Quantization

Abhinav Bhatele, Daniel Nichols

UNIVERSITY OF MARYLAND

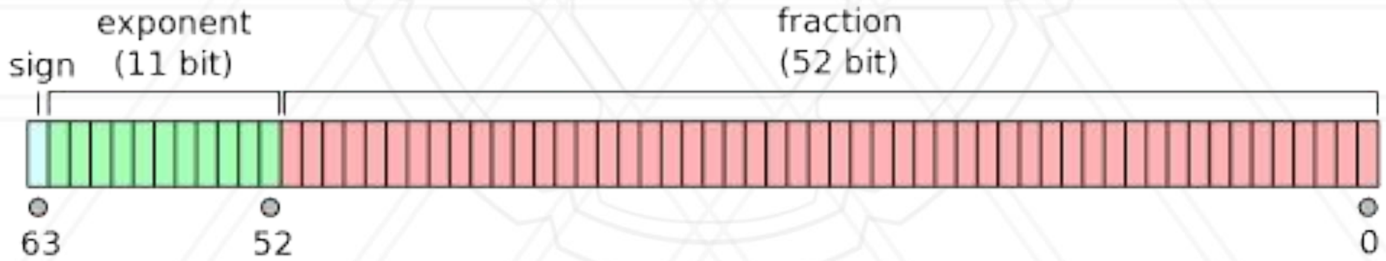
Announcements

- Interim report for the project is due on April 22
- Sign up for presentation slot (link on Piazza)
- Send data requirements (if needed) to course email by end of day today

Floating Point Numbers

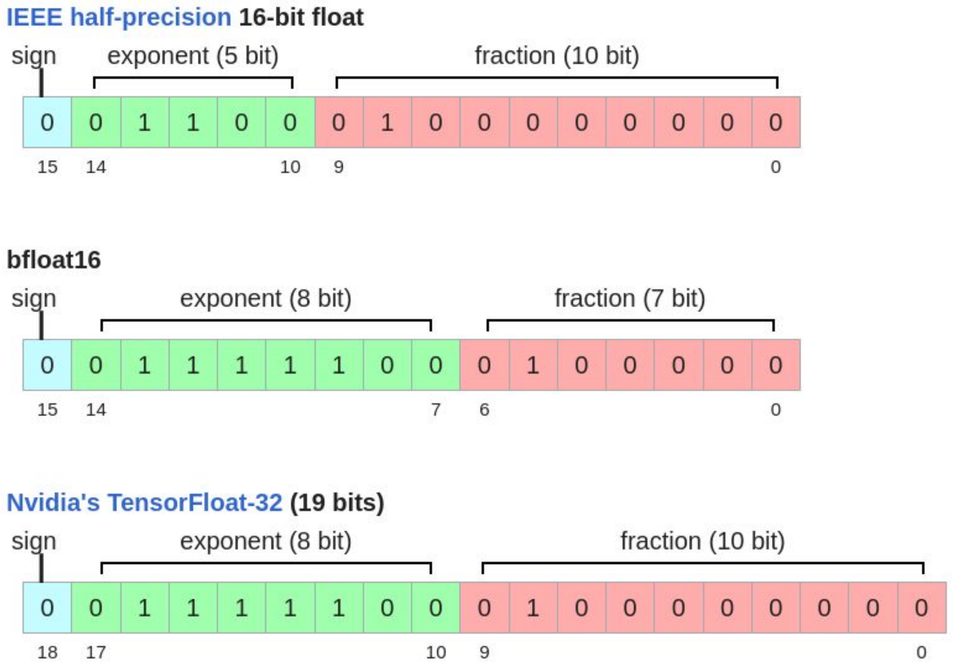
- Computers cannot store arbitrary precision
- Need approximations
- IEEE 754 floating point standard

$$(-1)^{\text{sign}} \left(\sum_{n=0}^{|\text{fraction}|-1} \frac{\text{bit}_n}{2^n} \right) \cdot 2^{e-\text{bias}}$$



Do we need all 64 bits?

- FP64 yields ~16 decimals of precision
  - https://en.wikipedia.org/wiki/IEEE_754#Basic_and_interchange_formats
- Deep learning does not need all this precision
  - We often use fp32, fp16, bf16, or some combination for training



Do we need all 64 bits?

- FP64 yields ~16 decimals of precision
  - https://en.wikipedia.org/wiki/IEEE_754#Basic_and_interchange_formats
- Deep learning does not need all this precision
  - We often use fp32, fp16, bf16, or some combination for training
- We usually need even less bits for inference!

Quantization

- Map existing weights/activations from higher to lower precision
  - i.e. fp32 -> fp16
- Why quantize?
  - Save memory
  - Potentially faster compute
  - Potentially lower energy consumption
- When not to quantize
  - Lack of native hardware support (if performance matters)
  - Operations sensitive to precision

Quantization: An Example

- Suppose we want to quantize a model with *N* parameters from fp32 to 8 bit integers
- fp32 can represent a much wider range of values
- How do we map them to int8?

