QLoRA: Efficient Finetuning of Quantized LLMs

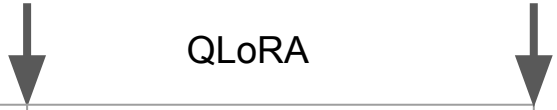
Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer
https://github.com/TimDettmers/bitsandbytes
https://timdettmers.com

Large models are not easily accessible

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

Large models are not easily accessible

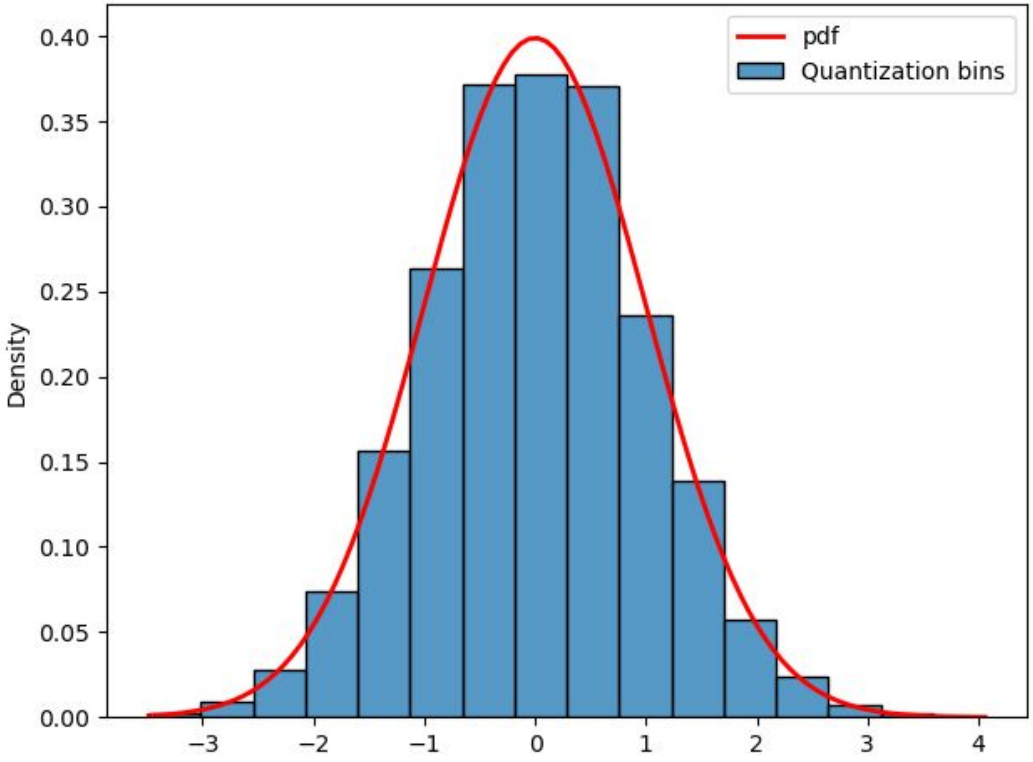
Model	Fine-tuning memory
T5-11B	132 GB
Mistral-7B	84 GB
LLaMA2-70B	840 GB



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

Background

How does quantization work?



Data types are mappings from symbols (bits) to numbers (floats), can we generalize?

