QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman,
Luke Zettlemoyer
University of Washington

Structure

- Motivation
- Background
  - LoRA
  - Quantization
- Method
  - 4-bit NormalFloat
  - Double quantization
  - Page optimizers
- Results
- Thoughts

Problem with finetuning?

Model	Fine-tuning memory
T5-11B	132 GB
Mistral-7B	84 GB
LLaMA2-70B	840 GB

QLoRA reduces the memory for finetuning by 15-20x!

Model	Fine-tuning memory
T5-11B	132 GB
Mistral-7B	84 GB
LLaMA2-70B	840 GB

QLoRA

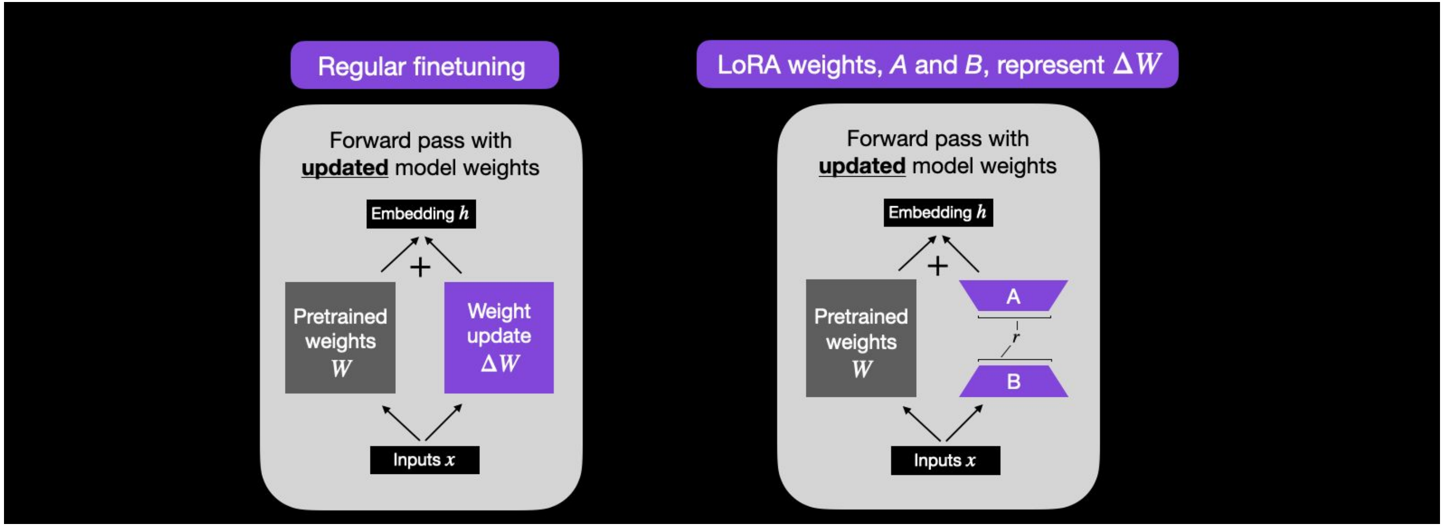
Model	Fine-tuning memory
T5-11B	6 GB
Mistral-7B	5 GB
LLaMA2-70B	46 GB

QLoRA

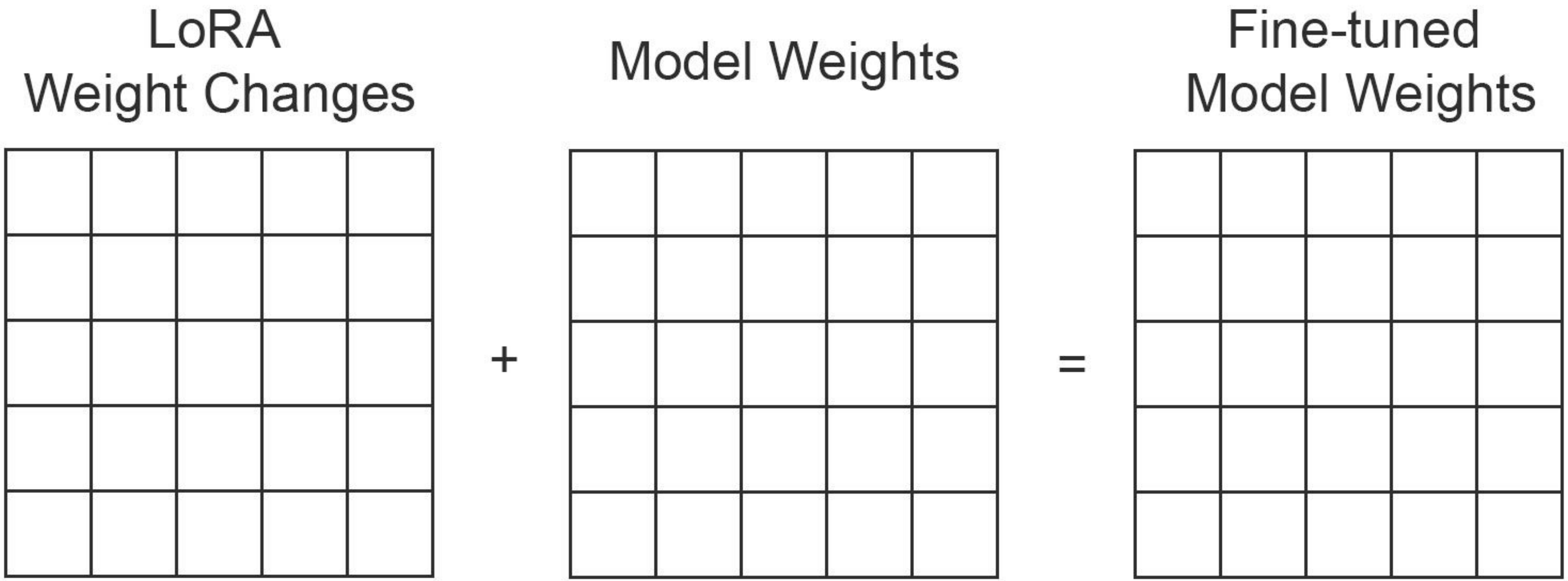
- LoRA + Quantization
- LoRA
  - Method for finetuning
- Quantization
  - Reduce memory footprint

