL (Pillow) installed to work with images:

python Copy code

```python
import matplotlib.pyplot as plt
from PIL import Image

def plot_image(image_path):
    try:
        # Open the image using PIL (Pillow)
        img = Image.open(image_path)

        # Create a Matplotlib figure and axis
        fig, ax = plt.subplots()
```

Introduction

❖ LLMs size over time

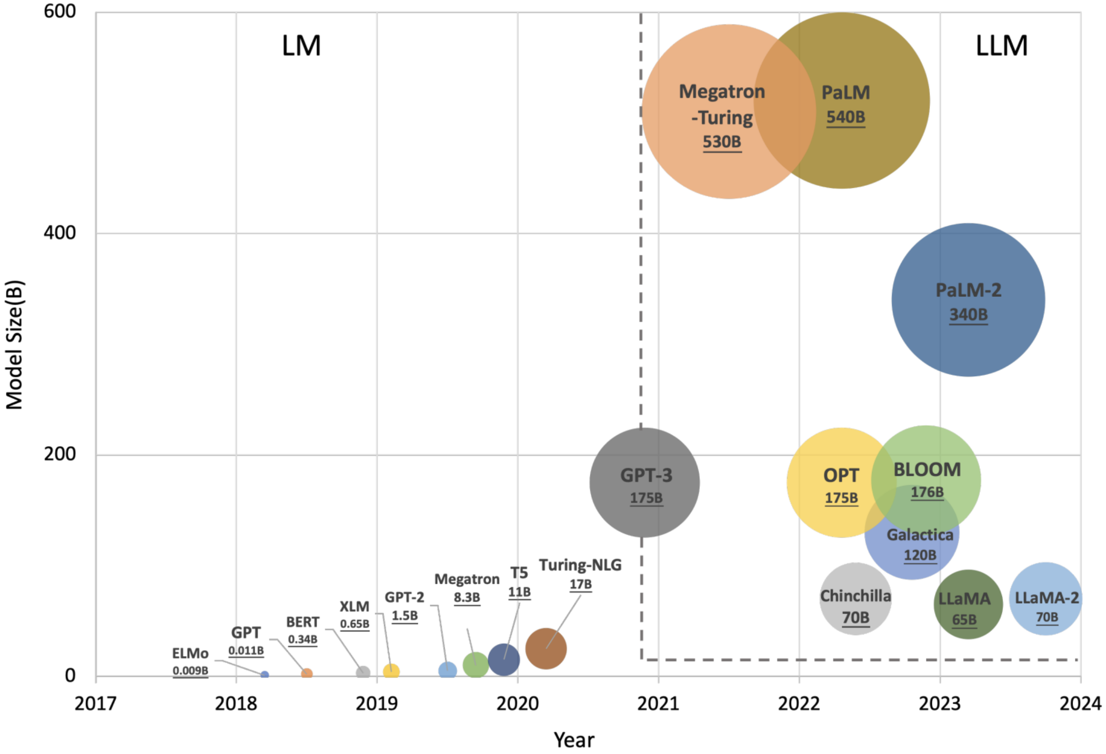
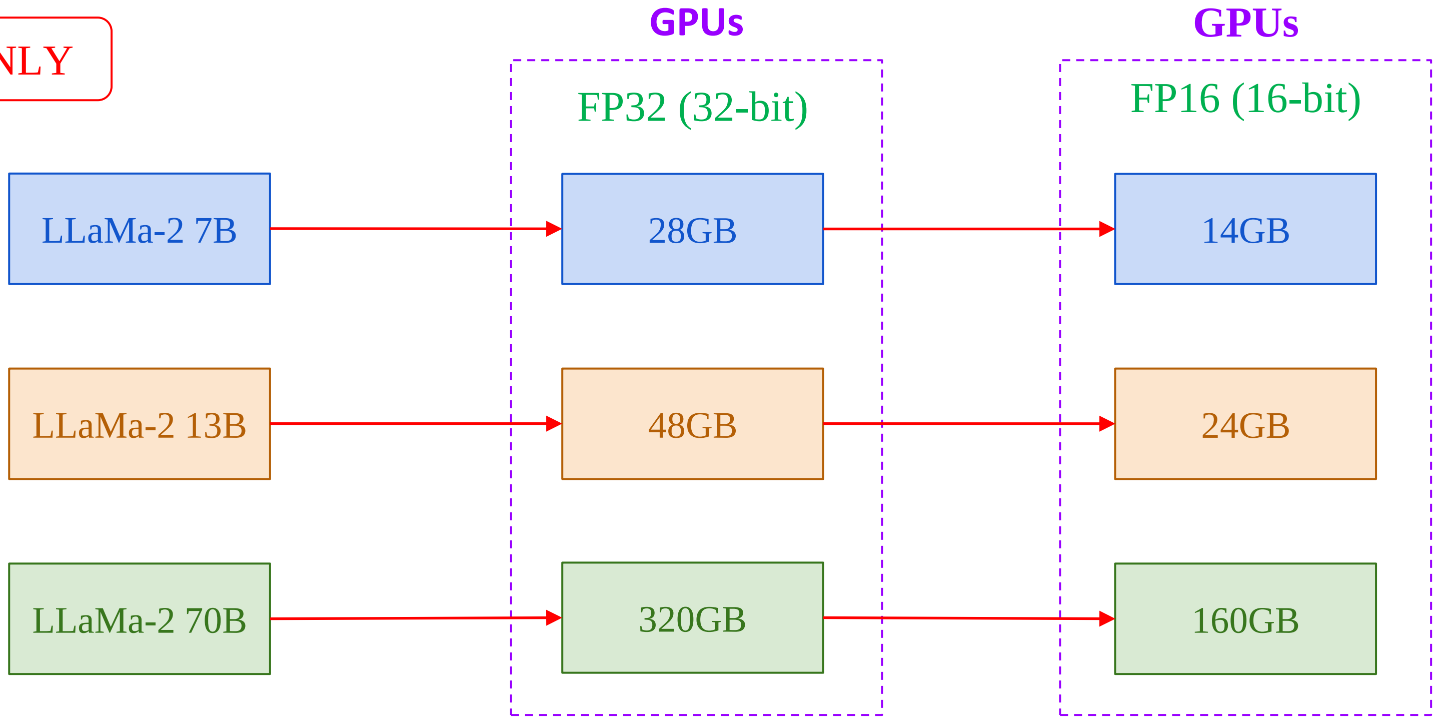


Image Reference: https://www.marktechpost.com/wp-content/uploads/2023/10/Screenshot-2023-10-15-at-4.25.57-PM.png