An fp32 float can be written as an affine transformation of an 8 bit integer

$$x = \mu \cdot (\hat{x} - \eta)$$

$$\hat{x} = \left\lfloor \frac{x}{\mu} + \eta \right\rceil$$

Any issues?

We need to clip the quantized values to some range i.e. 0-255, -127-127, …

Quantization: An Example

- Suppose we want to quantize a model with *N* parameters from fp32 to 8 bit integers
- fp32 can represent a much wider range of values
- How do we map them to int8?

An fp32 float can be written as an affine transformation of an 8 bit integer

$$x = \mu \cdot (\hat{x} - \eta)$$

$$\hat{x} = \left\lfloor \frac{x}{\mu} + \eta \right\rceil$$

Any other issues or improvements?

Parameters and range do not necessarily have to be fixed globally

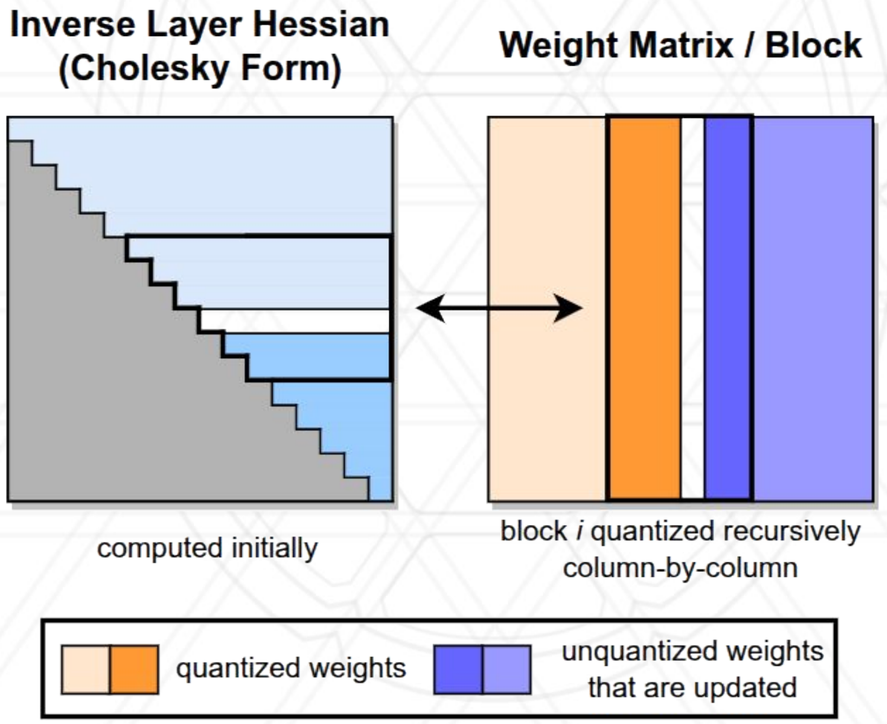
Types of Quantization

- Post-training quantization
  - Quantize the weights of a pre-trained model
  - Simple, less compute-intensive
- Quantization-aware Training
  - Adapt training process so that quantization is easier (i.e. ensure weights all in particular range)
  - Better quantizations at the cost of compute and training complexity

GPTQ

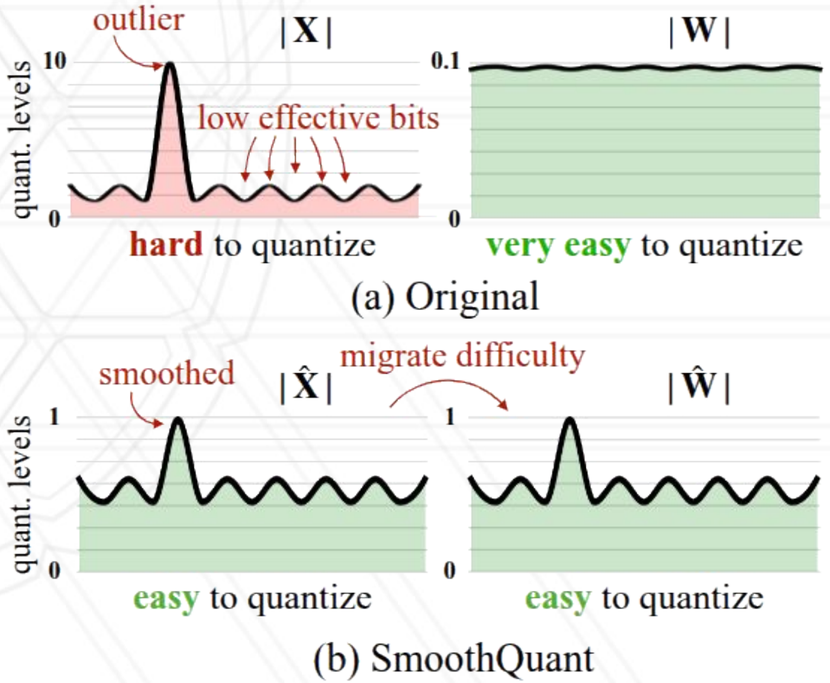
- Layer-wise quantization
- Efficient solution to optimization problem

$$\text{argmin}_{\widehat{\mathbf{W}}} ||\mathbf{W}\mathbf{X} - \widehat{\mathbf{W}}\mathbf{X}||_2^2.$$