LoRA

- Instead of updating weights directly, we track changes
- Track weight changes in two separate, smaller matrices that get multiplied together to form a matrix that is the same size of the model's weight matrix



LoRA



LoRA
Weight Changes
+
Model Weights
=
Fine-tuned
Model Weights

LoRA

- Matrix multiplication

$$
\begin{bmatrix} 1 \\ 3 \\ 7 \\ -4 \\ 2 \end{bmatrix}
\times
\begin{bmatrix} 5 & 1 & -1 & 3 & 4 \end{bmatrix}
=
\begin{bmatrix}
5 & 1 & -1 & 3 & 4 \\
15 & 3 & -3 & 9 & 12 \\
35 & 7 & -7 & 21 & 28 \\
-20 & -4 & 4 & -12 & -16 \\
10 & 2 & -2 & 6 & 8
\end{bmatrix}
$$

LoRA

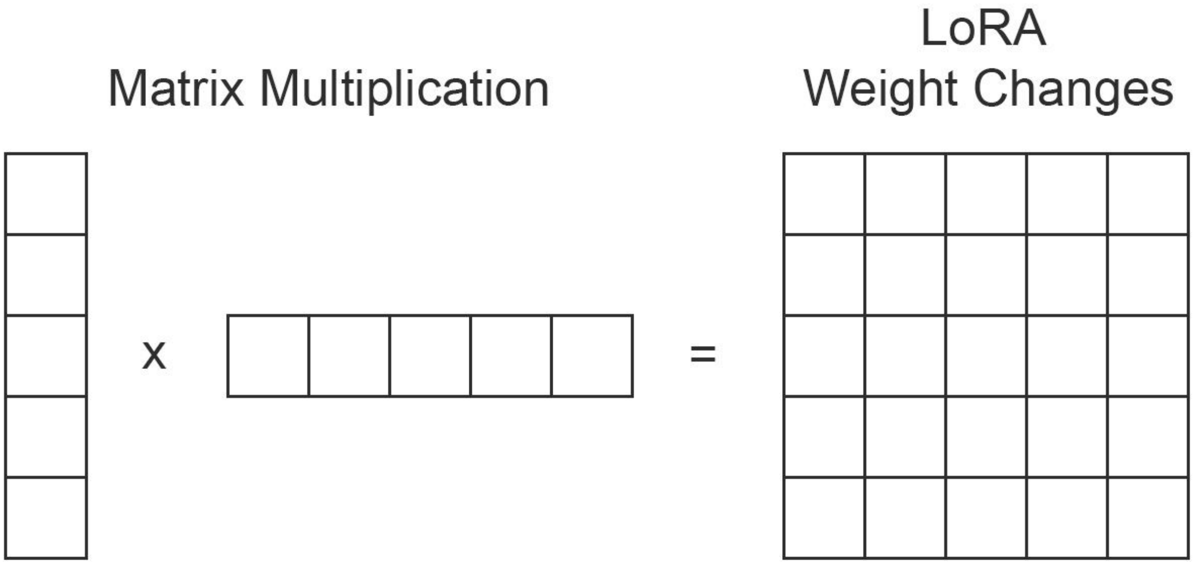
- Matrix decomposition

$$
\begin{bmatrix} 1 \\ 3 \\ 7 \\ -4 \\ 2 \end{bmatrix} \times \begin{bmatrix} 5 & 1 & -1 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 1 & -1 & 3 & 4 \\ 15 & 3 & -3 & 9 & 12 \\ 35 & 7 & -7 & 21 & 28 \\ -20 & -4 & 4 & -12 & -16 \\ 10 & 2 & -2 & 6 & 8 \end{bmatrix}
$$

$$
\boxed{\begin{bmatrix} 5 & 1 & -1 & 3 & 4 \\ 15 & 3 & -3 & 9 & 12 \\ 35 & 7 & -7 & 21 & 28 \\ -20 & -4 & 4 & -12 & -16 \\ 10 & 2 & -2 & 6 & 8 \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \times \begin{bmatrix} ? & ? & ? & ? & ? \end{bmatrix}}
$$