Introduction

❖ Low-precision

Load ONLY



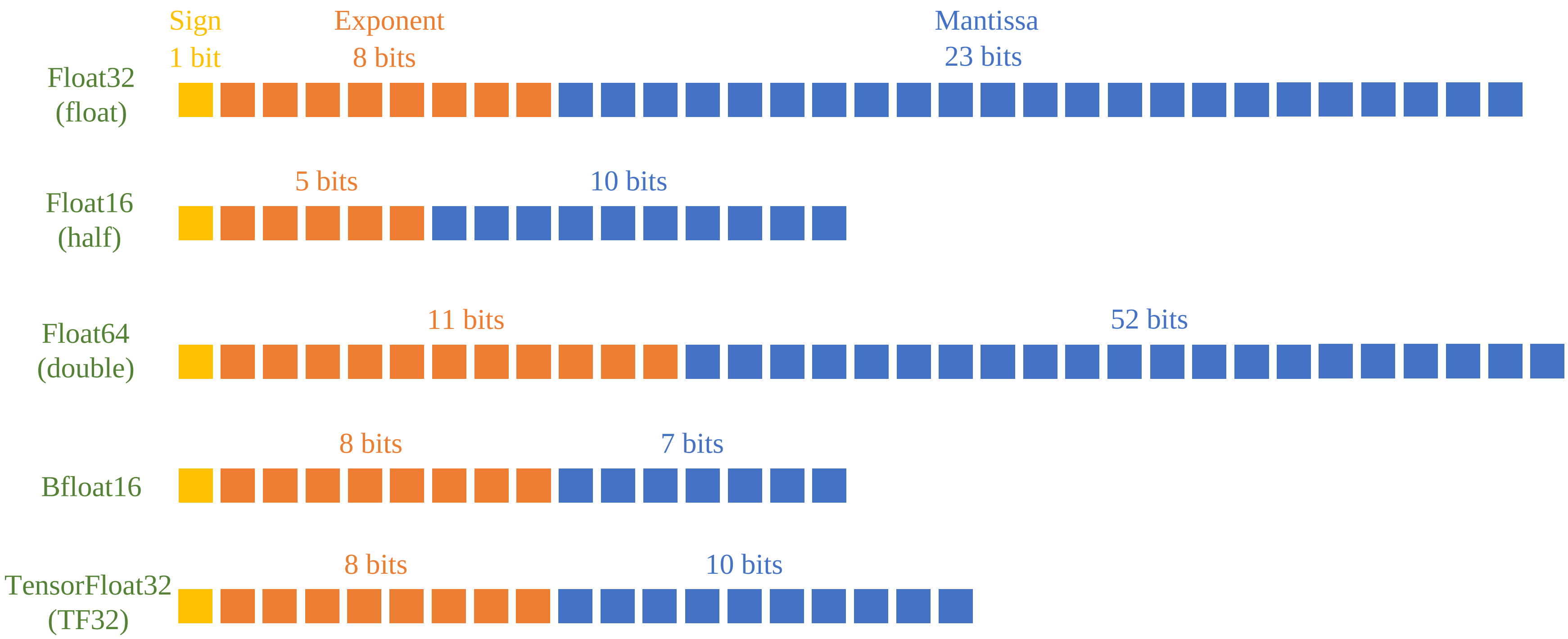
Reference: https://docs.nvidia.com/ai-enterprise/workflows-generative-ai/0.1.0/sizing-guide.html

Background

- Quantization
- LoRA

Background

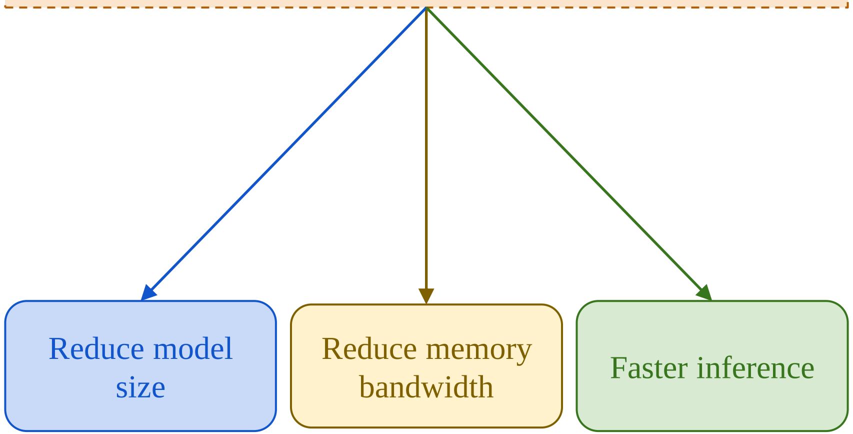
❖ Float Point (FP)

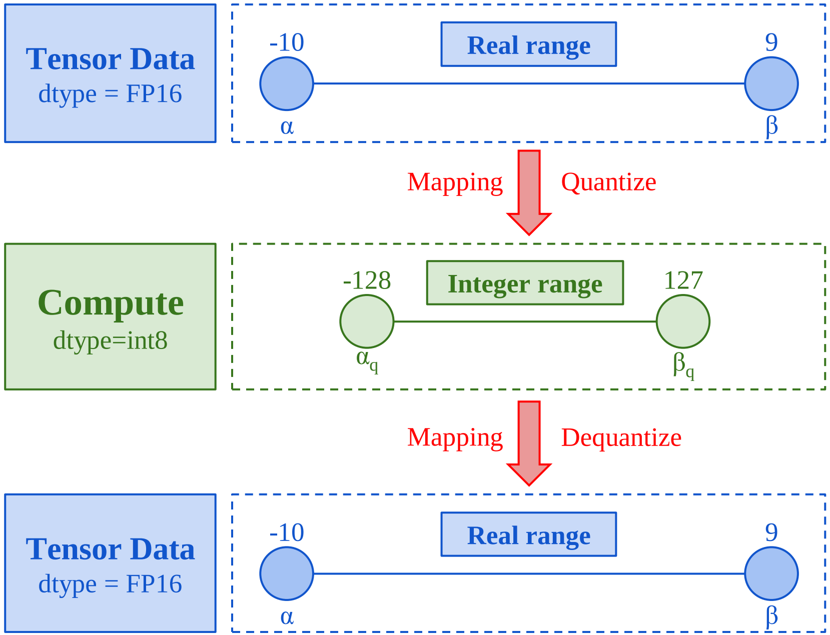


Background

❖ What is Quantization ?

Quantization refers to techniques for doing both computations and memory accesses with **<u>lower precision data</u>**, usually int8 compared to floating point implementations.