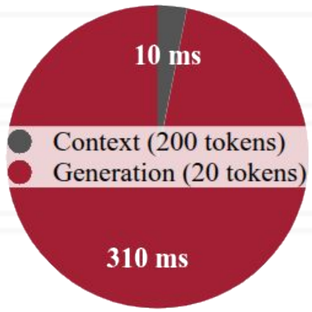
AWQ: Activation-Aware Weight Quantization

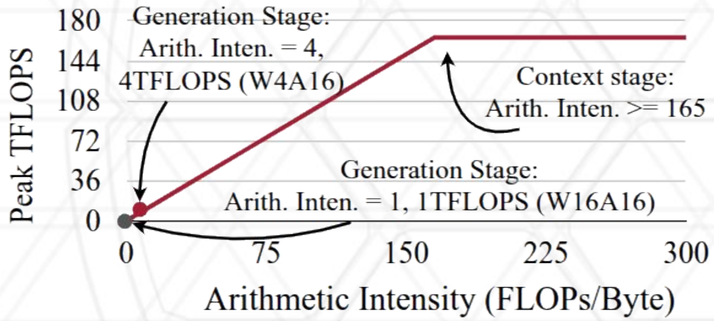
- Quantize weights of an LLM
- Not all weights are equally important
- Based on previous paper "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models"