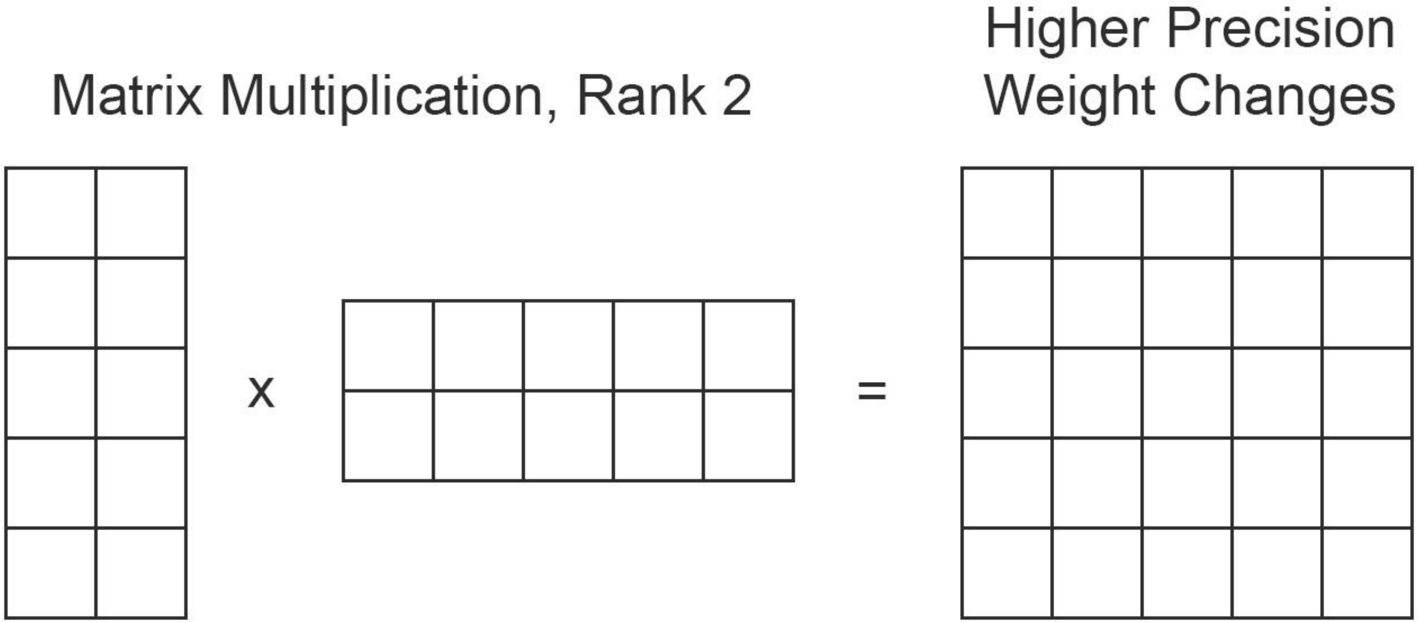
LoRA

Matrix decomposition (rank 1)



LoRA

Matrix decomposition



LoRA

# Total Paramete rs	Full Matrix Dimensions	Parameters in Decomposed Matrices (Rank 1)	Relative Number of Values
25	5x5	10	40%
100	10x10	20	20%
2.5k	50x50	100	4%
1M	1k x 1k	2k	0.2%
13B	114k x 114k	228k	0.001%