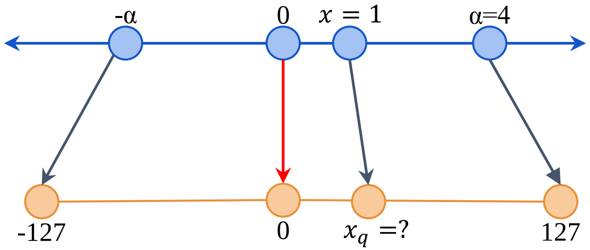




Reference: https://pytorch.org/docs/stable/quantization.html#introduction-to-quantization

Background

❖ Int8 Quantization (Weight)



Scale (Symmetric) Quantization

Quantize

$$x_q = clip(round(s \cdot x))$$

Dequantize

$$\hat{x} = \frac{1}{s} x_q$$

Scale

$$s = \frac{2^{b-1} - 1}{\alpha}$$

$$s = \frac{2^{8-1} - 1}{4} = 31.75$$

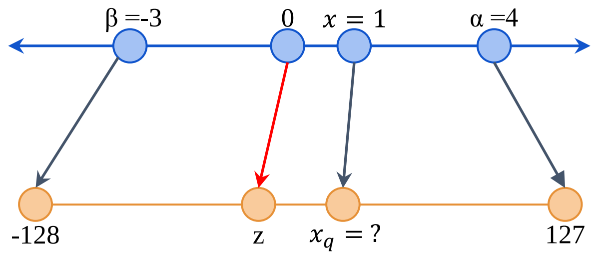
$$x_q = clip(round(31.75 \times 1))$$
$$= 32$$

$$\hat{x} = \frac{1}{31.75} 32 = 1.007874 \ldots$$

Reference: https://arxiv.org/pdf/2004.09602.pdf

Background

❖ Int8 Quantization (Weight)



Affine (Asymmetric) Quantization

Quantize: $x_q = clip(round(s \cdot x + z))$

Dequantize: $\hat{x} = \frac{1}{s}(x_q - z)$

Scale:

$$s = \frac{2^b - 1}{\alpha - \beta}$$

$$z = -round(\beta \cdot s) - 2^{b-1}$$

$$s = \frac{2^8 - 1}{4 - (-3)} = 36.42$$

$$z = -round(-3 \times 36.42) - 2^{8-1} = -19$$

$$x_q = clip(round(36.42 \times 1 - 19)) = 17$$

$$\hat{x} = \frac{1}{36.42}(17 + 19)) = 0.9882 \ldots$$

Reference: https://arxiv.org/pdf/2004.09602.pdf

Background

❖ Matrix Multiplication Quantization

$$quantize(x, s, z) = s \cdot x + z$$

$$dequantize(x, s, z) = \frac{1}{x}\left(x_q - z\right)$$

$$Y = XW + b$$

$$Y_{i,j} = \sum_{k=1}^{p} X_{i,k} W_{k,j} + b_j$$

$$Y_{q,i,j} = z_Y + \frac{s_Y}{s_b}\left(b_{q,j} - z_b\right) + \frac{s_Y}{s_X s_W}\left[\left(\sum_{k=1}^{p} X_{q,i,k} W_{q,k,j}\right) - \left(z_W \sum_{k=1}^{p} X_{q,i,k}\right) - \left(z_X \sum_{k=1}^{p} W_{q,k,j}\right) + p z_X z_W\right]$$

Reference: https://leimao.github.io/article/Neural-Networks-Quantization/#Floating-Point-Quantization

Background

❖ Activation Quantization

$$y = ReLU(x) = \begin{cases} 0 & if\ x < 0 \\ x & if\ x \geq 0 \end{cases}$$

$$y = dequantize(x_q, s_x, z_x) = \frac{1}{s_x}(x_q - z_x)$$

$$y = dequantize(y_q, s_y, z_y) = \frac{1}{s_y}(y_q - z_y)$$

$$\rightarrow \frac{1}{s_x}(x_q - z_x) = \frac{1}{s_y}(y_q - z_y)$$

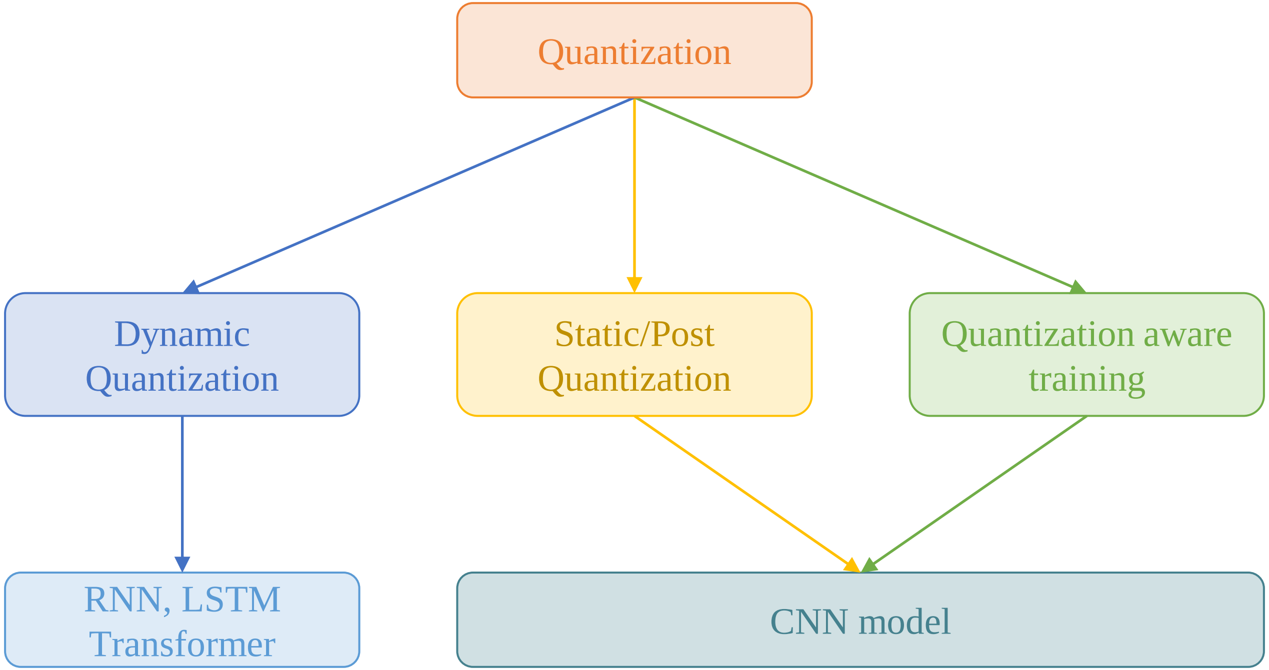
$$\rightarrow y_q = \frac{s_y}{s_x}(x_q - z_x)$$

$$y_q = \begin{cases} z_y & if\ x_q < z_x \\ z_y + \frac{s_y}{s_x}(x_q - z_x) & if\ x_q \geq z_x \end{cases}$$

