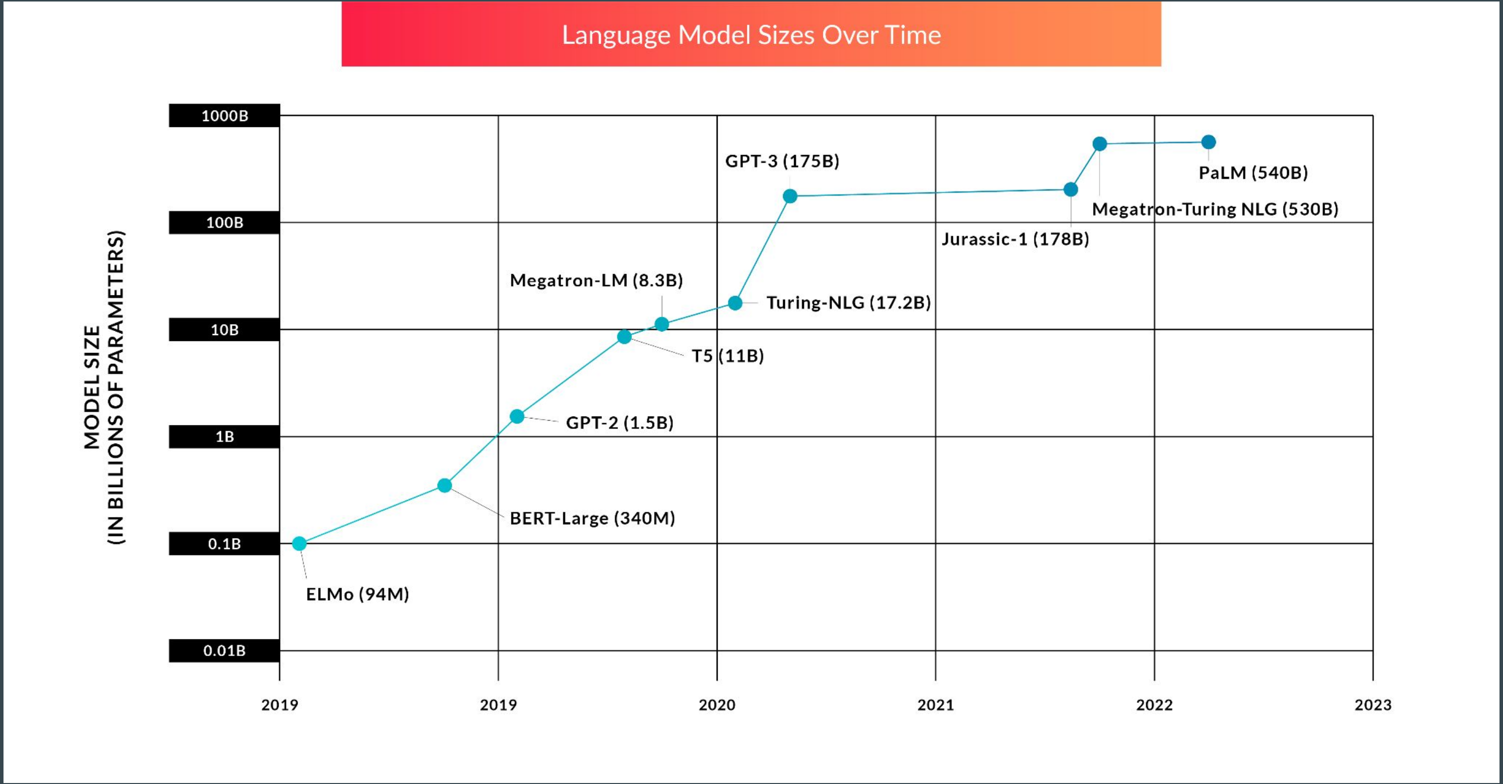
AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration

Sabrina Li, Krithika Ramesh

Language models are getting much bigger

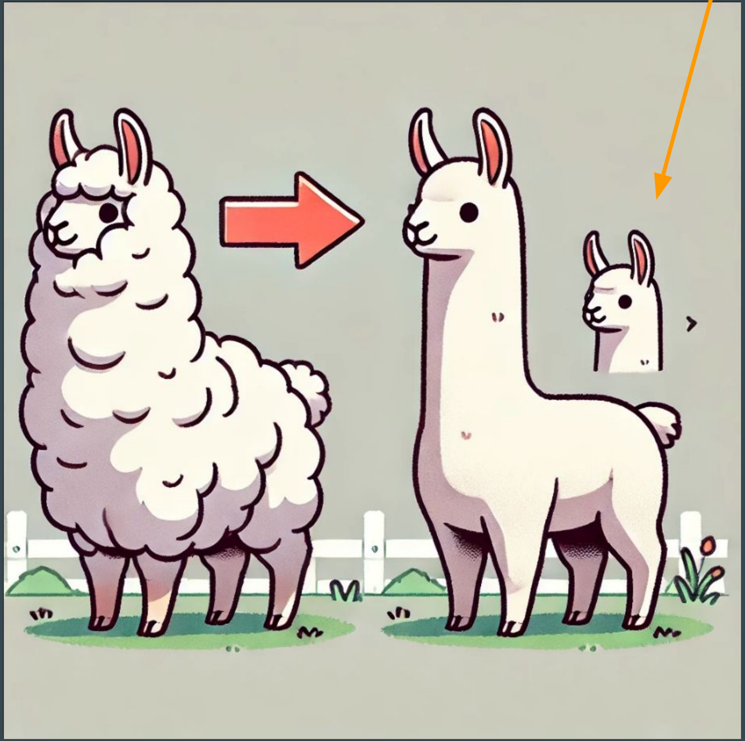


How can we address the resource-intensive challenges that come with scaling up to larger models (and make every byte count)?

Model Pruning

- Removes components of the model while (hopefully) preserving performance
- Many different types (static/dynamic, structured/unstructured, etc.)

Ideally, we don't introduce weird behaviors/artifacts while pruning...

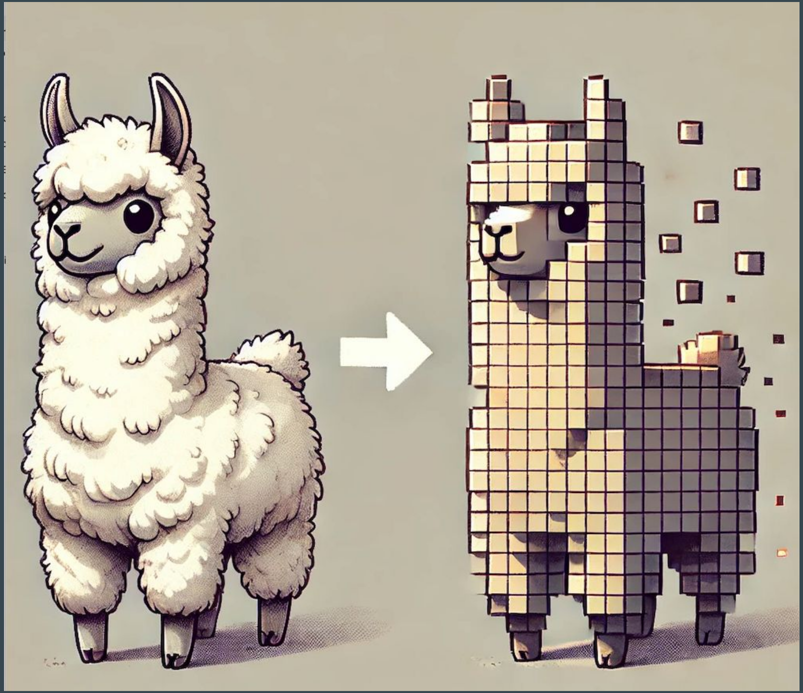


Model Distillation

- Knowledge transfer typically done from a larger (teacher) model to a smaller (student) model by minimizing differences in different components of the model
- Example: Logits, hidden states, embedding outputs, layer-wise distillation, etc.