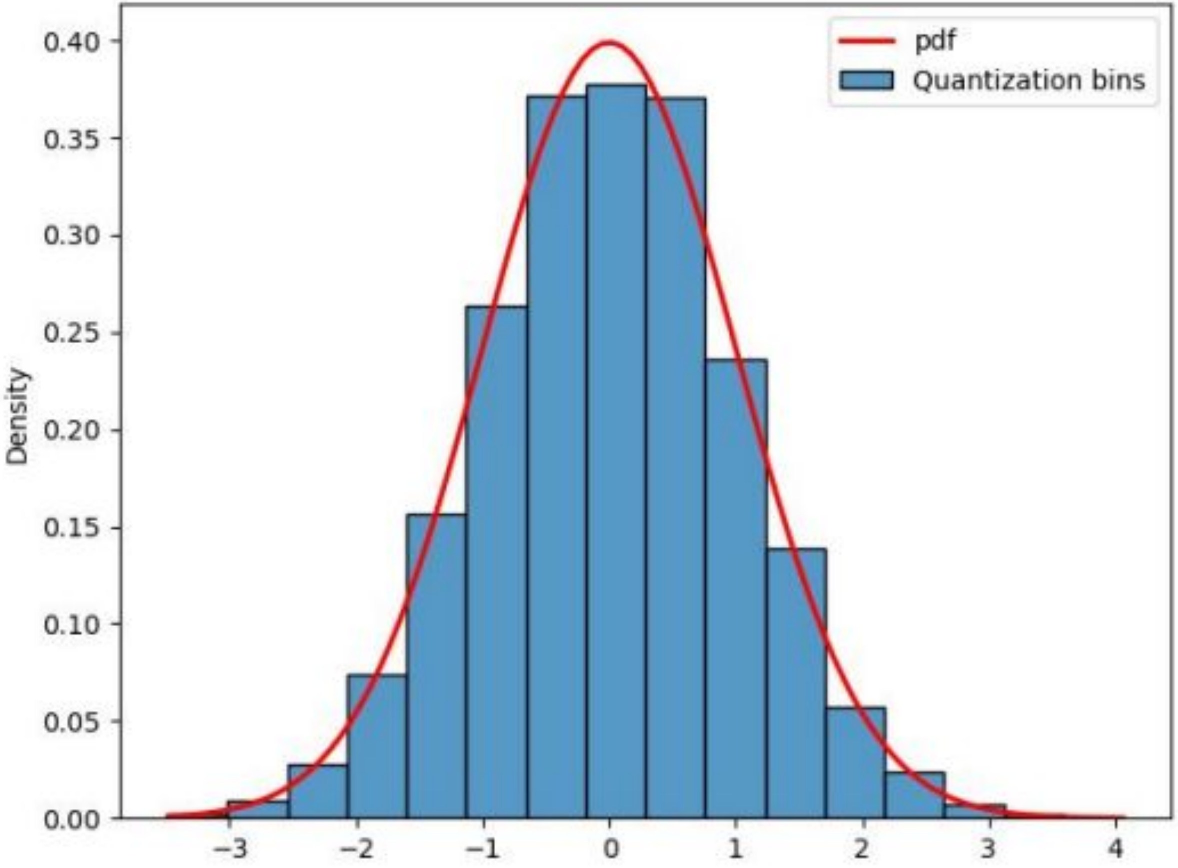
Simple Implementation

```python
class LoRALayer(torch.nn.Module):
    def __init__(self, in_dim, out_dim, rank, alpha):
        super().__init__()
        std_dev = 1 / torch.sqrt(torch.tensor(rank).float())
        self.A = torch.nn.Parameter(torch.randn(in_dim, rank) * std_dev)
        self.B = torch.nn.Parameter(torch.zeros(rank, out_dim))
        self.alpha = alpha

    def forward(self, x):
        x = self.alpha * (x @ self.A @ self.B)
        return x
```

```python
class LinearWithLoRA(torch.nn.Module):
    def __init__(self, linear, rank, alpha):
        super().__init__()
        self.linear = linear
        self.lora = LoRALayer(
            linear.in_features, linear.out_features, rank, alpha
        )

    def forward(self, x):
        return self.linear(x) + self.lora(x)
```

Quantization

- INT4 example



Quantization Example: A non-standard 2-bit data type

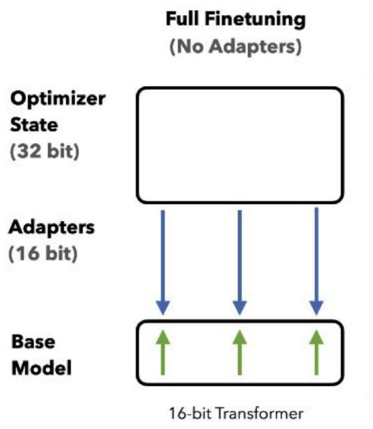
Map: {Index: 0, 1, 2, 3 -> Values: -1.0, 0.3, 0.5, 1.0}

Input tensor: [10, -3, 5, 4]

1. Normalize with absmax: [10, -3, 5, 4] -> [1, -0.3, 0.5, 0.4]
2. Find closest value: [1, -0.3, 0.5, 0.4] -> [1.0, 0.3, 0.5, 0.5]
3. Find the associated index: [1.0, 0.3, 0.5, 0.5] -> [3, 1, 2, 2] -> store
4. Dequantization: load -> [3, 1, 2, 2] -> lookup -> [1.0, 0.3, 0.5, 0.5] -> denormalize -> [10, 3, 5, 5]

Full Finetuning

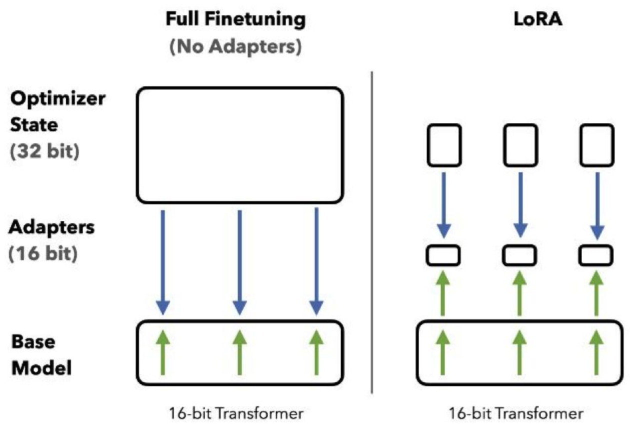
- Finetuning cost per parameter:
- Weight: 16 bits
    - Weight gradient: 16 bit
    - Optimizer state: 64 bit
    - 12 bytes per parameter
    -