Performance Motivation

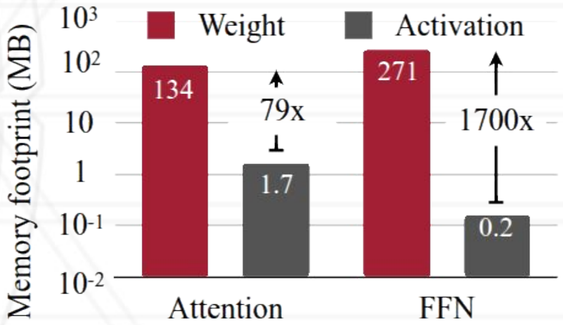
Most time in spent in generation





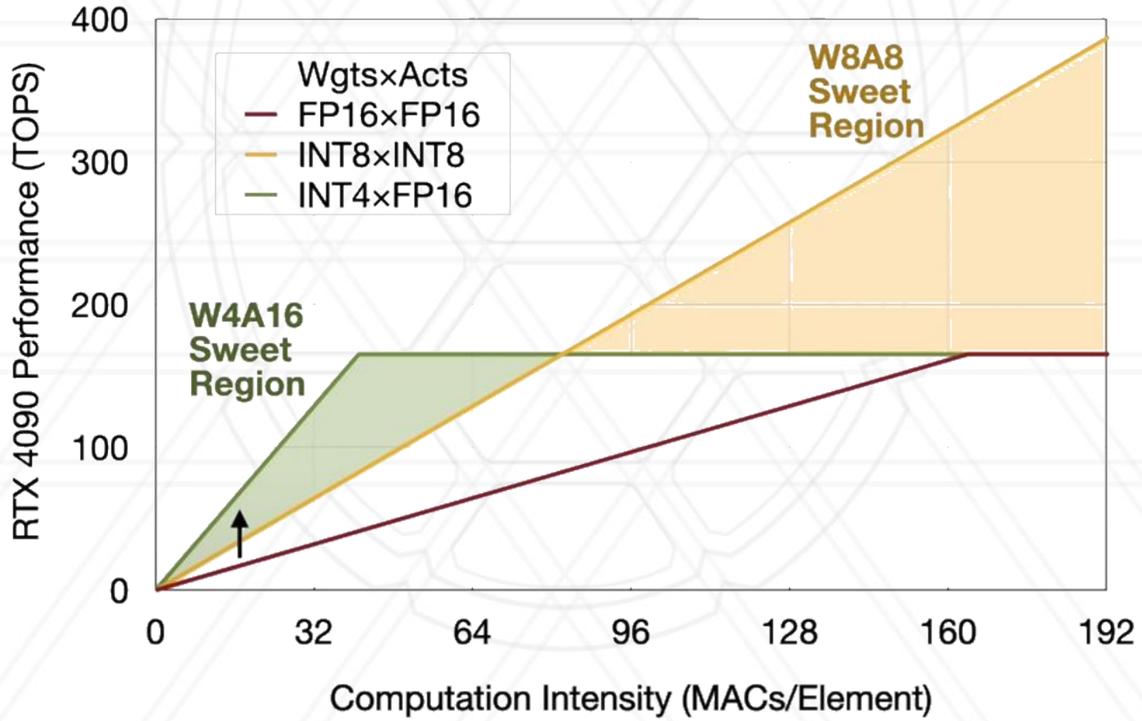
Generation is extremely memory bound

Quantization can significantly decrease the memory footprint



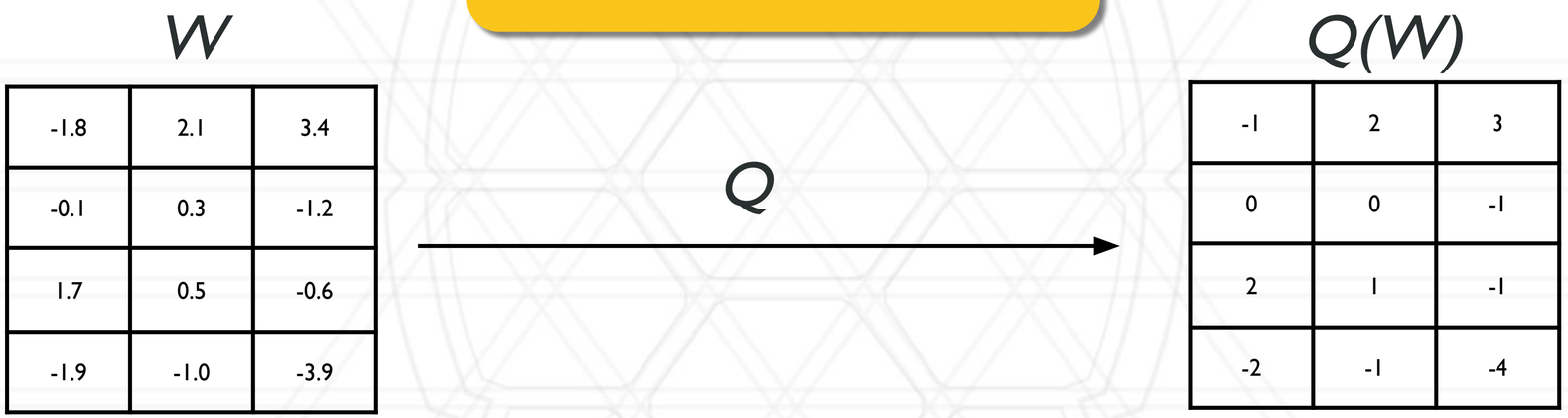
Why not quantize activations and weights?

- For lower arithmetic intensities weight-only quantization has a higher ceiling
- For larger scales quantizing both may be beneficial



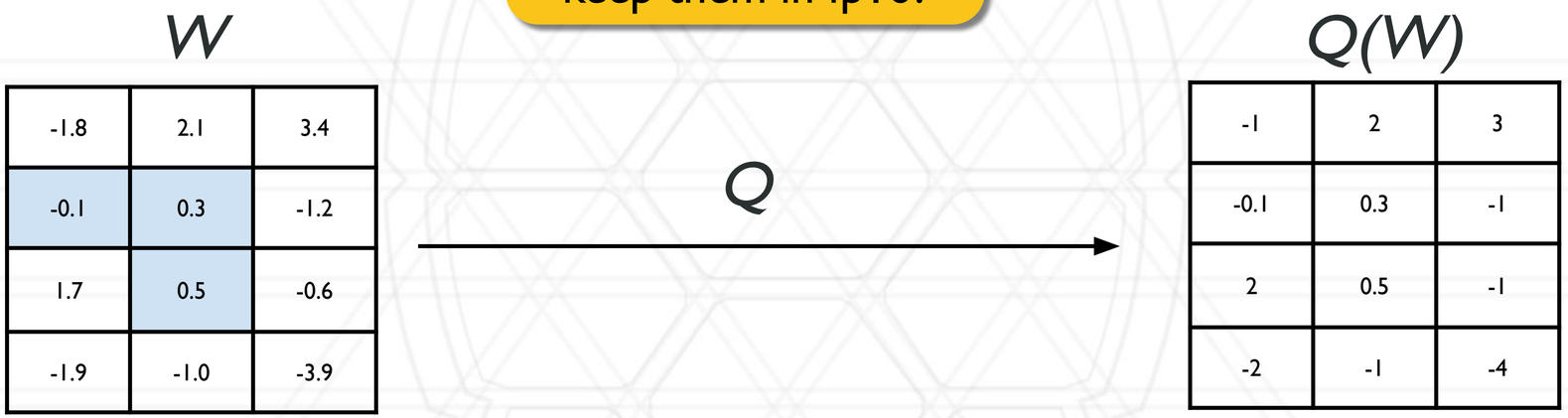
Naive Quantization Methods Hurt Model Performance

They find simple methods like RTN give subpar weights



Potential Weight-Aware Solution

Can we identify important weights and keep them in fp16?