Reference: https://leimao.github.io/article/Neural-Networks-Quantization/#Floating-Point-Quantization

Background

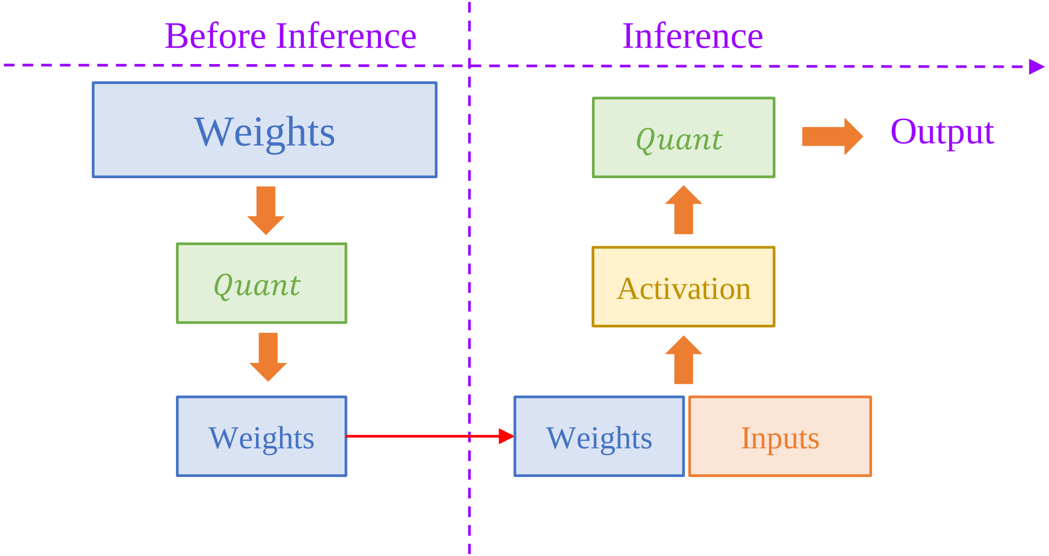
❖ Quantization types



Reference: https://pytorch.org/docs/stable/quantization.html#introduction-to-quantization