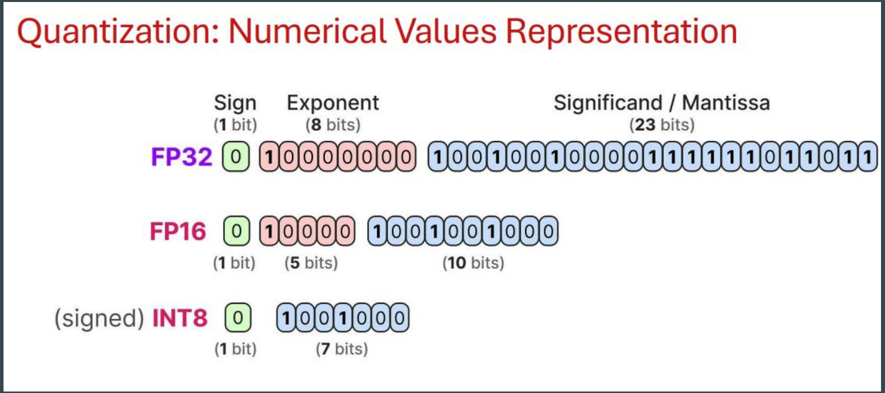
Model Quantization

- Doesn’t alter the model architecture, but reduces the memory footprint by simplifying representations into a lower bit precision format
- **This is what we’re going to focus on today**



What is quantization?

- *Quantization* is when you represent weights and/or activations with lower-precision data types (for example, converting fp32 to int8)
- This reduces memory and computational costs
- But, it inevitably leads to some amount of **quantization error**



Types of quantization

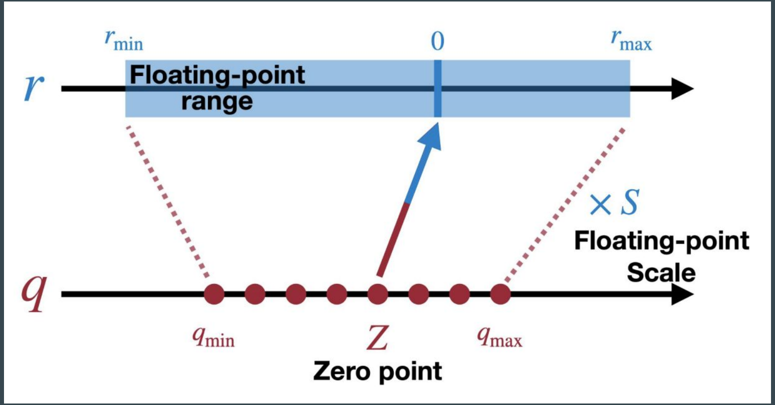
Quantization aware training (QAT)

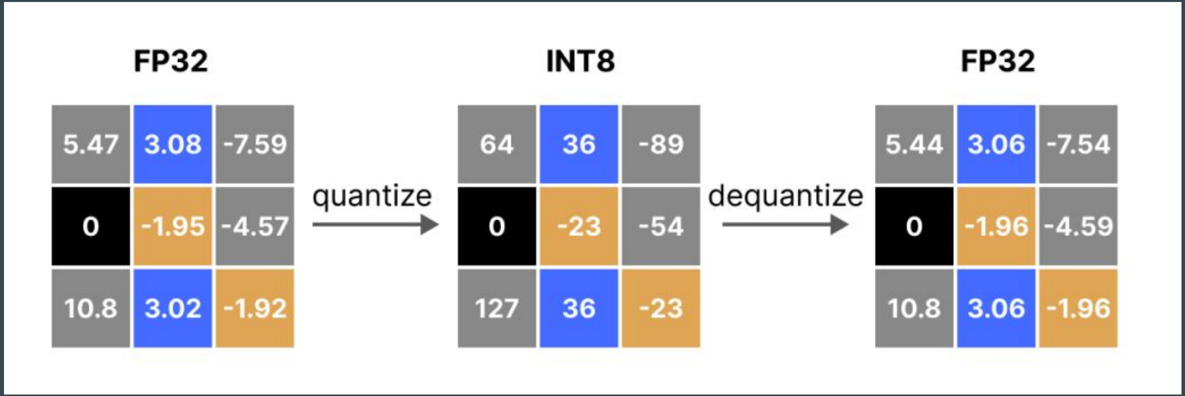
- Forward pass makes use of low-precision weights, but the backpropagation involves gradient estimation methods that maintain the original precision - this reduces the noise from quantization
- Scales badly for LLMs
  - Low precision often leads to training instability, resulting in higher overall training cost

Post-training quantization

- This method doesn't involve retraining the fine-tuned/pre-trained model; instead, we use a small calibration set to minimize the error from quantization by determining the scale and the zero point, which determine the range of values that the of the layers will map to when converted to lower-bit precisions.

Mapping from fp to ints and back





FP32

5.47	3.08	-7.59
0	-1.95	-4.57
10.8	3.02	-1.92

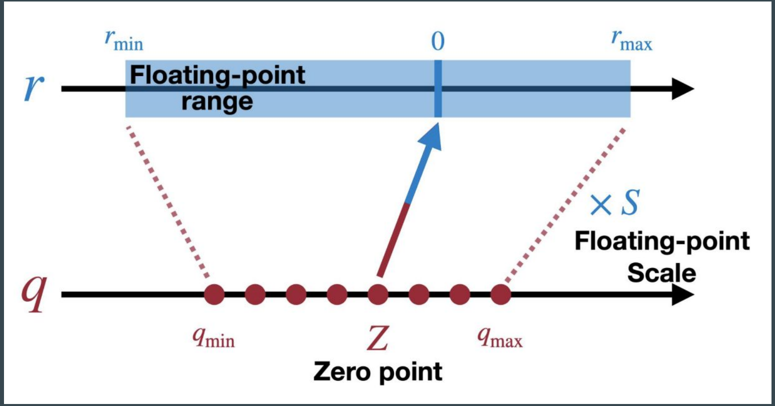
INT8

64	36	-89
0	-23	-54
127	36	-23

FP32

5.44	3.06	-7.54
0	-1.96	-4.59
10.8	3.06	-1.96

Linearly quantizing a layer



Original layer:

$$y = \mathbf{w}\mathbf{x},$$

Layer after quantization:

$$y = Q(\mathbf{w})\mathbf{x}$$

Quantization function (using absolute max quantization):

$$Q(\mathbf{w}) = \Delta \cdot \boxed{\text{Round}(\frac{\mathbf{w}}{\Delta}),}$$

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

Note: there is a typo in the paper

Quantized weights

N is the quantization bits, Δ is the quantization scaler

Just to be clear about notation:

These are the quantized weights (which are stored):

$$Q_{int}(\boldsymbol{w}) = \text{Round}(\frac{\boldsymbol{w}}{\Delta})$$

Q(w) is the **quantization function** :

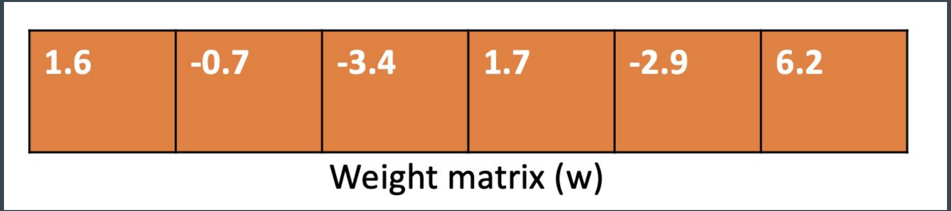
$$Q(\mathbf{w}) = \Delta \cdot \text{Round}(\frac{\mathbf{w}}{\Delta}),$$

The overall layer (after quantization) looks like this:

$$y = Q(\boldsymbol{w}) \cdot \mathbf{x} \quad \Rightarrow \quad y = \Delta \cdot \text{Round}\left(\frac{\boldsymbol{w}}{\Delta}\right) \cdot \mathbf{x}$$

Absolute Max Quantization Example

Given a weight matrix (w) that we want to quantize to INT4



1.6	-0.7	-3.4	1.7	-2.9	6.2

Weight matrix (w)

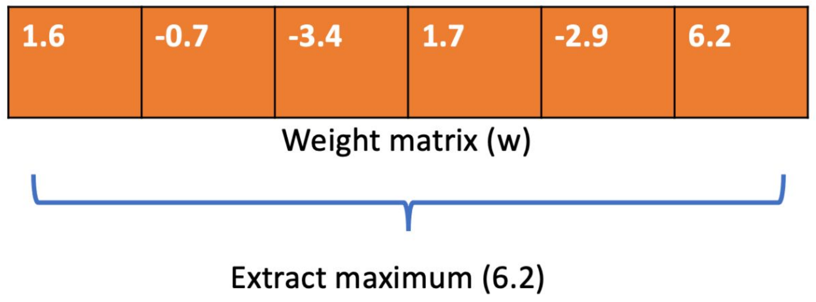
INT4 has a range of [-8, 7] (two's complement system).