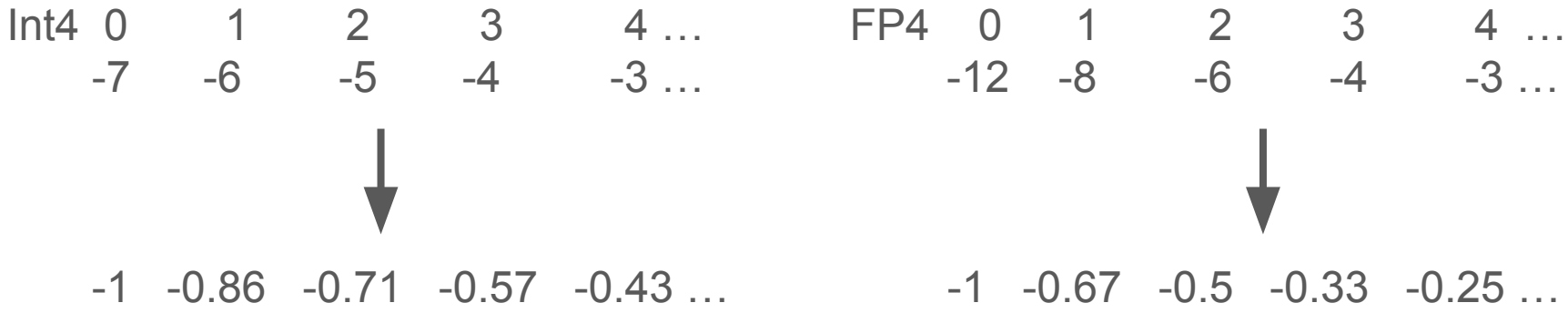
Most general form of describe quantization is through a mapping from integers to float values normalized to the range -1.0 and 1.0.

Int4	0	1	2	3	4 …
	-7	-6	-5	-4	-3 …

FP4	0	1	2	3	4 …
	-12	-8	-6	-4	-3 …

Quantization as a mapping

Most general form of describe quantization is through a mapping from integers to float values normalized to the range -1.0 and 1.0.



The mapping format { index : float value} generalizes to all data types.

Recipe: How to quantize a tensor?

Given a tensor X of any real data type. We can apply quantization as follows:

1. Normalize X into the range [-1.0, 1.0] by dividing by absmax(X)
2. Find the closest value in the data type (rounding for integers; in general binary search)

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

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

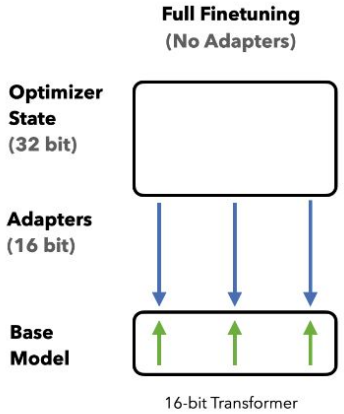
Input tensor: [10, -3, 5, 4]

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]

Finetuning is expensive.

Finetuning cost per parameter:

- Weight: 16 bit
- Weight gradient: 16 bit
- Optimizer state: 64 bit
- 12 byte per parameter

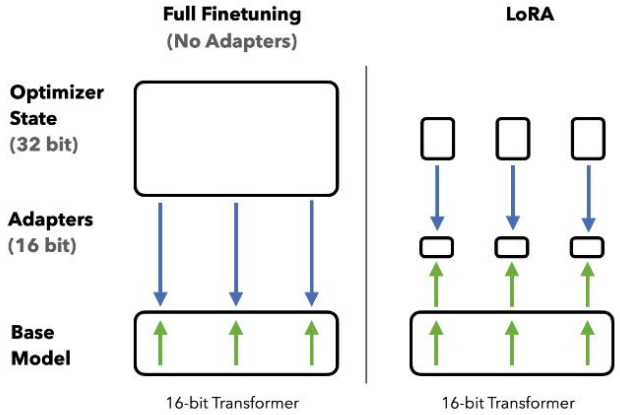


70B model -> 840 GB of GPU memory -> 36x consumer GPUs

Finetuning with Low Rank Adapters (LoRA).