70B model -> 840 GB of GPU memory -> 36x consumer GPUs

LoRA is not enough

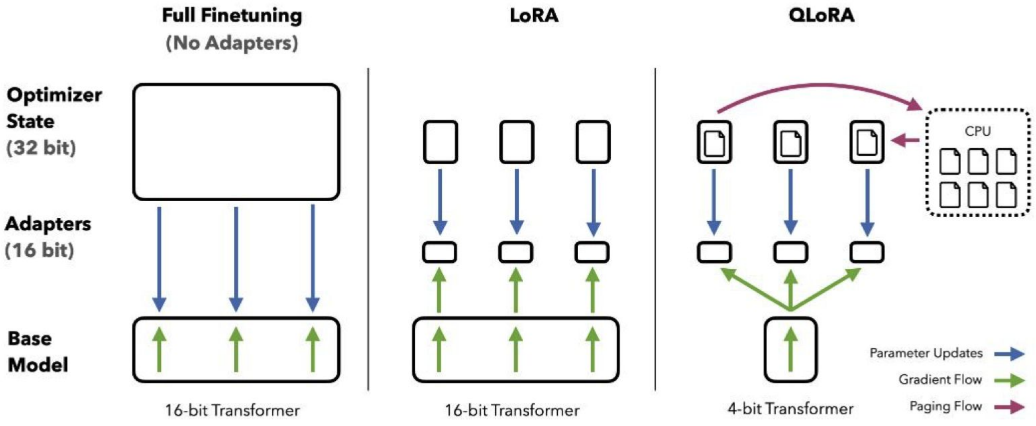
- Finetuning cost per parameter:
- Weight: 16 bits
  - Weight gradient: ~0.4bit
  - Optimizer state: ~0.8bit
  - Adapter weights: ~0.4bit
  - 17.6 bits per parameter



70B model -> 154 GB of GPU memory -> 8x consumer GPU

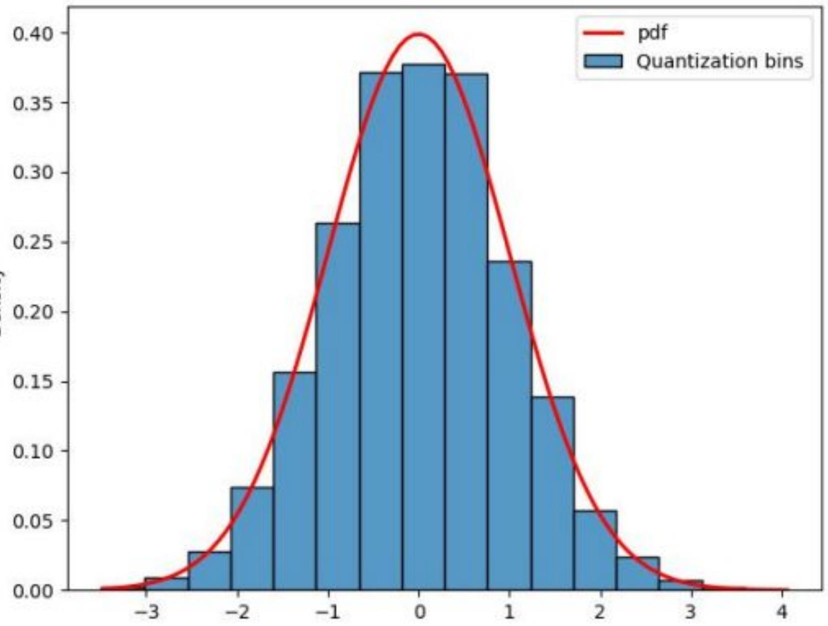
QLoRA: 4 bit frozen model + low rank adapters