In practice, we clip the negative range to make it symmetrical → [-7, 7]

Absolute Max Quantization Example - calculating the scaler Δ

Start by calculating quantization scaler (Δ):

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



$N = 4$ (since 4 bit precision)

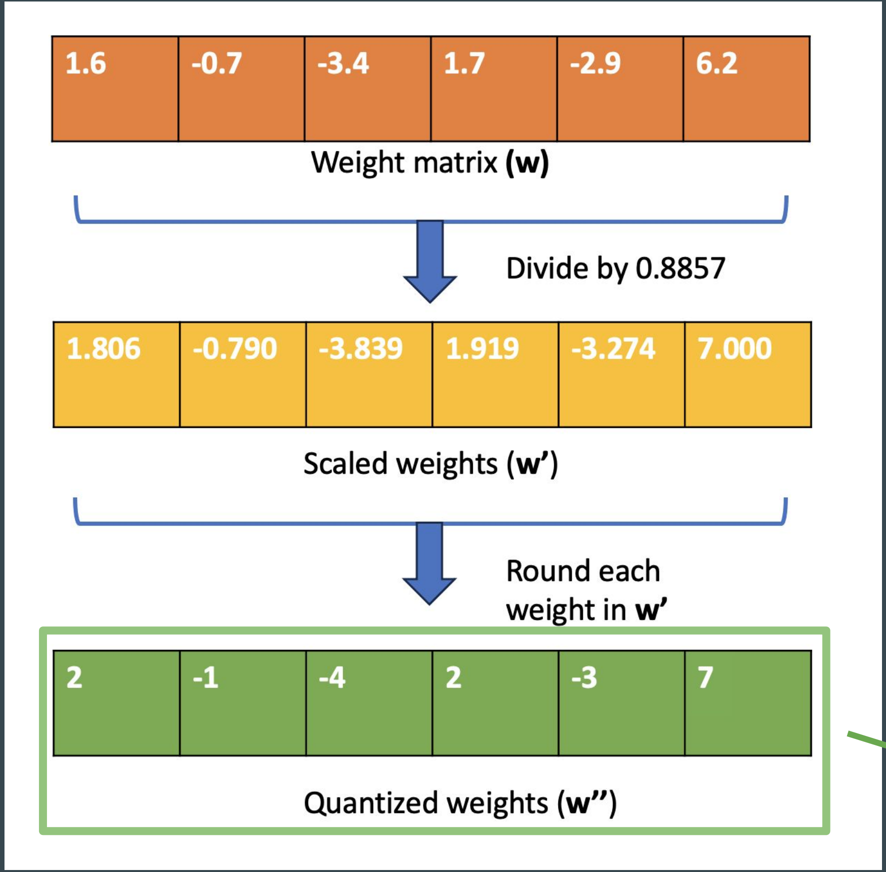
$$\Delta = \frac{\max(|\boldsymbol{w}|)}{2^{N-1}-1}$$

$$\Delta = \frac{6.2}{2^{4-1}-1}$$

$$\Delta = \frac{6.2}{7}$$

$$= 0.8857$$

Deriving the quantized weights:



$$Q(\mathbf{w}) = \Delta \cdot \boxed{\text{Round}(\frac{\boldsymbol{w}}{\Delta})}$$

$$\Delta = 0.8857$$

Notice that all weights in **w'** and **w''** are within the range of [-7, 7] as we would want for INT4

Yay, our weights are now quantized!

What about at inference time?

Looking at the quantization function again

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

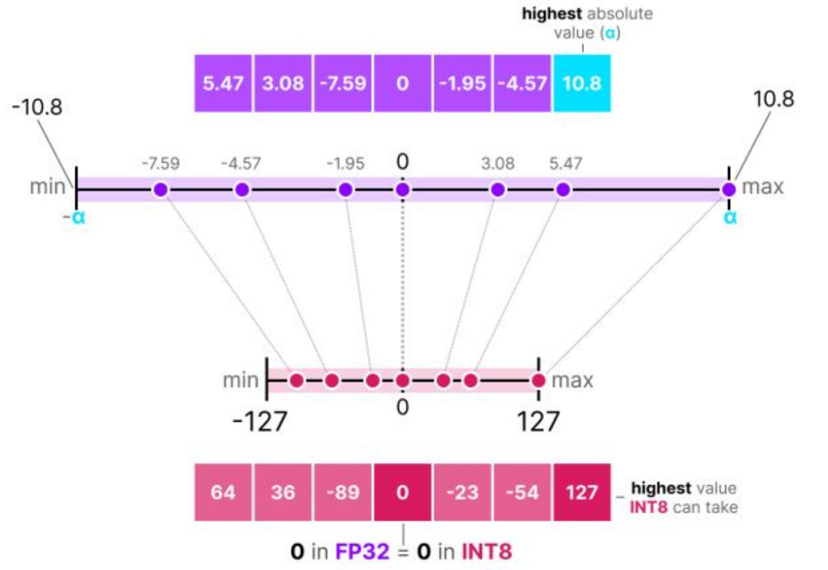
Reconstruction process: scale back up quantized weights before multiplying with input **x**

Quantized weights (these are saved/stored)

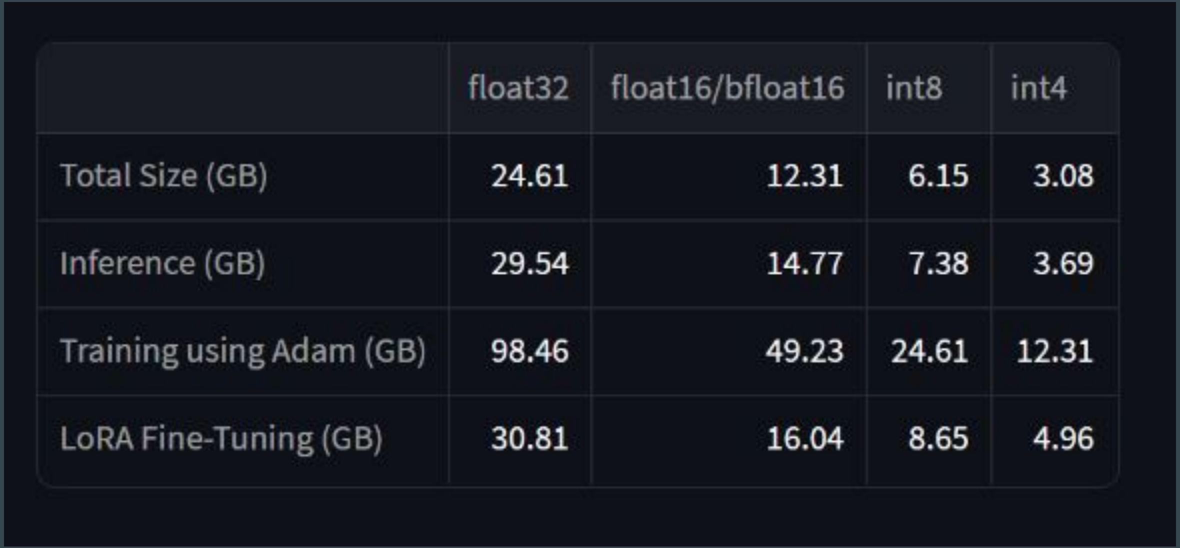
$$y = Q(\mathbf{w})\mathbf{x}$$

Benefit: we only store the quantized weights (low bit ints) and the scaling factor Δ (float). This is better than storing an entire weight matrix of floats

Quantizing FP32 to INT8



highest absolute value (α)
5.47
3.08
-7.59
0
-1.95
-4.57
10.8
-10.8
10.8
-7.59
-4.57
-1.95
0
3.08
5.47
min
max
-α
α
min
max
-127
0
127
64
36
-89
0
-23
-54
127
highest value **INT8** can take
0 in **FP32** = **0** in **INT8**

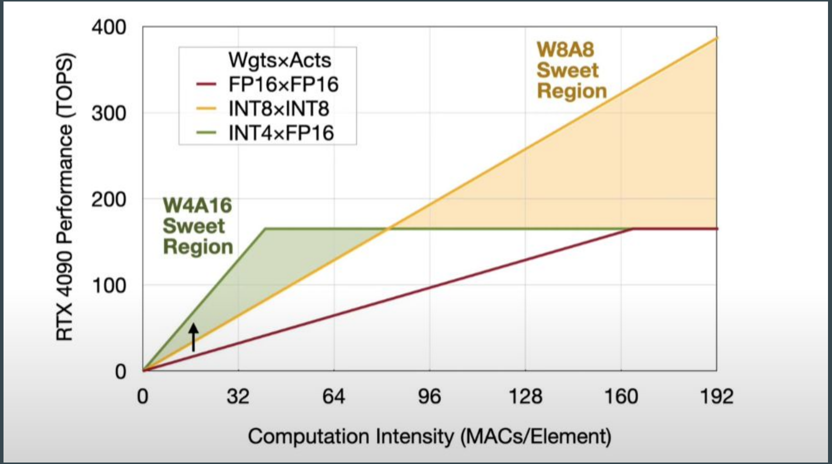


	float32	float16/bfloat16	int8	int4
Total Size (GB)	24.61	12.31	6.15	3.08
Inference (GB)	29.54	14.77	7.38	3.69
Training using Adam (GB)	98.46	49.23	24.61	12.31
LoRA Fine-Tuning (GB)	30.81	16.04	8.65	4.96

Memory consumption in GB for codellama/CodeLlama-7b-hf

Why do we only quantize the weights and not the activations?