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 GPUs

QLoRA

QLoRA: 4-bit frozen base model + Low rank Adapters

Finetuning cost per parameter:

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

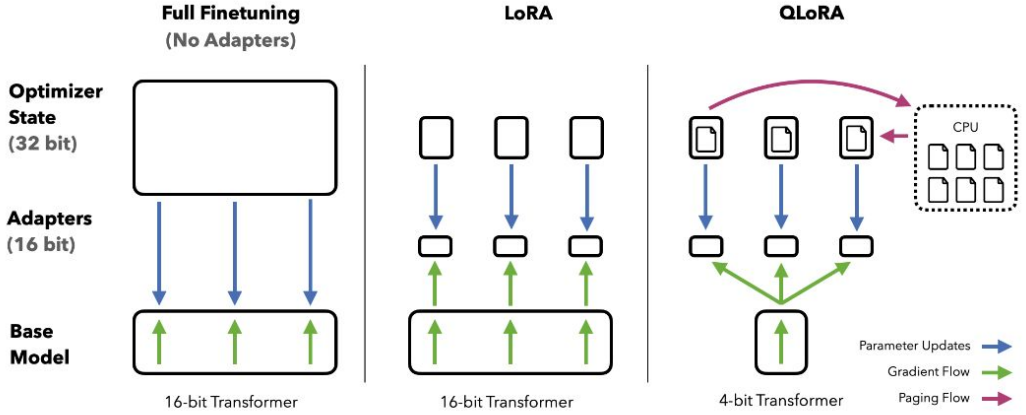
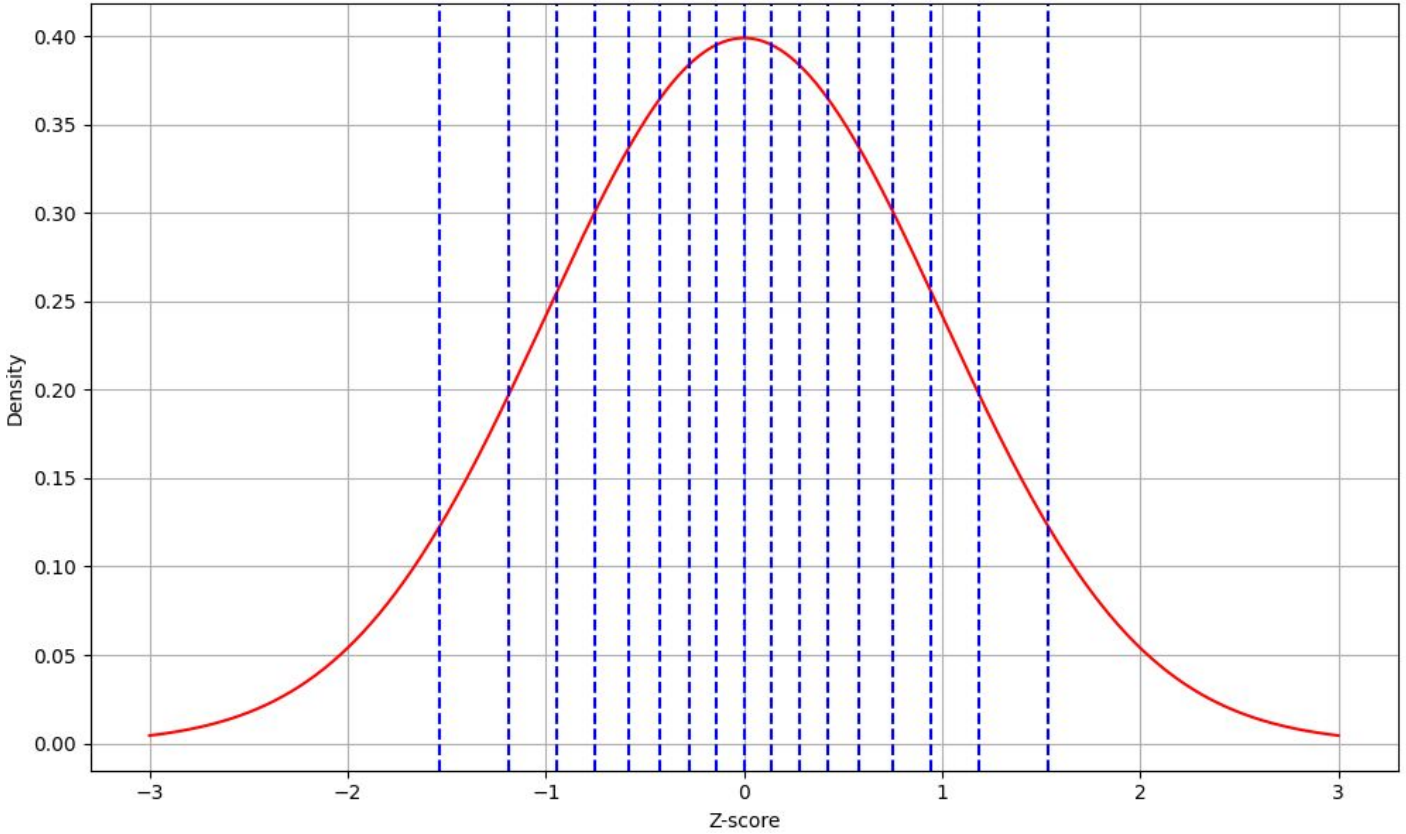