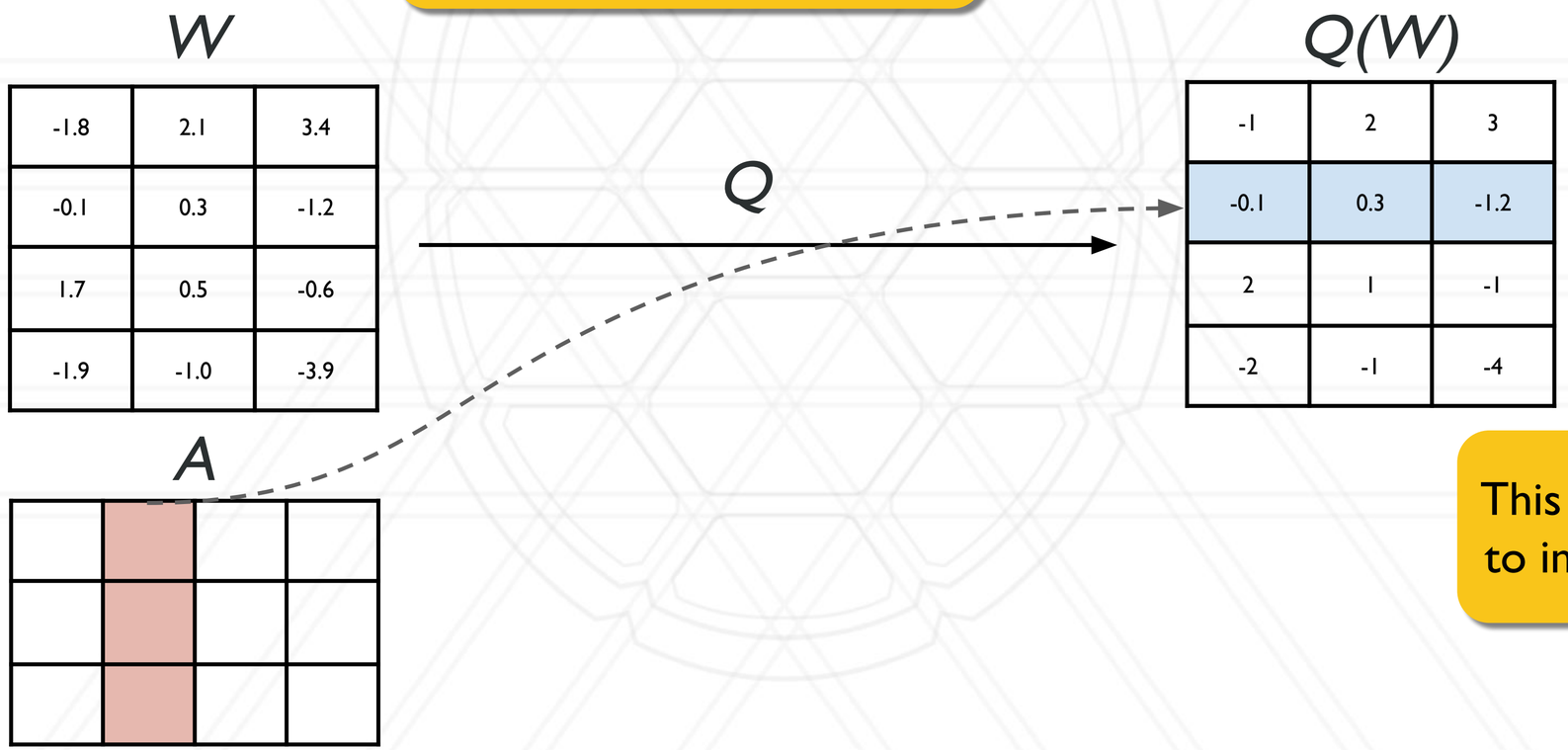


Use L1 or L2 norm to find most important weights

This also doesn't help much in practice

Use Activations to Determine Important Weights

Using corresponding activation magnitudes gives much better quantizations



This would be really difficult to implement and optimize!

Observation: Scaling Weights

- What happens when we scale particular weights before quantization?

$$y = \boldsymbol{w}\boldsymbol{x}, \quad y = Q(\boldsymbol{w})\boldsymbol{x}$$

Forward pass and forward pass with quantization

Linear quantization

$$Q(\boldsymbol{w}) = \Delta \cdot \text{Round}\left(\frac{\boldsymbol{w}}{\Delta}\right), \quad \Delta = \frac{\max\left(|\boldsymbol{w}|\right)}{2^{N-1}}$$

$$Q(w \cdot s) \cdot \left(\frac{x}{s}\right)$$

What happens when we scale by *s>1*?

Observation: Scaling Weights

- What happens when we scale particular weights before quantization?

$$y = \boldsymbol{w}\boldsymbol{x}, \quad y = Q(\boldsymbol{w})\boldsymbol{x}$$

$$Q(\boldsymbol{w}) = \Delta \cdot \text{Round}\left(\frac{\boldsymbol{w}}{\Delta}\right)$$

$$\Delta = \frac{\max\left(|\boldsymbol{w}|\right)}{2^{N-1}}$$

Observation 1: Expected error of rounding is ~0.25; independent of w and s

$$Q(w \cdot s) \cdot \left(\frac{x}{s}\right) = \Delta' \cdot \text{Round}\left(\frac{ws}{\Delta'}\right) \cdot \frac{x}{s}$$

Observation 2: Scaling a single or small subset of w values has little impact on Δ. $\Delta' \approx \Delta$

Observation 3: x and Δ are fp16 and have no quantization error

Observation: Scaling Weights

- What happens when we scale particular weights before quantization?