Background

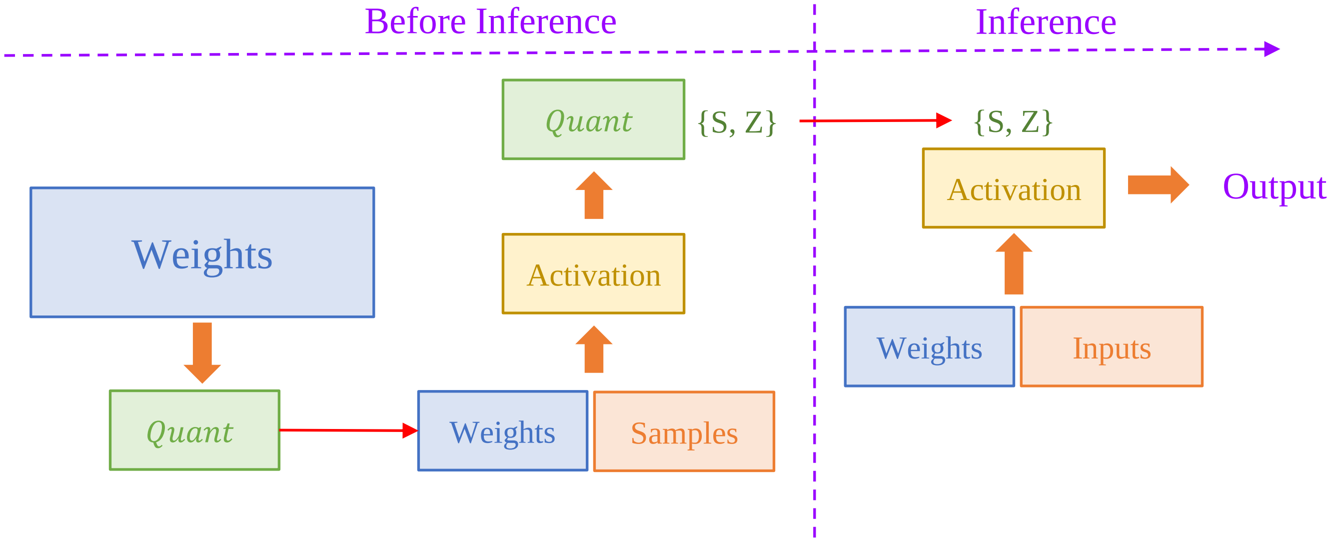
❖ Dynamic Quantization



Background

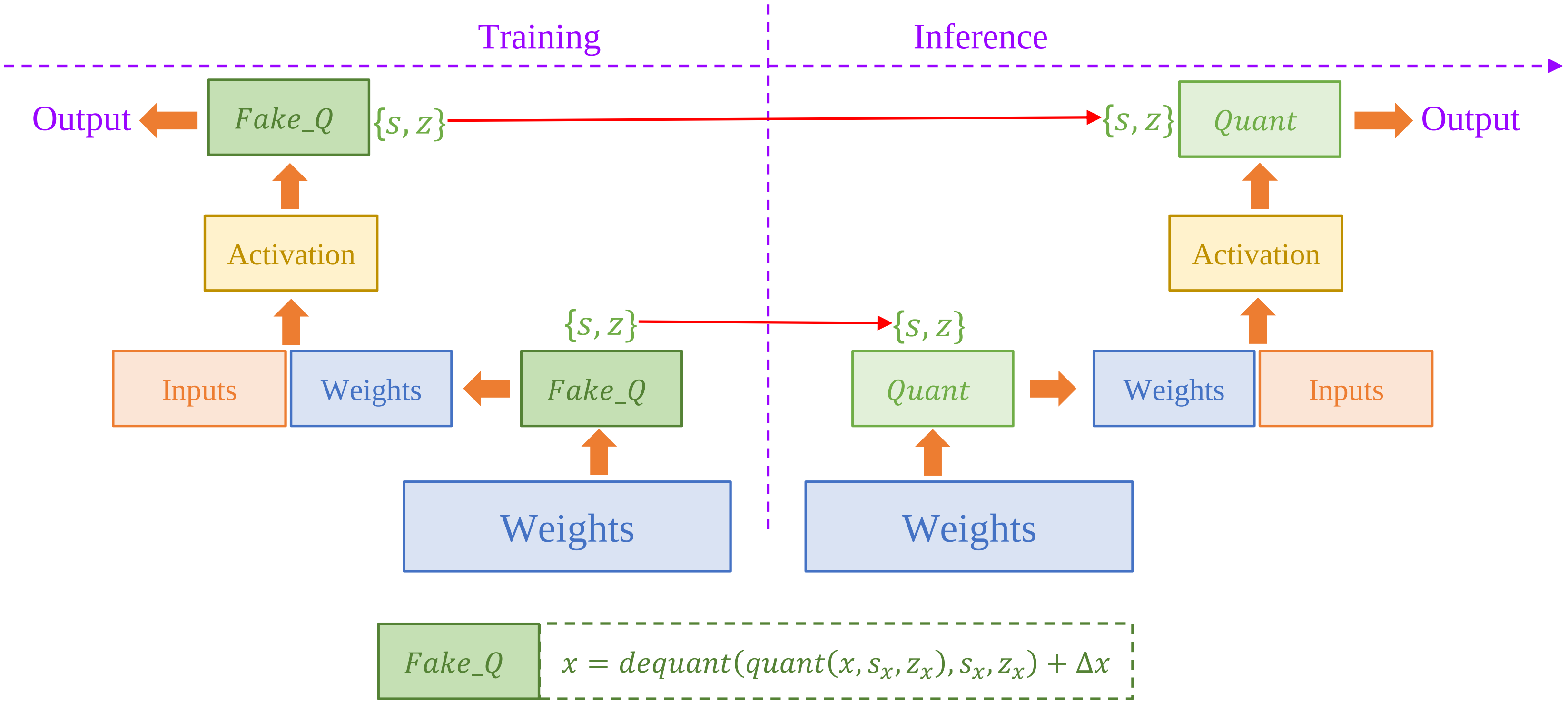
❖ Static/Post Quantization

{S: scale, Z: zero point}



Background

❖ Quantization aware training



Background

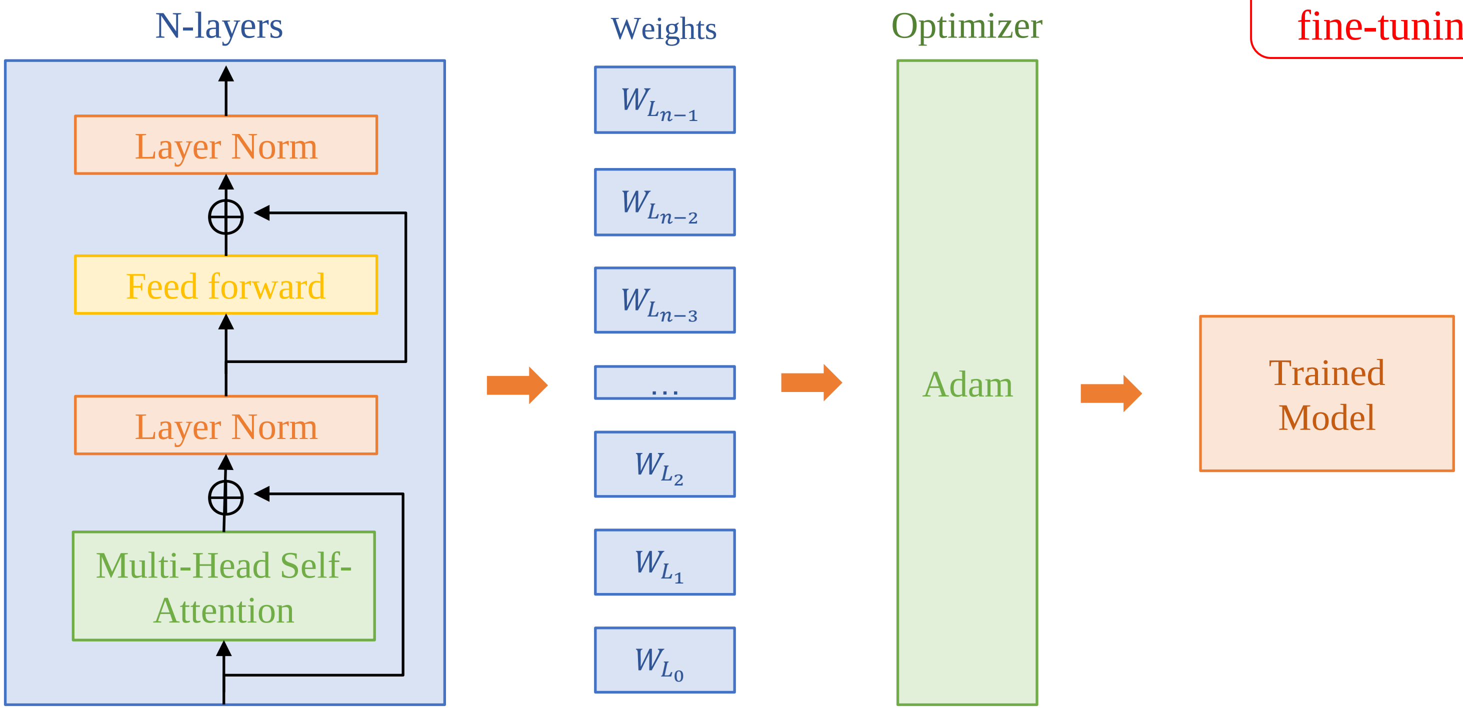
❖ Quantization Types

Types	Data requirements	Inference speed	Performance degradation
Dynamic Quantization	No data	Slow	Low
Static/Post Quantization	Unlabelled representative sample	Fast	High
Quantization aware training	Labelled training data	Fast	Low