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

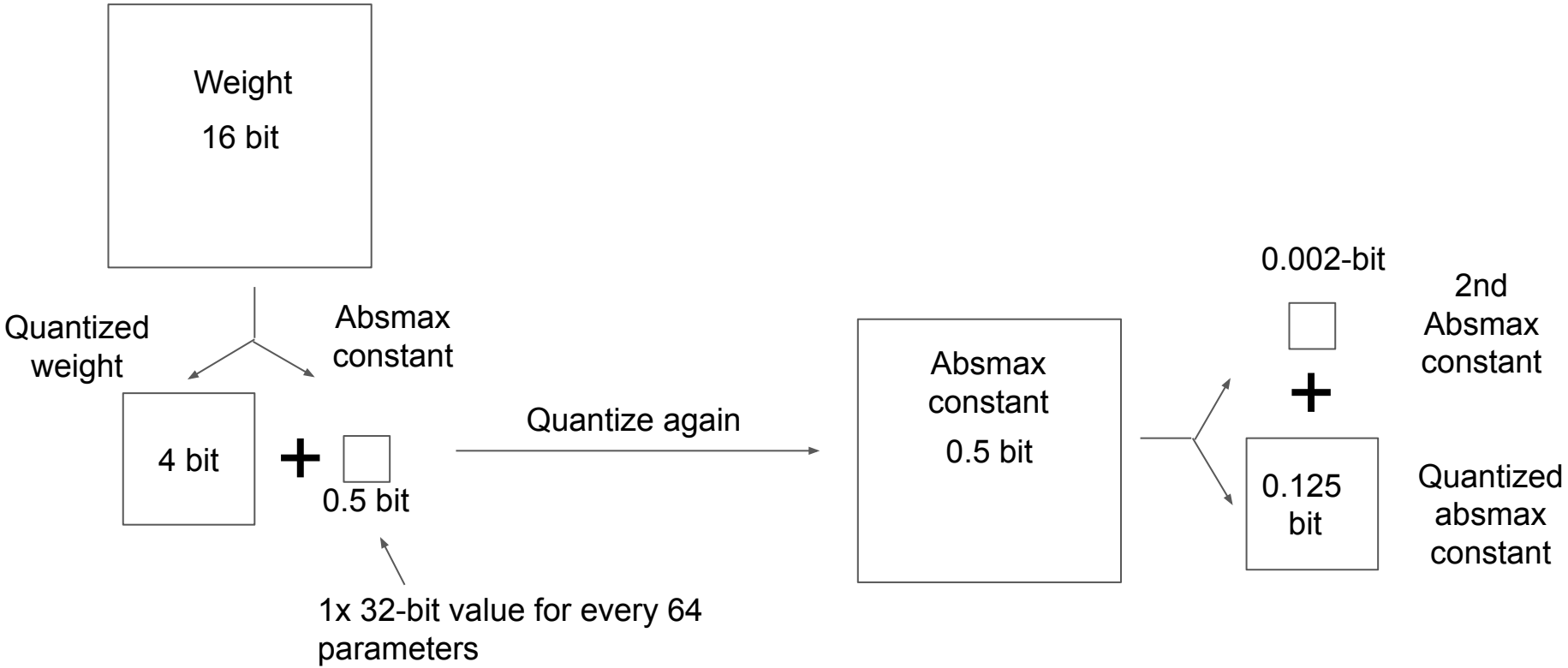
70B model -> 46 GB of GPU memory -> 2x consumer GPUs.