1. W8A8 quantization (both activations and weights are quantized to INT8)
2. **Low-bit weight-only quantization** (ex: W4A16) - only weights are quantized into low-bit ints
    a. Pros: requires smaller memory size, speeds up token generation
    b. Cons: current methods like GPTQ overfit the dev set and may not preserve generalist abilities of LLMs

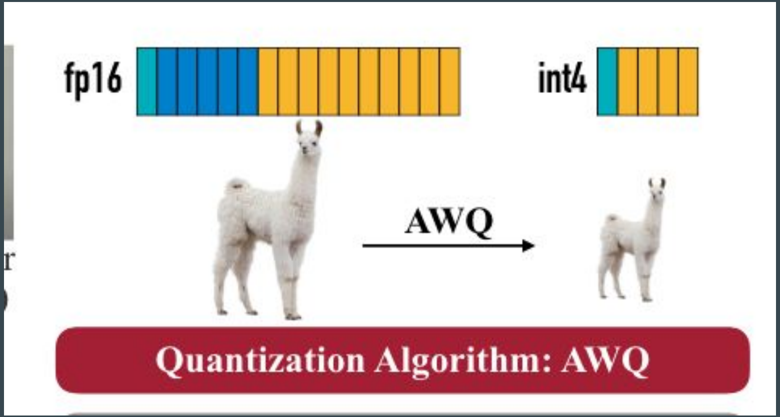


All weights are important, but some weights are more important than others.

What does AWQ try to do and how is it different from other methods?

- The intuition: **some weights are more important than others. Don't quantize the most important (salient) weights.**
- This should reduce overall model degradation due to quantization error

- AWQ is a quantization algorithm that is sensitive to the **saliency of weights**
- Weight magnitude is not used to determine weight importance
- Instead, **using the magnitude of the activation** demonstrates significantly better results



How do we protect salient weights during quantization?

•••

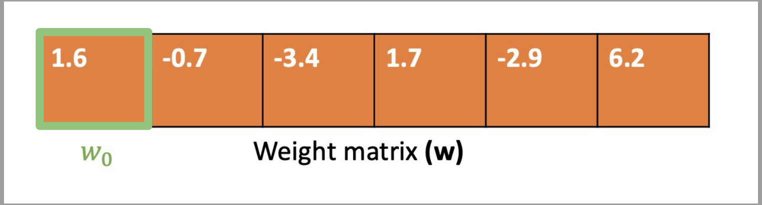
Skipping the quantization of weights that significantly influence LLM performance should reduce the quantization loss.

Scale them!

- Intuition: when weight values are small, they are more susceptible to quantization error due to rounding
- By scaling up an element in the weight matrix **w** by a factor of **s** before quantizing it, we protect this element during the quantization process

Notation Warning!

- w (not bolded) represents one weight in the weight matrix **w (bolded)**



Scale them! - continued

Instead of:

$$Q_{int}(\boldsymbol{w}) = \mathrm{Round}(\frac{\boldsymbol{w}}{\Delta})$$

Quantize the entire weight matrix directly.

We have:

$$Q_{int}(w_i \cdot s_i) = \mathrm{Round}(\frac{w_i \cdot s_i}{\Delta})$$

For each element w_i in the weight matrix **w**, multiply it by s_i and then quantize

What does the quantization function look like now?

$$Q(\boldsymbol{w} \circ \boldsymbol{s}) = \Delta' \cdot \mathrm{Round}(\frac{\boldsymbol{w} \circ \boldsymbol{s}}{\Delta'})$$

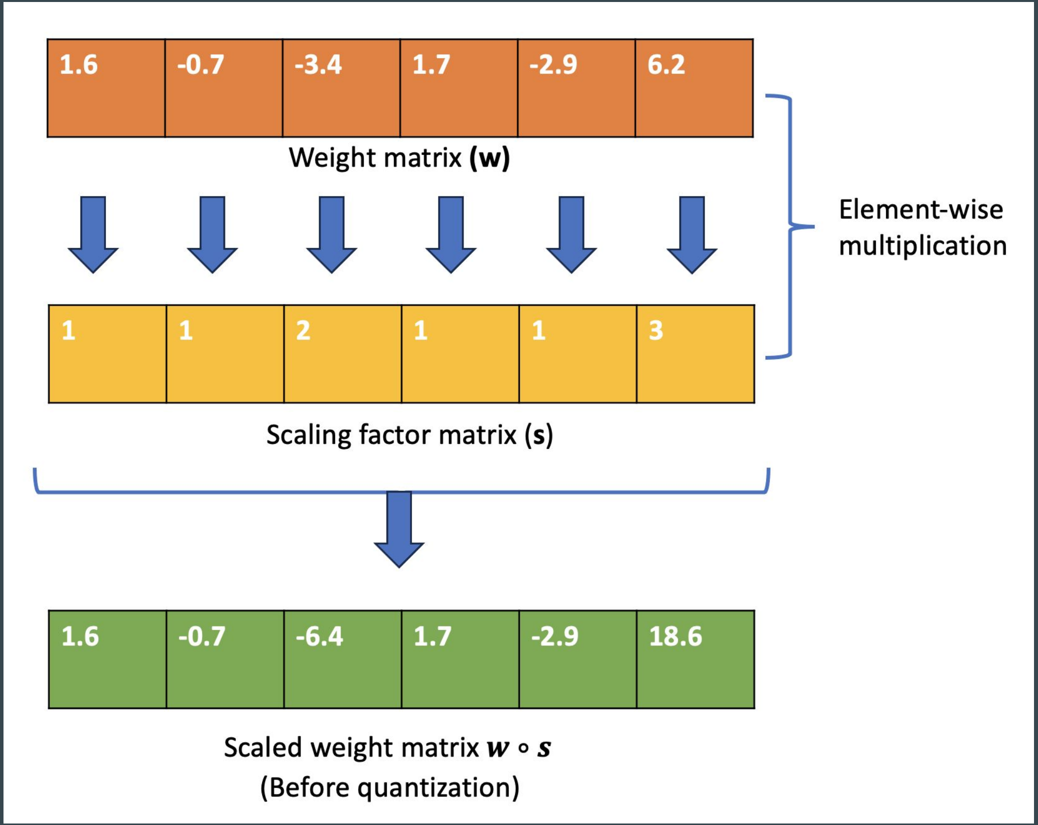
where the quantization scaling factor is

$$\Delta' = \frac{\max(|\boldsymbol{w} \circ \boldsymbol{s}|)}{2^{N-1}}$$

Before:

$$Q(\mathbf{w}) = \Delta \cdot \mathrm{Round}(\frac{\mathbf{w}}{\Delta}), \quad \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}$$

Scaling weights before quantization (intuition)