Background

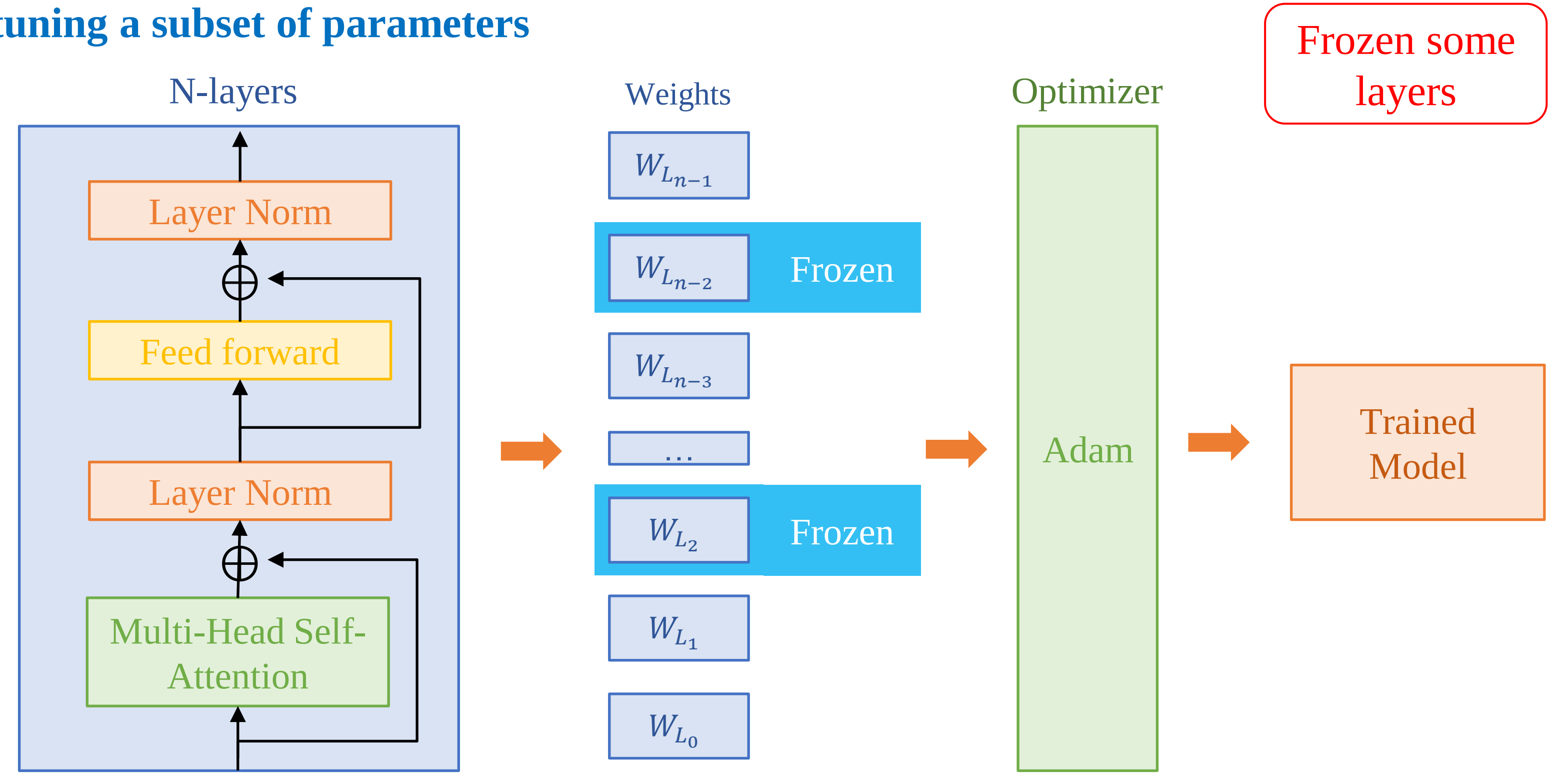
❖ Full Fine-tuning

Full layers fine-tuning



Background

❖ Fine-tuning a subset of parameters



Background

❖ LoRA: Low-rank Adaptation

Normal training

$$W = W_0 + \Delta W$$

$W_0 \in \mathbb{R}^{d \times k}$

$\Delta W \in \mathbb{R}^{d \times k}$

$d \times k$

LoRA training

$$h = Wx = W_0 x + \Delta W x$$

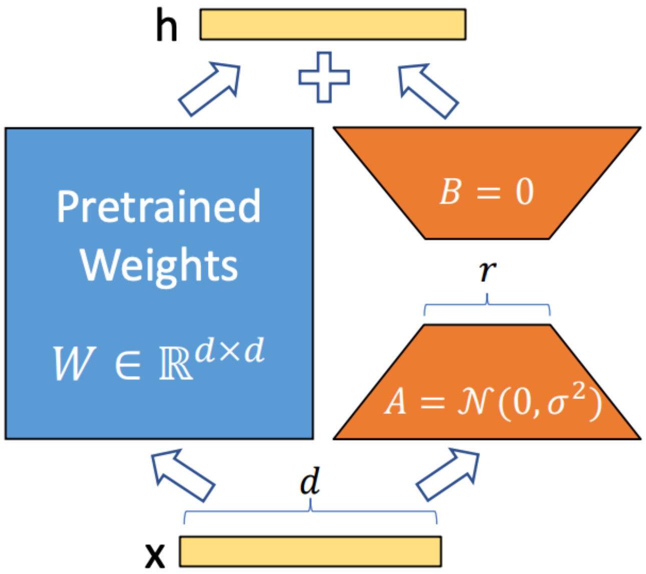
$$h = W_0 x + BAx$$

$B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$

$r \ll \{d, k\}$

$r \times (d + k)$

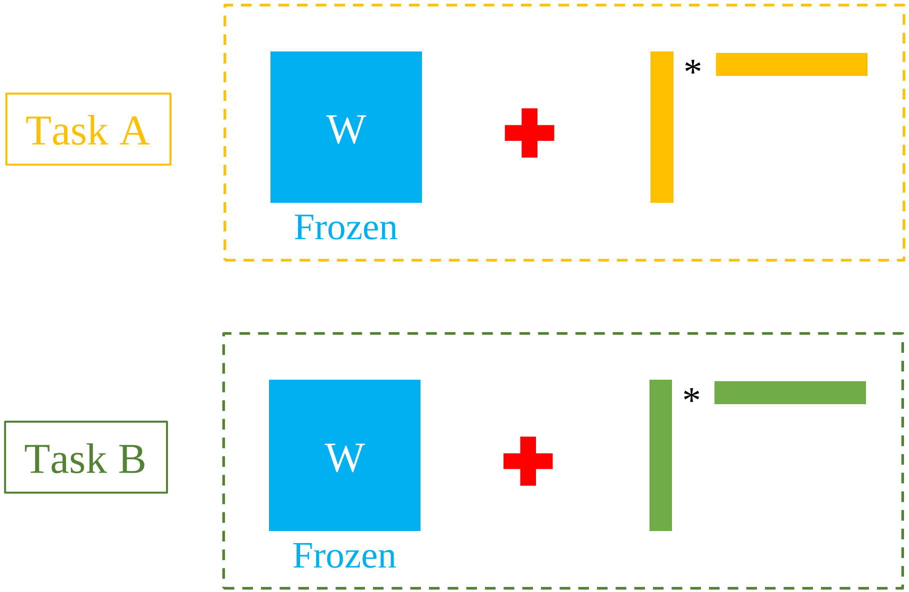
Trainable parameters

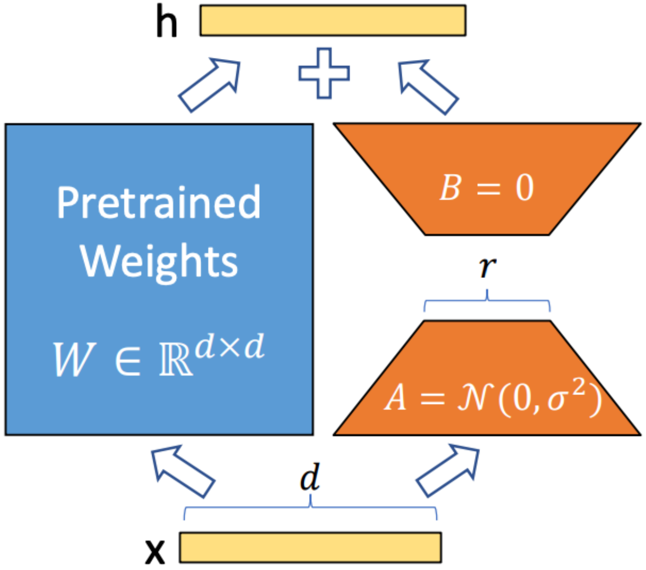


Reference: https://arxiv.org/abs/2106.09685

Background

❖ Switch to another task





Reference: https://arxiv.org/abs/2106.09685

Background

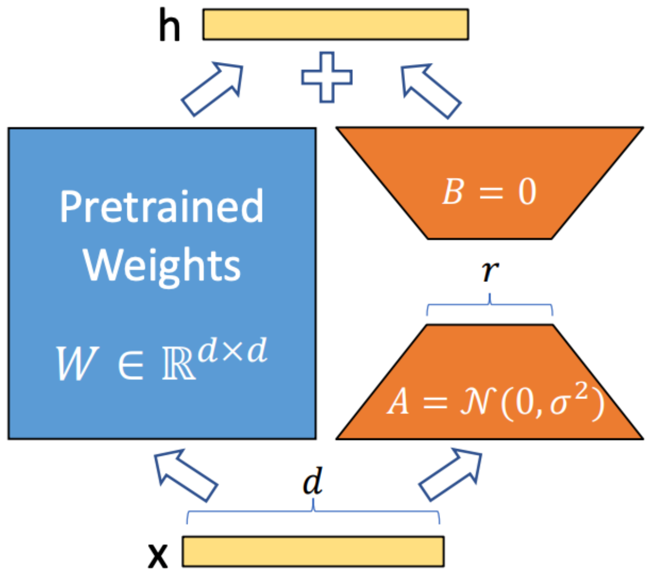
❖ LoRA: Low-rank Adaptatio of LLMs

$$GPT3\ Config = \begin{cases} layers = 96 \\ d_{model} = 12288 \end{cases}$$

$params: \sim 175B$

$$Trainable\ params = 2 \times d_{model} \times r \times \hat{L}_{LoRA}$$

$$r = 4$$
$$\Rightarrow params = 2 \times 12288 \times 4 \times (96 \times 4) \sim 37.7M$$



Reference: https://arxiv.org/abs/2106.09685