Saving memory while preserving fine-tuning quality

4-bit NormalFloat (NF4) an information-theoretically optimal data type for normal distributions

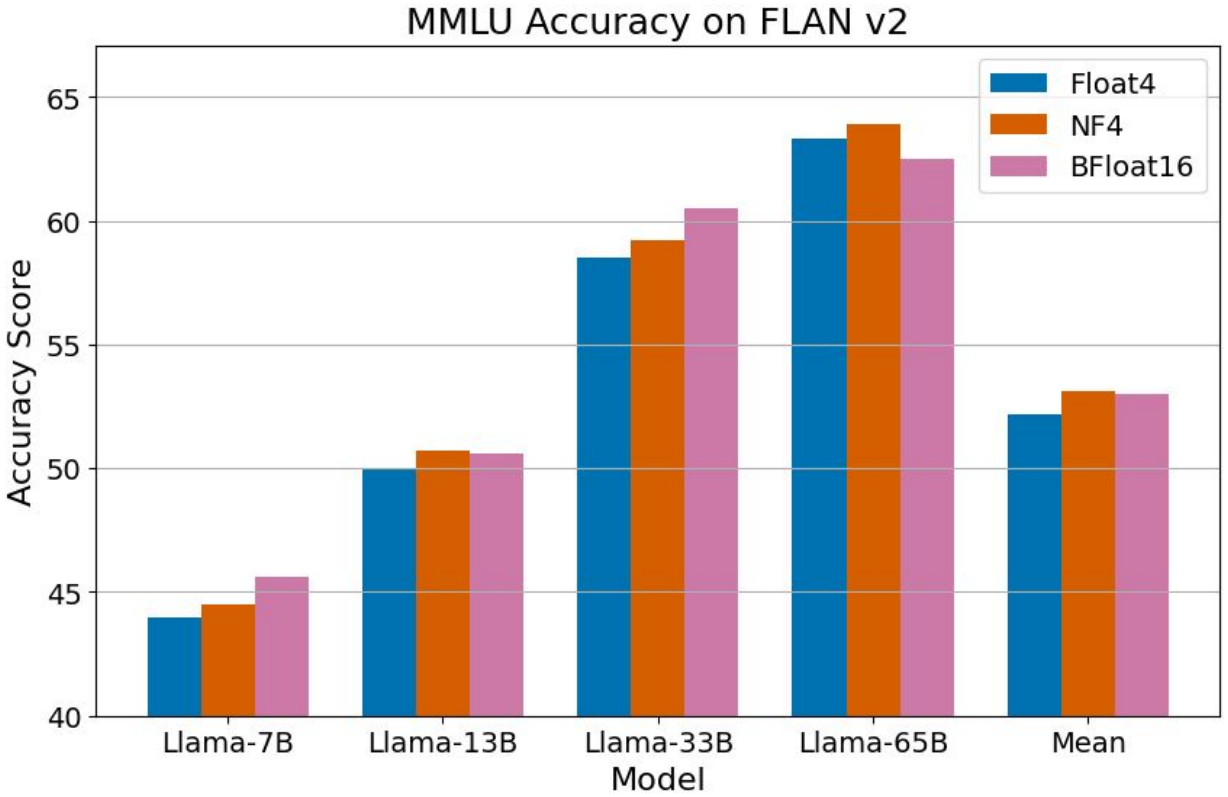


Reduce absmax constant size with Double Quantization

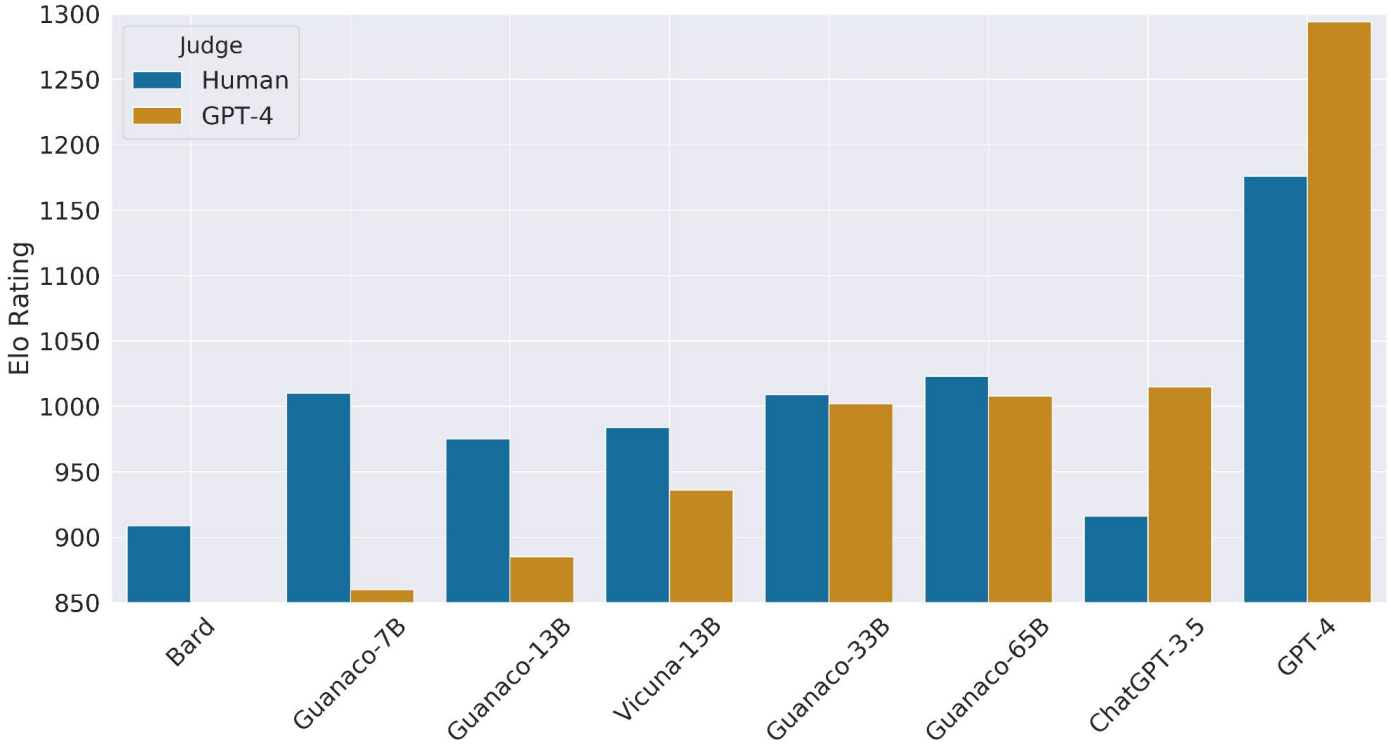


Results

QLoRA recovers lost performance through fine-tuning



4-bit Guanaco: A ChatGPT-quality 4-bit chatbot finetuned in 24h on a single GPU



Demo

Conclusion

- QLoRA makes finetuning 18x cheaper
- 4-bit NormalFloat (NF4) replicates 16-bit finetuning performance
- 4-bit chatbots created with QLoRA can be competitive with ChatGPT

QLoRA is available through the bitsandbytes Python library and fully integrated into the HuggingFace transformers stack.

I am on the academic job market — please get in touch:
dettmers@cs.washington.edu