1.6
-0.7
-3.4
1.7
-2.9
6.2
Weight matrix **(w)**
Element-wise multiplication
1
1
2
1
1
3
Scaling factor matrix (**s**)
1.6
-0.7
-6.4
1.7
-2.9
18.6
Scaled weight matrix $\boldsymbol{w} \circ \boldsymbol{s}$
(Before quantization)

$$Q(\boldsymbol{w} \circ \boldsymbol{s}) = \Delta' \cdot \mathrm{Round}\left(\frac{\mathbf{w} \circ \boldsymbol{s}}{\Delta'}\right)$$

$$\Delta' = \frac{\max(|\boldsymbol{w} \circ \boldsymbol{s}|)}{2^{N-1}}$$

What the quantized layer looks like (when applying weight scaling):

$$y = Q(\boldsymbol{w} \circ \boldsymbol{s}) \cdot \boldsymbol{x} \circ \boxed{\frac{1}{\boldsymbol{s}}} \rightarrow y = \Delta' \cdot \mathrm{Round}\left(\frac{\mathbf{w} \circ \boldsymbol{s}}{\Delta'}\right) \cdot \boldsymbol{x} \circ \frac{1}{\boldsymbol{s}}$$

Since we scaled up the weight matrix **w** by a factor of **s**, we will want to multiply by 1/**s** before getting our result

Empirical observations about scaling and quantization error:

1. Rounding error is not affected by scaling (error caused by converting floating-point to integer is roughly uniformly distributed [0, 0.5] with average error of 0.25)
2. Scaling up an element w_i does not usually change the maximum value in **w** so $\Delta' \approx \Delta$

$$\Delta' = \frac{\max(|\boldsymbol{w} \circ \boldsymbol{s}|)}{2^{N-1}}$$

for reference

3. Δ and x are represented in FP16 and have no quantization error

Quantization error:

Quantization error for non-scaled weights (s = 1):

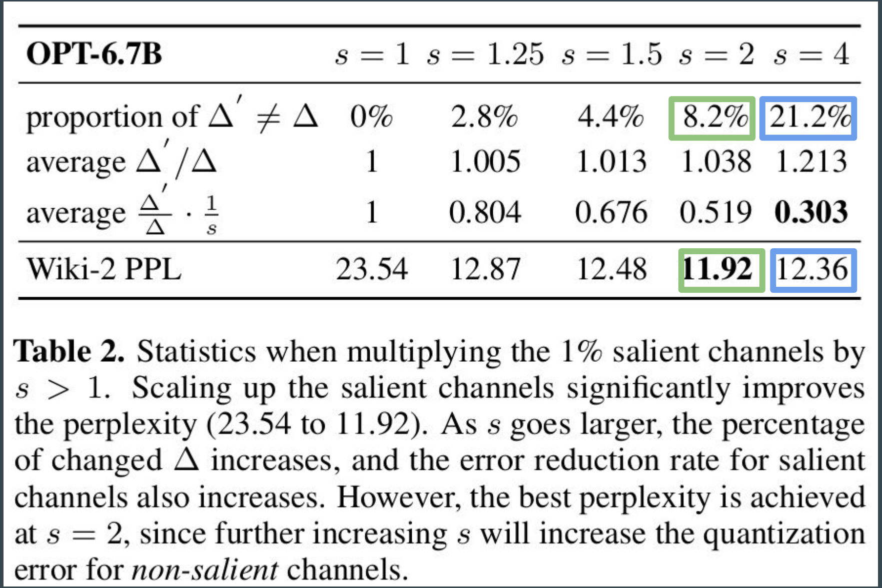
$$\texttt{Err}(Q(w)x) = \Delta \cdot \texttt{RoundErr}(\frac{w}{\Delta}) \cdot x$$

Quantization error for scaled weights (s >= 1):

$$\texttt{Err}(Q(w \cdot s)(\frac{x}{s})) = \Delta^{'} \cdot \texttt{RoundErr}(\frac{ws}{\Delta^{'}}) \cdot x \cdot \frac{1}{s}$$

Given the observations from the last slide, the quantization error is smaller for weights with s > 1.

Effect of generally increasing scale-up for weights

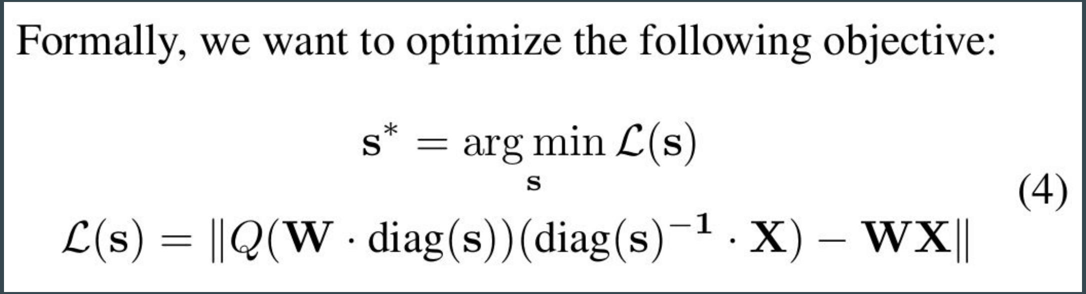


OPT-6.7B	$s = 1$	$s = 1.25$	$s = 1.5$	$s = 2$	$s = 4$
proportion of $\Delta^{'} \neq \Delta$	0%	2.8%	4.4%	8.2%	21.2%
average $\Delta^{'}/\Delta$	1	1.005	1.013	1.038	1.213
average $\frac{\Delta^{'}}{\Delta} \cdot \frac{1}{s}$	1	0.804	0.676	0.519	**0.303**
Wiki-2 PPL	23.54	12.87	12.48	**11.92**	12.36

Table 2. Statistics when multiplying the 1% salient channels by $s > 1$. Scaling up the salient channels significantly improves the perplexity (23.54 to 11.92). As s goes larger, the percentage of changed Δ increases, and the error reduction rate for salient channels also increases. However, the best perplexity is achieved at $s = 2$, since further increasing s will increase the quantization error for *non-salient* channels.

While the quantization error of salient channels continues to go down as s is increased, this eventually comes at the cost of non-salient channel quantization error (the relative error over non-salient channels increases)

How do we determine the per-input channel scaling factor?


Formally, we want to optimize the following objective:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \mathcal{L}(\mathbf{s})$$
$$\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathrm{diag}(\mathbf{s}))(\mathrm{diag}(\mathbf{s})^{\mathbf{-1}} \cdot \mathbf{X}) - \mathbf{W}\mathbf{X}\| \quad (4)$$

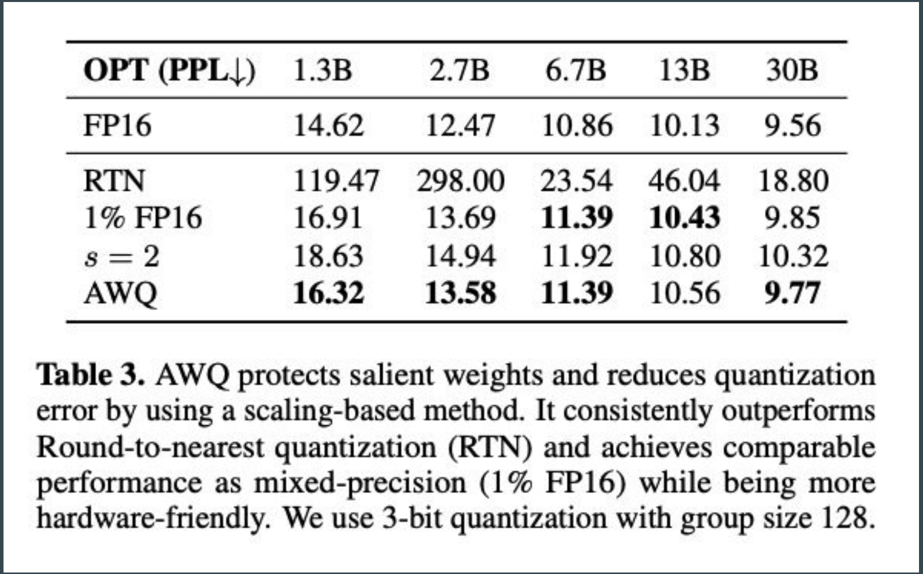
Unfortunately, we cannot optimize this directly because the quantization function (Q) is not differentiable.

Solution: grid search over exponent for magnitude of activation

$$\mathbf{s} = \mathbf{s_X}^{\alpha}, \quad \alpha^* = \arg\min_{\alpha} \mathcal{L}(\mathbf{s_X}^{\alpha})$$

- s_x is the magnitude of activation for a given channel
- α is the hyperparameter (ranging from [0, 1]) we perform grid-search over
    - When α is 0, we get s = 1 (aka no upscaling of weights)
    - When α is 1, we get $s = s_x$ (upscaling is equal to magnitude of activation)
- Intuitively, we want the α that gives us the best trade-off between minimizing the quantization error for salient vs non-salient weights

AWQ performance compared to other quantization strategies



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**

Table 3. AWQ protects salient weights and reduces quantization error by using a scaling-based method. It consistently outperforms Round-to-nearest quantization (RTN) and achieves comparable performance as mixed-precision (1% FP16) while being more hardware-friendly. We use 3-bit quantization with group size 128.