Background

❖ Result of LoRA in GPT-2/3

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{H})	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

"This suggests that the low-rank adaptation matrix potentially **amplifies** the **important features** for <u>specific downstream tasks</u> that were learned but not emphasized in the general pre-training model."

Reference: https://arxiv.org/abs/2106.09685

Background

❖ What is the optimal rank (r) for LoRA

	Weight Type	$r=1$	$r=2$	$r=4$	$r=8$	$r=64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Directions corresponding to the top singular vector overlap significantly between $A_{r=8}$ and $A_{r=64}$, while others do not. Specifically, ΔW_v (resp. ΔW_q) of $A_{r=8}$ and ΔW_v (resp. ΔW_q) of $A_{r=64}$ share a subspace of dimension 1 with normalized similarity > 0.5, providing an explanation of why $r = 1$ performs quite well in our downstream tasks for GPT-3.

Reference: https://arxiv.org/abs/2106.09685

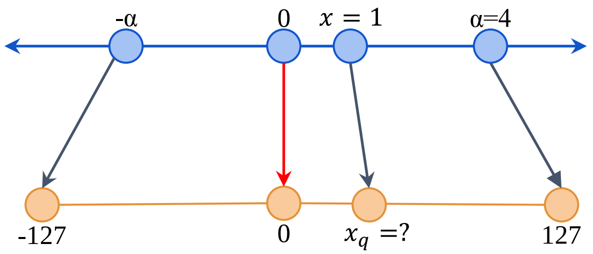
Methods

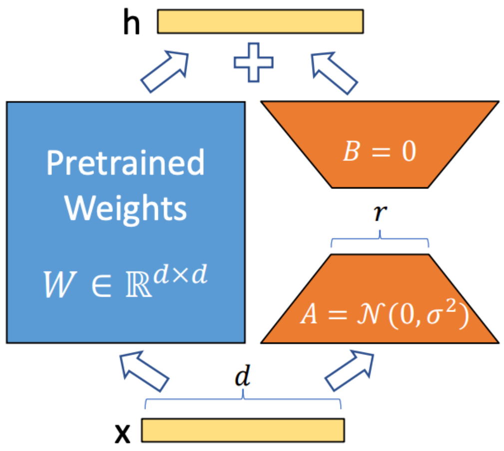
- ❖ Introduction
- ❖ QLoRA
- ❖ 4-bit Quantization
- ❖ Double Quantization