$$y = \boldsymbol{w}\boldsymbol{x}, \quad y = Q(\boldsymbol{w})\boldsymbol{x}$$

$$Q(\boldsymbol{w}) = \Delta \cdot \text{Round}\left(\frac{\boldsymbol{w}}{\Delta}\right)$$

$$\Delta = \frac{\max\left(|\boldsymbol{w}|\right)}{2^{N-1}}$$

$$\texttt{Err}(Q(w)x) = \Delta \cdot \texttt{RoundErr}\left(\frac{w}{\Delta}\right) \cdot x$$

$$\texttt{Err}\left(Q(w \cdot s)\left(\frac{x}{s}\right)\right) = \Delta' \cdot \texttt{RoundErr}\left(\frac{ws}{\Delta'}\right) \cdot \frac{x}{s}$$

Based on observation 3 we can represent the quantization errors

Scaled term has lower relative error for *s>1*!

$$\frac{\texttt{Err}\left(Q(w \cdot s)\left(\frac{x}{s}\right)\right)}{\texttt{Err}\left(Q(w)x\right)} = \frac{\Delta'}{\Delta \cdot s} \approx \frac{1}{s}$$

But Does It Work?

- Try different *s* in practice

As *s* gets larger the $\Delta' \approx \Delta$ assumption breaks down

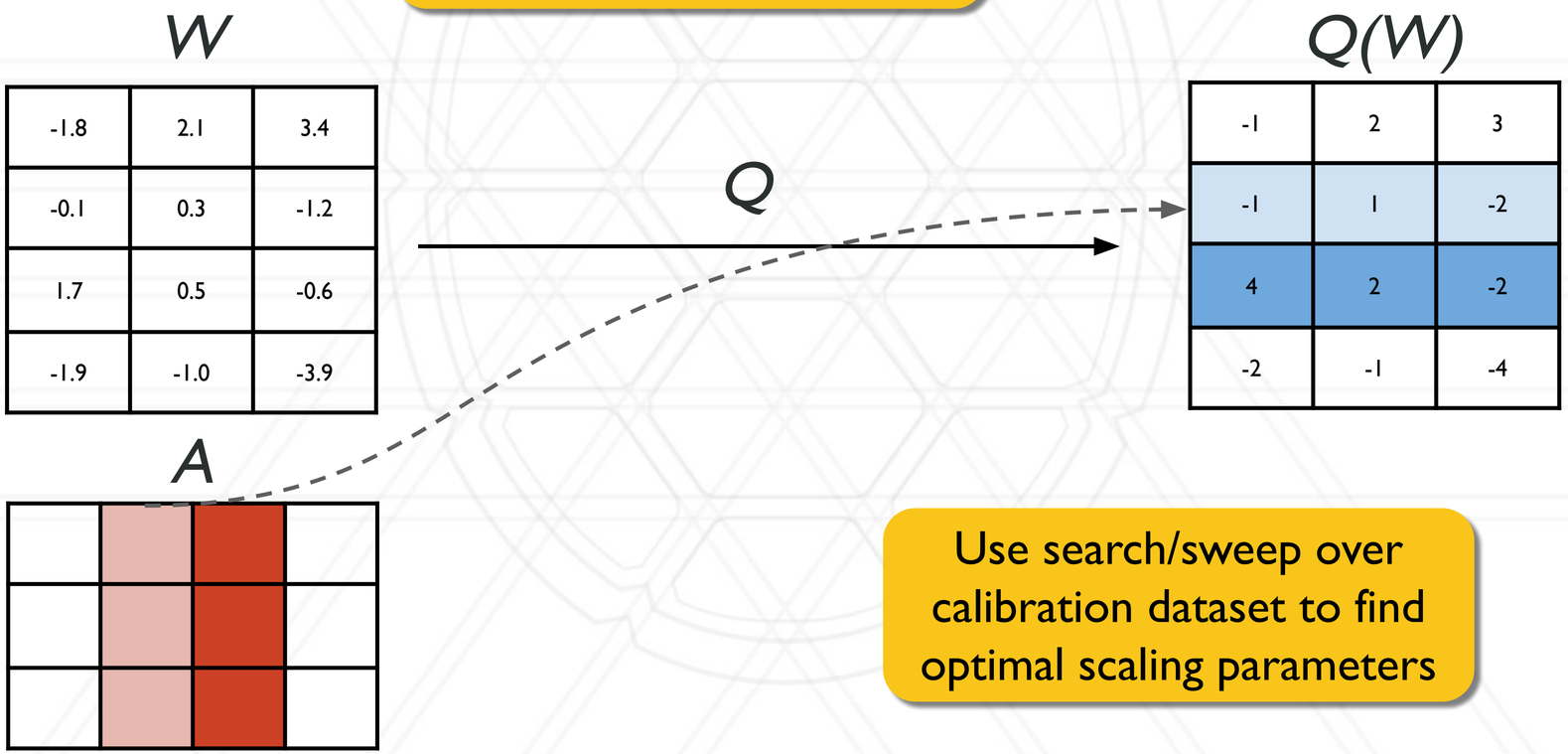
OPT-6.7B	$s = 1$	$s = 1.25$	$s = 1.5$	$s = 2$	$s = 4$
proportion of $\Delta' \neq \Delta$	0%	2.8%	4.4%	8.2%	21.2%
average Δ'/Δ	1	1.005	1.013	1.038	1.213
average $\frac{\Delta'}{\Delta} \cdot \frac{1}{s}$	1	0.804	0.676	0.519	**0.303**
Wiki-2 PPL	23.54	12.87	12.48	**11.92**	12.36