AWQ tends to achieve lower perplexity for OPT than other quantization strategies

Round to Nearest (RTN):

$$Q_{int}(\boldsymbol{w}) = \text{Round}(\frac{\boldsymbol{w}}{\Delta})$$

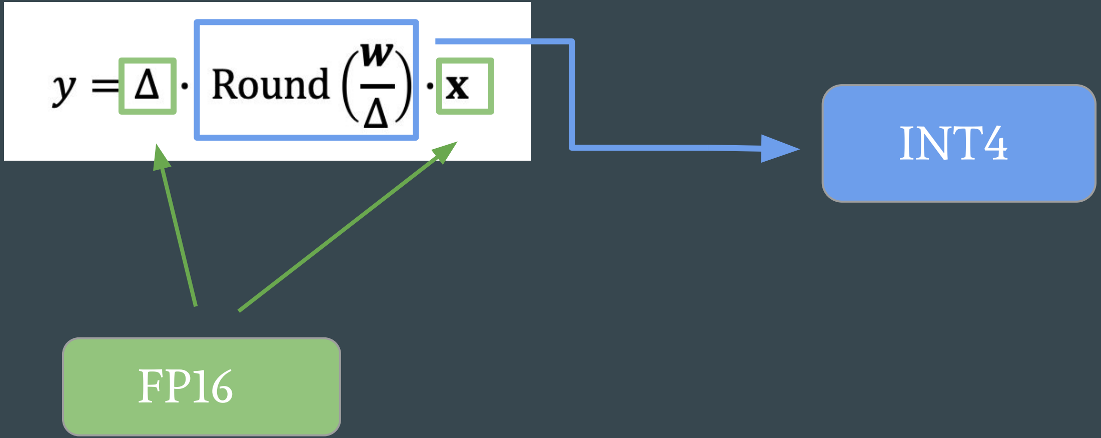
the process we went through before for quantization with no scaling applied

1% FP16:

Keep the top 1% of weights in FP16 while quantizing the rest to INT8.

Dequantization

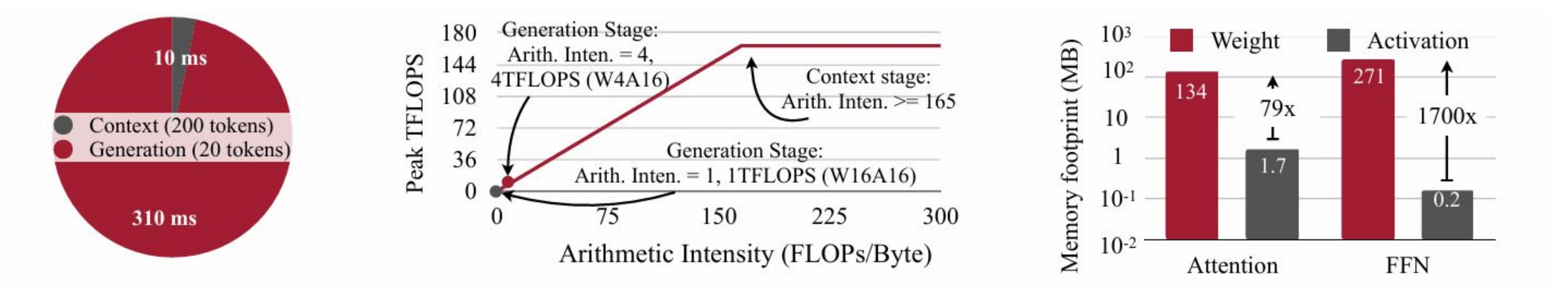
- Not a trivial process, especially since we have mixed data type in W4A16
- If we remember the simplified quantization process from before (without the upscaling):



$y = \Delta \cdot \text{Round}\left(\frac{\mathbf{w}}{\Delta}\right) \cdot \mathbf{x}$
INT4
FP16

How can AWQ accelerate on-device LLMs?

- Profiling LLaMA-7B:
    - Generation is much slower than context-based tasks
        - 310 ms for generation; 10 ms for summarization
    - Generation stage is memory-bound
    - Weight access dominates memory traffic - therefore, more effective to quantize weights instead of activations



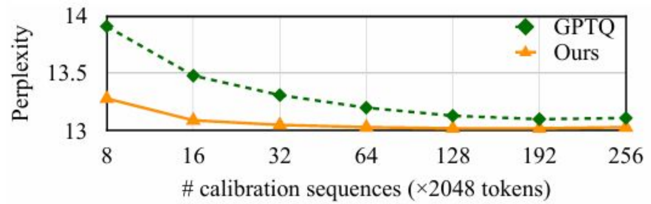
(a) Generation stage is slower (b) Generation stage is bounded by memory bandwidth (c) Weight loading is more expensive

Experiments evaluating AWQ

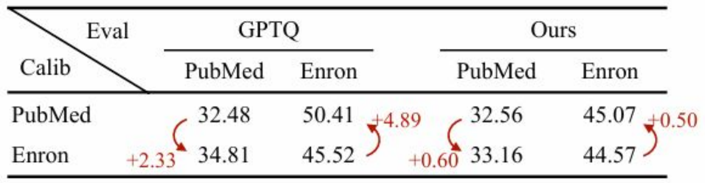
- Models
    - Llama and OPT families
    - OpenFlamingo & LLaVa-13 B for multimodal eval
    - Vicuna for instruction-tuning related tasks
- Datasets
    - Perplexity eval on WikiText-2
    - COCO for multimodal captioning + 11 visual language benchmarks
    - MBPP and GSM8K datasets

Pros of AWQ

- No reliance on regression/backpropagation (since we only need to measure the average activation scale on the calibration set)
- It needs far less data in its calibration set to achieve the same performance compared to GPTQ
    - Only needs 16 sequences vs 192 sequences (10x smaller set)
- It overfits less to the calibration set; when switching calibration sets, perplexity increase is less than for GPTQ



(a) Our method needs a smaller calibration set



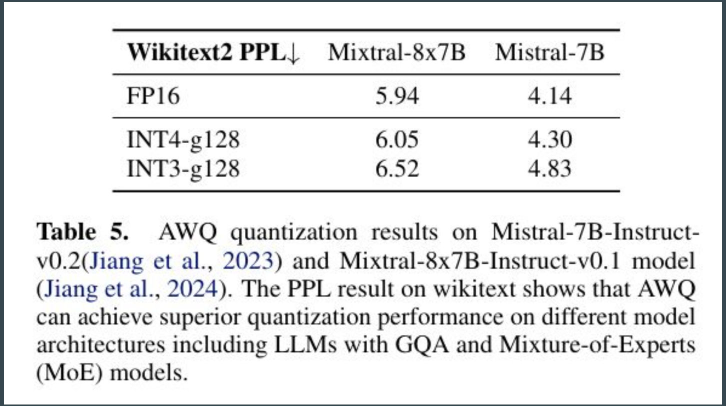
Calib \ Eval	GPTQ PubMed	GPTQ Enron		Ours PubMed	Ours Enron	
PubMed	32.48	50.41	+4.89	32.56	45.07	+0.50
Enron	34.81 (+2.33)	45.52		33.16 (+0.60)	44.57	

(b) Our method is more robust to calibration set distribution

Figure 8. Left: AWQ needs a much smaller calibration set to reach a good quantized performance. It can achieve better perplexity using 10× smaller calibration set compared to GPTQ. **Right:** Our method is more robust to the calibration set distribution. Overall, using the same calibration and evaluation distribution works the best (PubMed-PubMed, Enron-Enron). But when using a different calibration distribution (PubMed-Enron, Enron-PubMed), AWQ only increases the perplexity by 0.5-0.6, while GPTQ has 2.3-4.9 worse perplexity. All experiments are done with the OPT-6.7B model under INT3-g128 quantization.