Methods

❖ Introduction

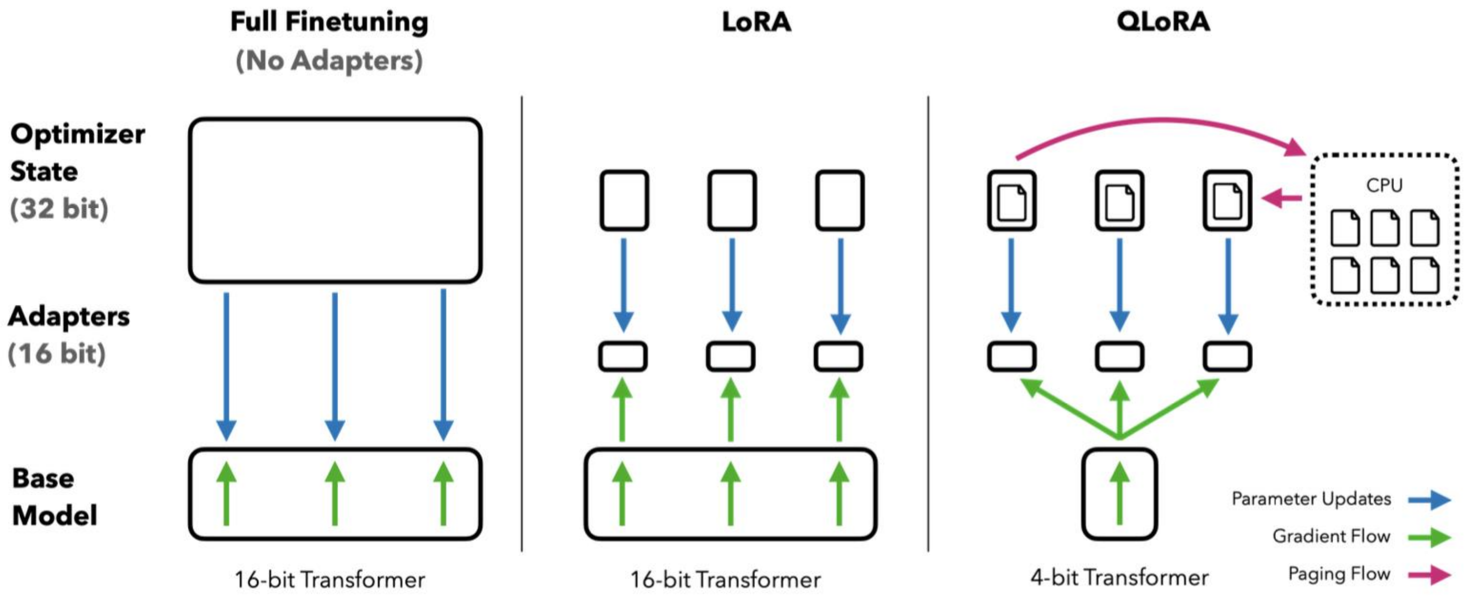
QLoRA = Quantization + LoRA





Methods

❖ QLoRa



AI VIETNAM
Seminar

Methods

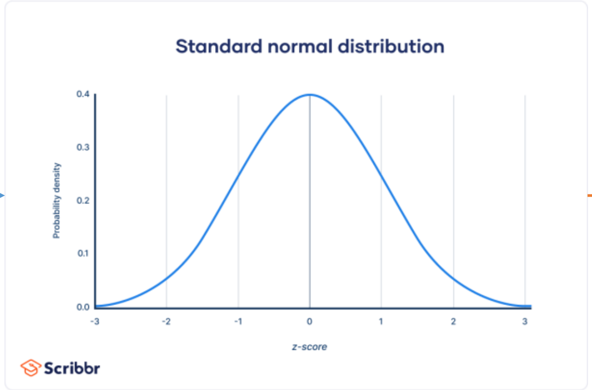
❖ **Block-wise k-bit Quantization**

$$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}})$$

$$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}}$$

Methods

❖ 4-bit NormalFloat Quantization

Weights → 

→ Normalize → [-1, 1]

$$q_i = \frac{1}{2}\left(Q_X\left(\frac{i}{2^k+1}\right) + Q_X\left(\frac{i+1}{2^k+1}\right)\right),$$

Methods

❖ Double Quantization (DQ)

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{L}_1^{\text{BF16}}\mathbf{L}_2^{\text{BF16}},$$

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}},$$

Evaluation

Evaluation

❖ Guanaco with QLoRA

Table 6: Zero-shot Vicuna benchmark scores as a percentage of the score obtained by ChatGPT evaluated by GPT-4. We see that OASST1 models perform close to ChatGPT despite being trained on a very small dataset and having a fraction of the memory requirement of baseline models.

Model / Dataset	Params	Model bits	Memory	ChatGPT vs Sys	Sys vs ChatGPT	Mean	95% CI
GPT-4	-	-	-	119.4%	110.1%	**114.5%**	2.6%
Bard	-	-	-	93.2%	96.4%	94.8%	4.1%
Guanaco	65B	4-bit	41 GB	96.7%	101.9%	**99.3%**	4.4%
Alpaca	65B	4-bit	41 GB	63.0%	77.9%	70.7%	4.3%
FLAN v2	65B	4-bit	41 GB	37.0%	59.6%	48.4%	4.6%
Guanaco	33B	4-bit	21 GB	96.5%	99.2%	**97.8%**	4.4%
Open Assistant	33B	16-bit	66 GB	91.2%	98.7%	94.9%	4.5%
Alpaca	33B	4-bit	21 GB	67.2%	79.7%	73.6%	4.2%
FLAN v2	33B	4-bit	21 GB	26.3%	49.7%	38.0%	3.9%
Vicuna	13B	16-bit	26 GB	91.2%	98.7%	**94.9%**	4.5%
Guanaco	13B	4-bit	10 GB	87.3%	93.4%	90.4%	5.2%
Alpaca	13B	4-bit	10 GB	63.8%	76.7%	69.4%	4.2%
HH-RLHF	13B	4-bit	10 GB	55.5%	69.1%	62.5%	4.7%
Unnatural Instr.	13B	4-bit	10 GB	50.6%	69.8%	60.5%	4.2%
Chip2	13B	4-bit	10 GB	49.2%	69.3%	59.5%	4.7%
Longform	13B	4-bit	10 GB	44.9%	62.0%	53.6%	5.2%
Self-Instruct	13B	4-bit	10 GB	38.0%	60.5%	49.1%	4.6%
FLAN v2	13B	4-bit	10 GB	32.4%	61.2%	47.0%	3.6%
Guanaco	7B	4-bit	5 GB	84.1%	89.8%	**87.0%**	5.4%
Alpaca	7B	4-bit	5 GB	57.3%	71.2%	64.4%	5.0%
FLAN v2	7B	4-bit	5 GB	33.3%	56.1%	44.8%	4.0%

Conclusion

Conclusion

Advantages	Limitations
• 4-bit finetuning with LoRA replicate 16-bit full finetuning. • QLoRA + Guanaco – archive State-of-the-art performance AI chatbot.	• Can not evaluation with difference bit (such as 3-bit). • Cannot marger with difference adapter