The relative error does keep getting smaller

In practice, some s^* gives the best modeling performance

Use Activations to Determine Important Weights

Scale weights based on activation magnitude before quantization



Use search/sweep over calibration dataset to find optimal scaling parameters

AWQ Results

AWQ is performs better or near 1% fp16, but is much faster and more memory efficient

OPT (PPL↓)	1.3B	2.7B	6.7B	13B	30B
FP16	14.62	12.47	10.86	10.13	9.56
RTN	119.47	298.00	23.54	46.04	18.80
1% FP16	16.91	13.69	**11.39**	**10.43**	9.85
$s = 2$	18.63	14.94	11.92	10.80	10.32
AWQ	**16.32**	**13.58**	**11.39**	10.56	**9.77**

A fixed value of *s* gives decent results, but it's better to select dynamically

AWQ Comparison with Other Methods

AWQ yields better perplexity than other SotA approaches (GPTQ*)

PPL↓		Llama-2			LLaMA			
		7B	13B	70B	7B	13B	30B	65B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	3.53
INT3 g128	RTN	6.66	5.52	3.98	7.01	5.88	4.88	4.24
	GPTQ	6.43	5.48	3.88	8.81	5.66	4.88	4.17
	GPTQ-R	6.42	5.41	3.86	6.53	5.64	4.74	4.21
	AWQ	**6.24**	**5.32**	**3.74**	**6.35**	**5.52**	**4.61**	**3.95**
INT4 g128	RTN	5.73	4.98	3.46	5.96	5.25	4.23	3.67
	GPTQ	5.69	4.98	3.42	6.22	5.23	4.24	3.66
	GPTQ-R	5.63	4.99	3.43	5.83	5.20	4.22	3.66
	AWQ	**5.60**	**4.97**	**3.41**	**5.78**	**5.19**	**4.21**	**3.62**

Still worse than fp16

AWQ Downstream Tasks

- Math and coding downstream tasks

AWQ performance is close to fp16 performance

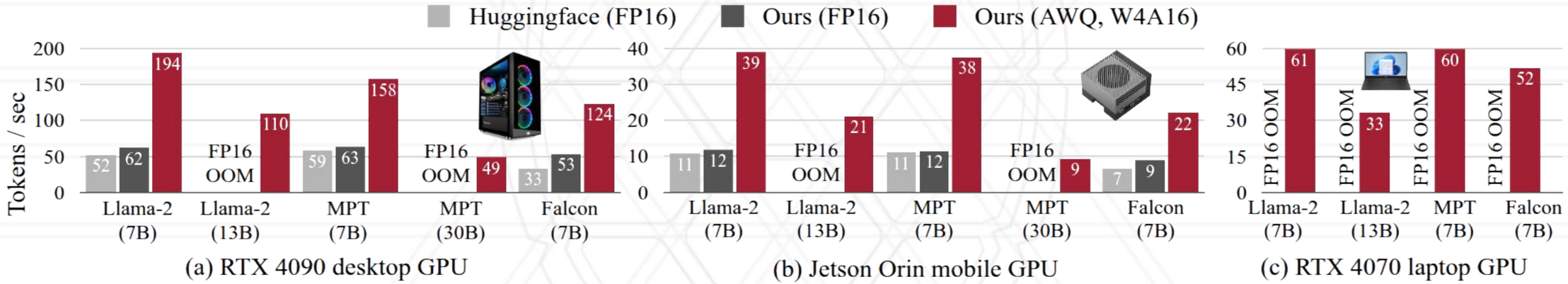
MBPP (7B)	pass@1	pass@10
FP16	38.53	49.77
RTN	37.51	48.49
GPTQ	31.97	44.75
AWQ	**40.64**	**49.25**

GSM8K	7B	13B	70B
FP16	13.87	26.16	56.41
RTN	11.07	21.23	53.98
GPTQ	12.13	24.26	56.03
AWQ	**13.57**	**25.25**	**56.40**

AWQ Runtime Results

- TinyChat benchmark for edge devices

AWQ is faster and saves on memory



Binarization

- Special form of quantization
- Map weights to {-1,1} or {-1,0,1}
- 1.58-bit models, "The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits"
- Train using int8 activations and BitLinear layers

What is large potential optimization we can make assuming we have ternary weights?