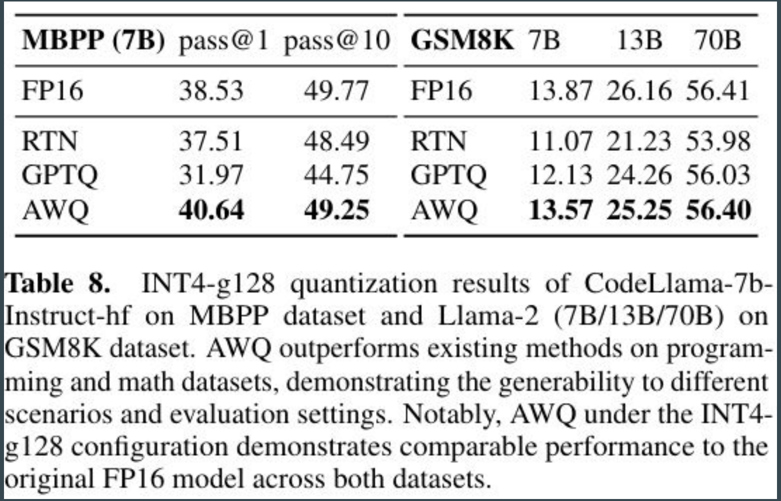
Performance of AWQ under different settings/tasks

- AWQ quantization shows good results for Mistral and Mixtral
- AWQ quantized models perform better than other quantized models on programming and math datasets (MBPP, GSM8K)



Wikitext2 PPL↓	Mixtral-8x7B	Mistral-7B
FP16	5.94	4.14
INT4-g128	6.05	4.30
INT3-g128	6.52	4.83

Table 5. AWQ quantization results on Mistral-7B-Instruct-v0.2(Jiang et al., 2023) and Mixtral-8x7B-Instruct-v0.1 model (Jiang et al., 2024). The PPL result on wikitext shows that AWQ can achieve superior quantization performance on different model architectures including LLMs with GQA and Mixture-of-Experts (MoE) models.



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**

Table 8. INT4-g128 quantization results of CodeLlama-7b-Instruct-hf on MBPP dataset and Llama-2 (7B/13B/70B) on GSM8K dataset. AWQ outperforms existing methods on programming and math datasets, demonstrating the generability to different scenarios and evaluation settings. Notably, AWQ under the INT4-g128 configuration demonstrates comparable performance to the original FP16 model across both datasets.

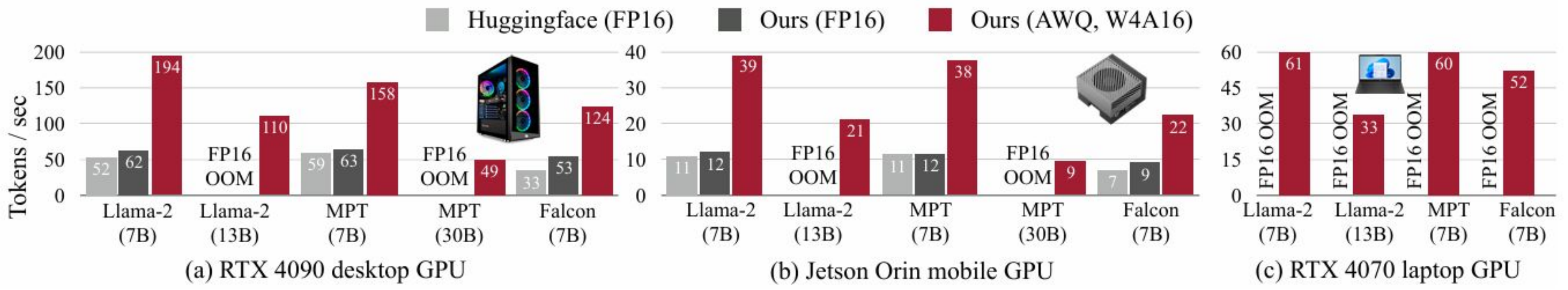
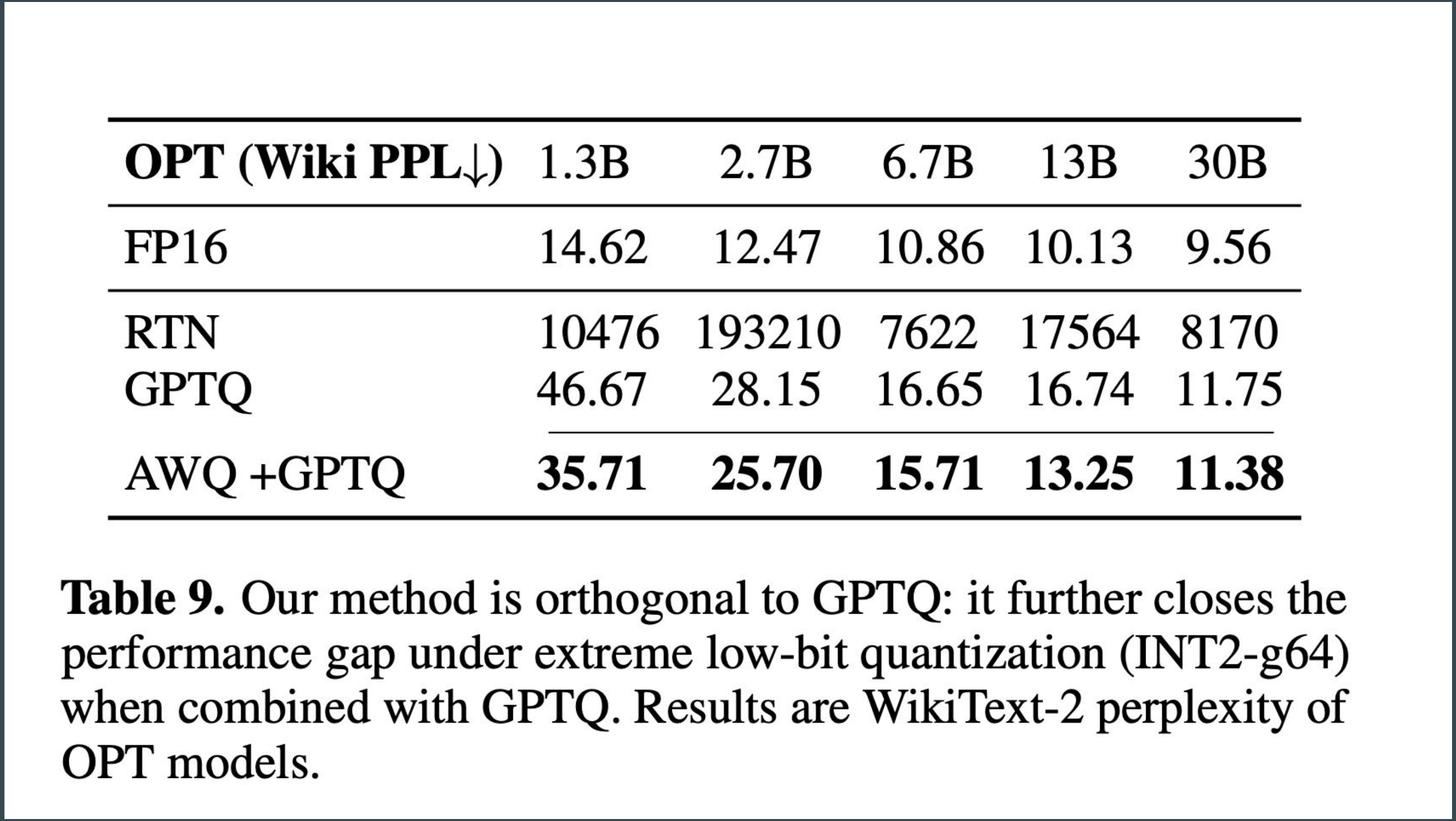


Figure 9. TinyChat provides a turn-key solution to transform the theoretical memory footprint reduction into a quantifiable speedup. As a result, TinyChat is up to **3.9**× and **3.5**× faster than the FP16 implementation from Huggingface on 4090 (desktop GPU) and Orin (mobile GPU), respectively. AWQ also democratizes Llama-2-13B deployment on laptop GPUs (4070) with merely 8GB memory.

Speed-up comparison across models (FP16 vs AWQ, W4A16)



OPT (Wiki PPL↓)	1.3B	2.7B	6.7B	13B	30B
FP16	14.62	12.47	10.86	10.13	9.56
RTN	10476	193210	7622	17564	8170
GPTQ	46.67	28.15	16.65	16.74	11.75
AWQ +GPTQ	**35.71**	**25.70**	**15.71**	**13.25**	**11.38**

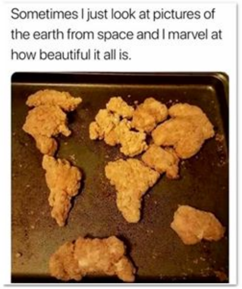
Table 9. Our method is orthogonal to GPTQ: it further closes the performance gap under extreme low-bit quantization (INT2-g64) when combined with GPTQ. Results are WikiText-2 perplexity of OPT models.