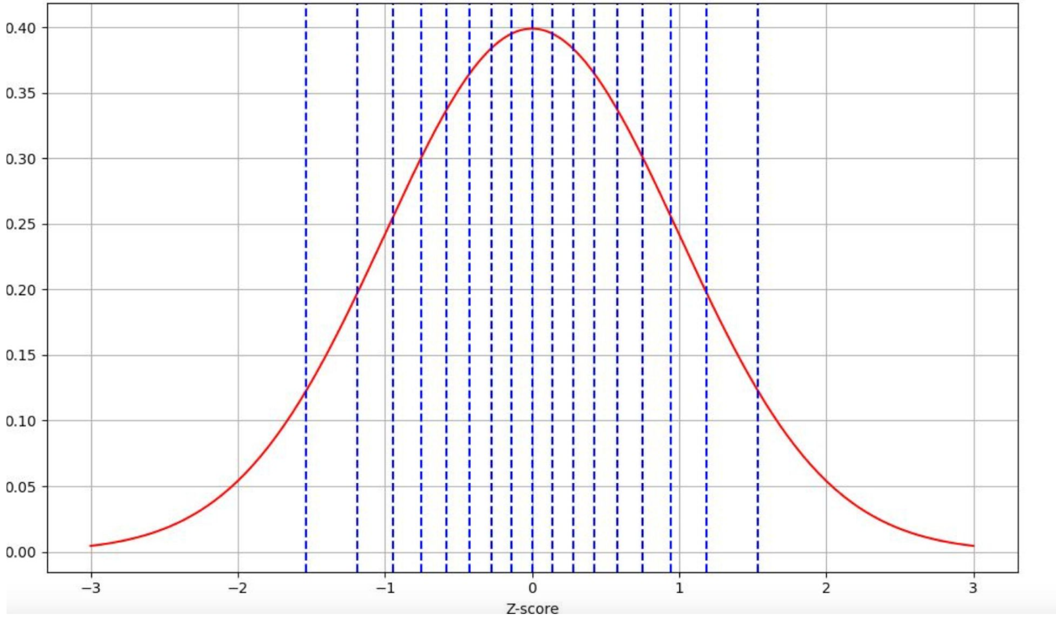
- Finetuning cost per parameter:
- Weight: 4 bits
    - Weight gradient: ~0.4bit
    - Optimizer state: ~0.8bit
    - Adapter weights: ~0.4bit
    - 5.2 bits per parameter



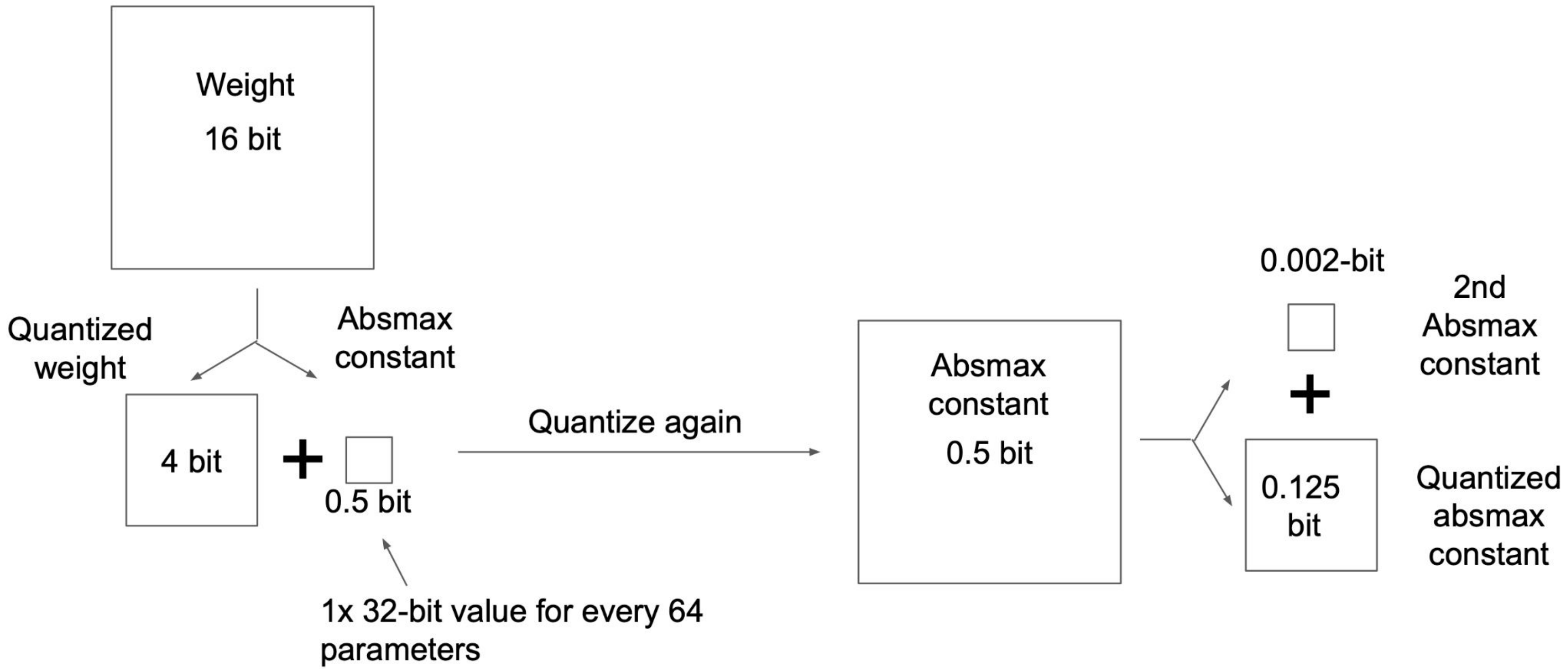
70B model -> 46 GB of GPU memory -> 2x consumer GPUs.