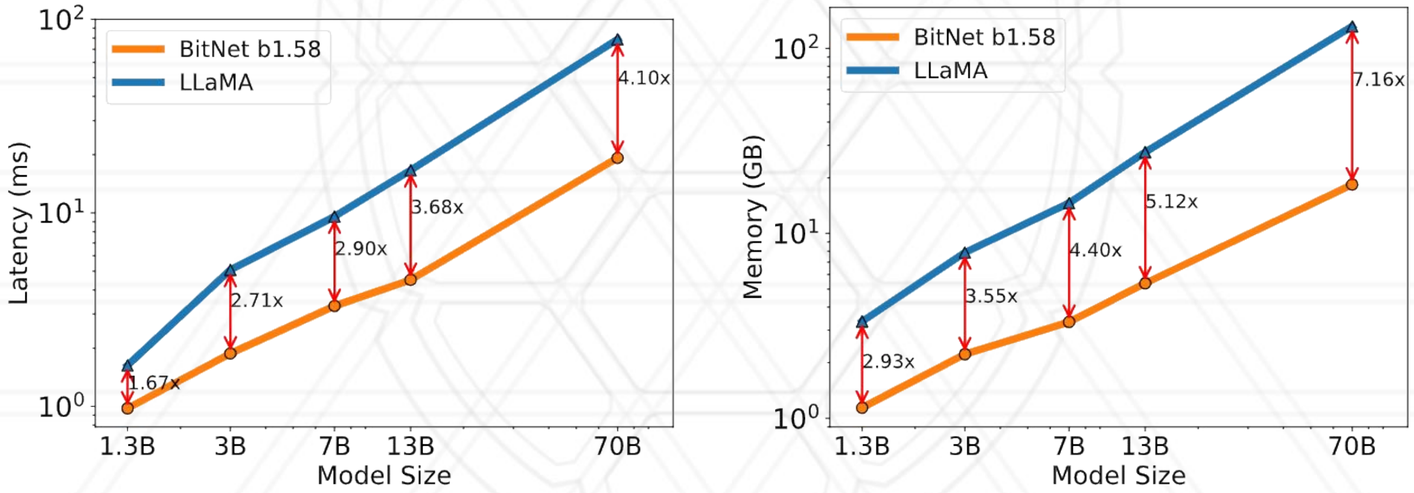
$$\widetilde{W} = \mathrm{RoundClip}(\frac{W}{\gamma + \epsilon}, -1, 1),$$

$$\mathrm{RoundClip}(x, a, b) = \max(a, \min(b, \mathrm{round}(x))),$$

$$\gamma = \frac{1}{nm} \sum_{ij} |W_{ij}|.$$

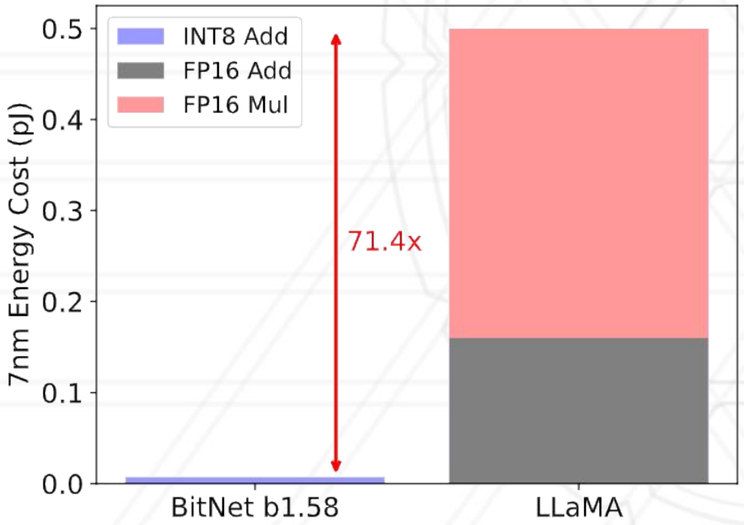
Potential Savings with Binarized Models

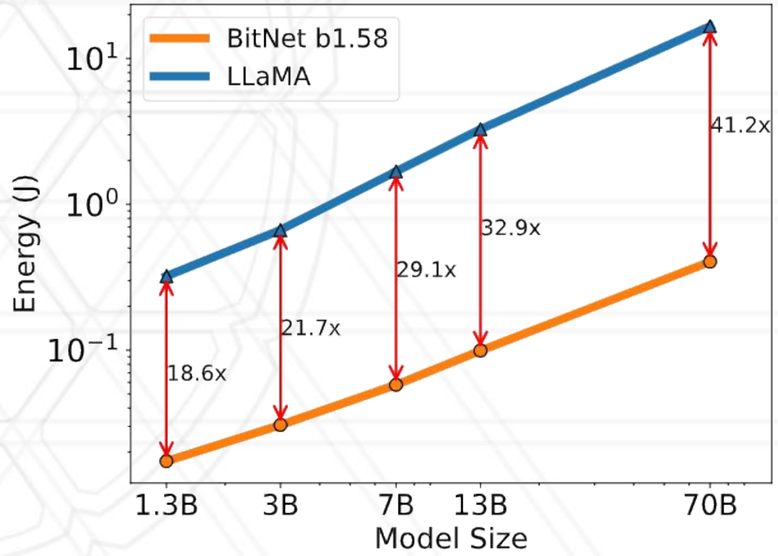
Binarized models are faster and take up less memory



Potential Savings with Binarized Models

Using exclusively int8 additions is extremely energy efficient





UNIVERSITY OF
MARYLAND