New datatype: 4-bit NormalFloat (NF4)



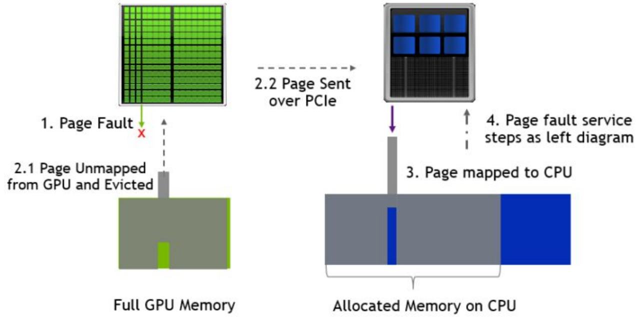


Second contribution: Double quantization



Page Optimizers (Unified memory)

- Manage memory by page transfers between CPU <-> GPU automatically (like os paging)
- High level
  - Bigger batch uses large GPU memory
  - Paging engine evicts optimizer state to CPU
  - During optimizer step, prefetch from CPU to GPU
  - Perform optimizer step

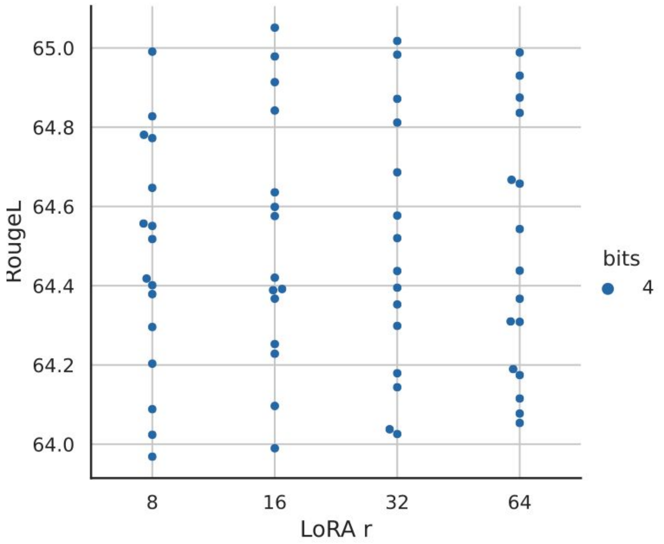


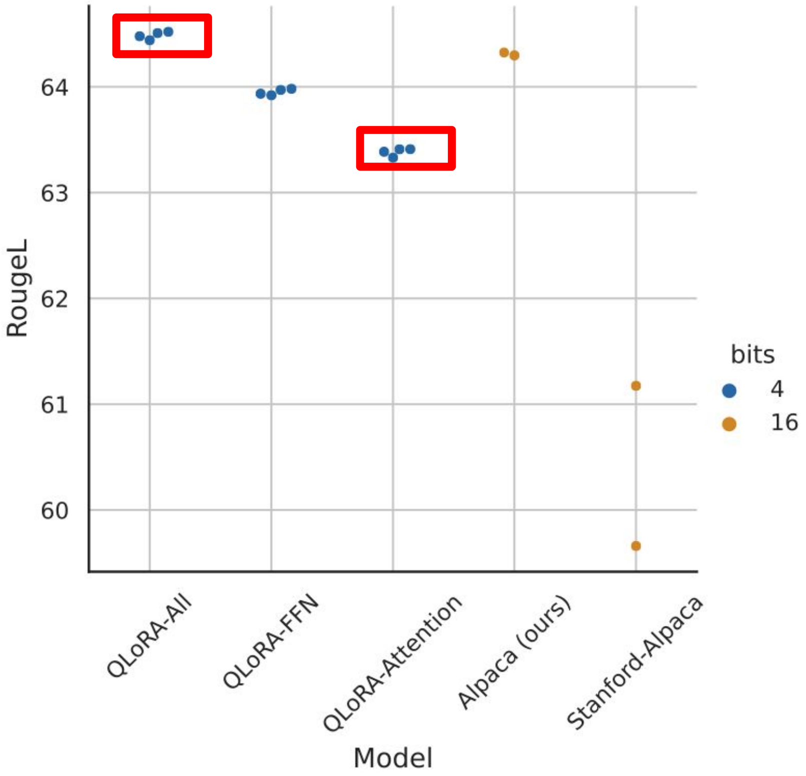
Evaluation

- MMLU (Massively Multitask Language Understanding)
  - Multiple choice benchmark
- Chatbots (ELO system)
- Trained Guanaco (finetuned on OASST1)

Default hyperparameters for LoRA do not work

- Rank does not matter
- Number of LoRA adapters matter





4bit normal float works!

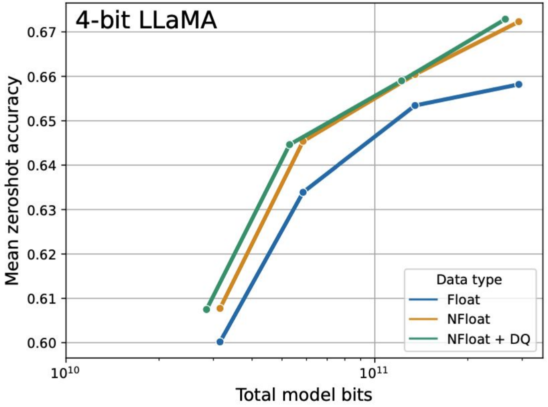


Figure 3: Mean zero-shot accuracy over Winogrande, HellaSwag, PiQA, Arc-Easy, and Arc-Challenge using LLaMA models with different 4-bit data types. The NormalFloat data type significantly improves the bit-for-bit accuracy gains compared to regular 4-bit Floats. While Double Quantization (DQ) only leads to minor gains, it allows for a more fine-grained control over the memory footprint to fit models of certain size (33B/65B) into certain GPUs (24/48GB).

Table 3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

Table 4: Mean 5-shot MMLU test accuracy for LLaMA 7-65B models finetuned with adapters on Alpaca and FLAN v2 for different data types. Overall, NF4 with double quantization (DQ) matches BFloat16 performance, while FP4 is consistently one percentage point behind both.

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

MMLU dataset (multiple choice reasoning)

Table 5: MMLU 5-shot test results for different sizes of LLaMA finetuned on the corresponding datasets using QLoRA.

Dataset	7B	13B	33B	65B
LLaMA no tuning	35.1	46.9	57.8	63.4
Self-Instruct	36.4	33.3	53.0	56.7
Longform	32.1	43.2	56.6	59.7
Chip2	34.5	41.6	53.6	59.8
HH-RLHF	34.9	44.6	55.8	60.1
Unnatural Instruct	41.9	48.1	57.3	61.3
Guanaco (OASST1)	36.6	46.4	57.0	62.2
Alpaca	38.8	47.8	57.3	62.5
FLAN v2	44.5	51.4	59.2	63.9

Vicuna chatbot benchmark (tournament)

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%

Vicuna chatbot benchmark

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%

Tournament

Table 7: Elo rating for a tournament between models where models compete to generate the best response for a prompt, judged by human raters or GPT-4. Overall, Guanaco 65B and 33B tend to be preferred to ChatGPT-3.5 on the benchmarks studied. According to human raters they have a Each 10-point difference in Elo is approximately a difference of 1.5% in win-rate.

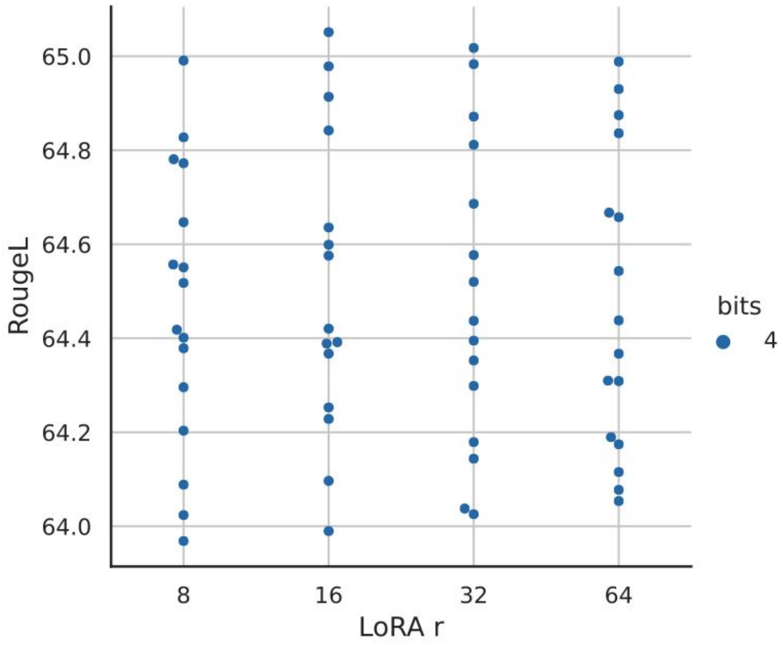
Benchmark	Vicuna		Vicuna		Open Assistant		
# Prompts	80		80		953		
Judge	Human raters		GPT-4		GPT-4		Median Rank
Model	Elo	Rank	Elo	Rank	Elo	Rank	
GPT-4	1176	1	1348	1	1294	1	1
Guanaco-65B	1023	2	1022	2	1008	3	2
Guanaco-33B	1009	4	992	3	1002	4	4
ChatGPT-3.5 Turbo	916	7	966	5	1015	2	5
Vicuna-13B	984	5	974	4	936	5	5
Guanaco-13B	975	6	913	6	885	6	6
Guanaco-7B	1010	3	879	8	860	7	7
Bard	909	8	902	7	-	-	8

Findings, Strengths and others

- Dataset suitably matters a lot
  - Some datasets affect quality of chatbots
- Strengths
  - Allow consumers to finetune LLMs on their own hardware even on phones
- Could this be applied to multimodal/MoE models and how well do they perform?

Questions about decisions

- Why did the authors decide on r = 16/64 for eval? (could set r = 8)?



Conclusion

- Able to finetune large language models with way less memory
  - NF4 quantization
  - Double quantization
  - Page optimizers