AWQ can be combined with GPTQ for better extreme low-bit quantization

Summary of AWQ’s Contributions

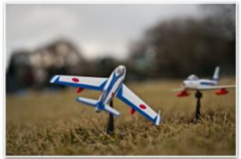
- AWQ allows us to perform low-bit weight specific compression
- Activation magnitude can indicate weight saliency
- Reducing quantization loss over salient weights improves performance
- Performance gains over previous post-training quantization approaches across multiple tasks. Orthogonal to method proposed in GPT-Q paper.
- Theoretical memory savings can improve speedup over FP16 implementations of resource-constrained models
- Across a variety of architectures and tasks, AWQ achieves performance that is nearly on par with FP16

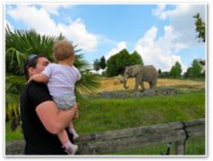
Appendix


Sometimes I just look at pictures of the earth from space and I marvel at how beautiful it all is.

Question: Can you explain this meme in detail?
INT4-g128 RTN: The meme in the image features a plate of food, likely fried chicken, with a unique twist. There are small pictures of the Earth and other planets placed on top of the food, creating an interesting and amusing visual. Alongside the plate, a caption reads, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." This combination of food and planet images adds a touch of humor and curiosity, making the viewer wonder about the context and intent behind the meme.
INT4-g128 AWQ: The meme in the image is a lighthearted and humorous take on the concept of looking at pictures of the Earth from space. A plate of fried food, specifically chicken nuggets, is presented with the caption, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." The playful combination of a food item and an Earth-related context creates a fun and entertaining visual.

Figure 6. Visual reasoning examples from LLaVA-13B model (Liu et al., 2023a). AWQ improves over the round-to-nearest (RTN) baseline, providing more reasonable answers. We color the text to show the correct or wrong responses.


W4-RTN: A model airplane flying in the sky.
W4-AWQ: Two toy airplanes sit on a grass field.


W4-RTN: A man is holding a baby elephant in his arms.
W4-AWQ: A man and his daughter pose with an elephant.

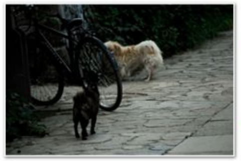
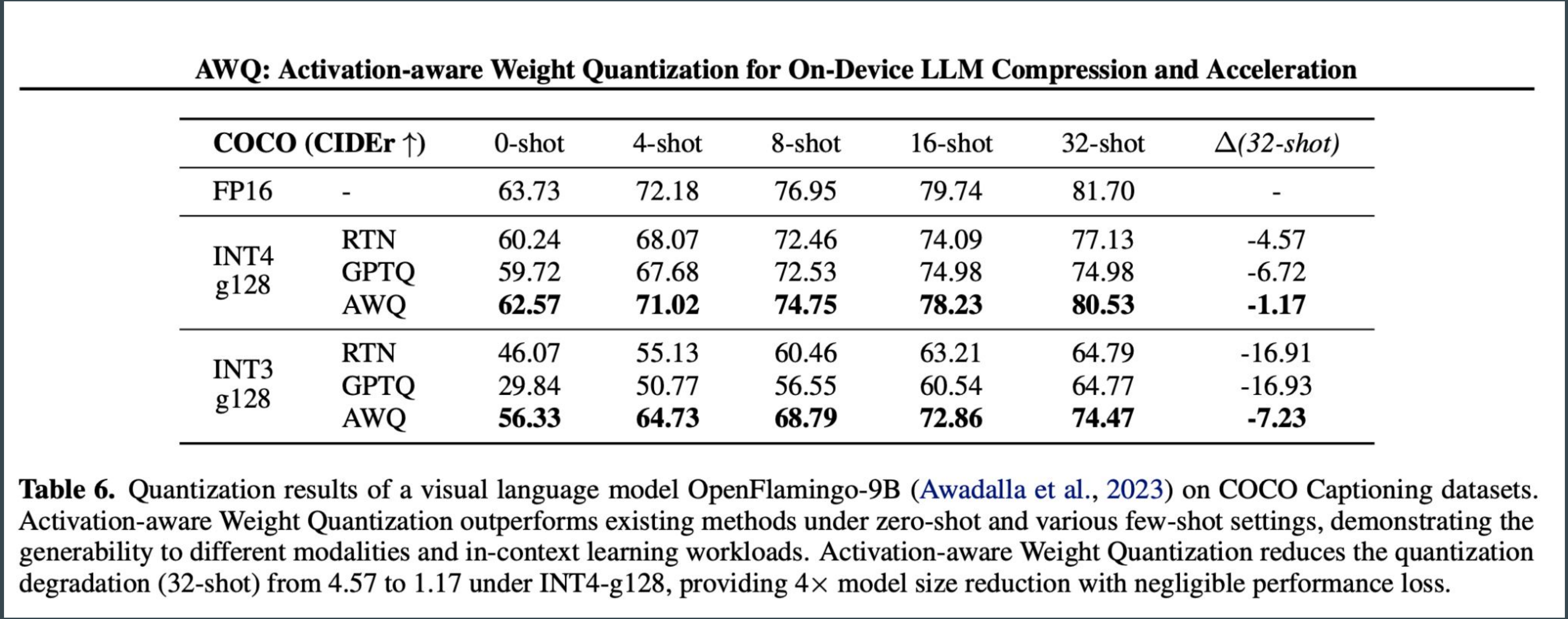

W4-RTN: A man and a dog walking past some bushes.
W4-AWQ: Two dogs are walking on the street.

Figure 7. Qualitative results of quantized OpenFlamingo-9B (Awadalla et al., 2023) on COCO captioning dataset (4-shot, INT4-g128 quantization). Our method significantly improves the captioning quality compared to the round-to-nearest (RTN) baseline. We color the text to show the correct or wrong captions.

Improvement compared to other quantization methods for LLaVA-13B and OpenFlamingo-9B on COCO captioning; RTN produces incorrect captions



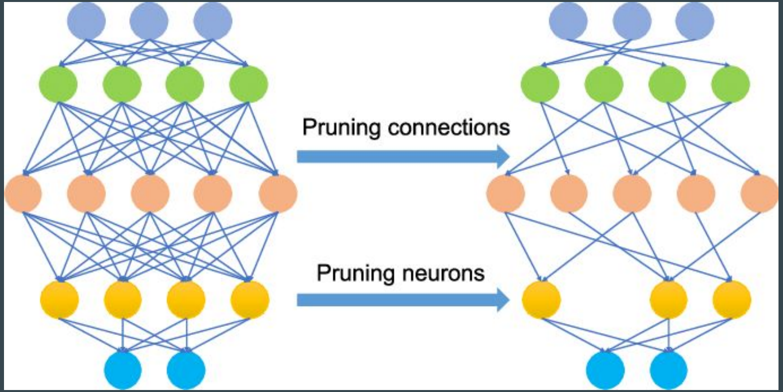
AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration

COCO (CIDEr ↑)		0-shot	4-shot	8-shot	16-shot	32-shot	*Δ(32-shot)*
FP16	-	63.73	72.18	76.95	79.74	81.70	-
INT4 g128	RTN	60.24	68.07	72.46	74.09	77.13	-4.57
	GPTQ	59.72	67.68	72.53	74.98	74.98	-6.72
	AWQ	**62.57**	**71.02**	**74.75**	**78.23**	**80.53**	**-1.17**
INT3 g128	RTN	46.07	55.13	60.46	63.21	64.79	-16.91
	GPTQ	29.84	50.77	56.55	60.54	64.77	-16.93
	AWQ	**56.33**	**64.73**	**68.79**	**72.86**	**74.47**	**-7.23**

Table 6. Quantization results of a visual language model OpenFlamingo-9B (Awadalla et al., 2023) on COCO Captioning datasets. Activation-aware Weight Quantization outperforms existing methods under zero-shot and various few-shot settings, demonstrating the generability to different modalities and in-context learning workloads. Activation-aware Weight Quantization reduces the quantization degradation (32-shot) from 4.57 to 1.17 under INT4-g128, providing 4× model size reduction with negligible performance loss.

Multi-shot performance of OpenFlamingo-9B with different quantization methods

Model Pruning



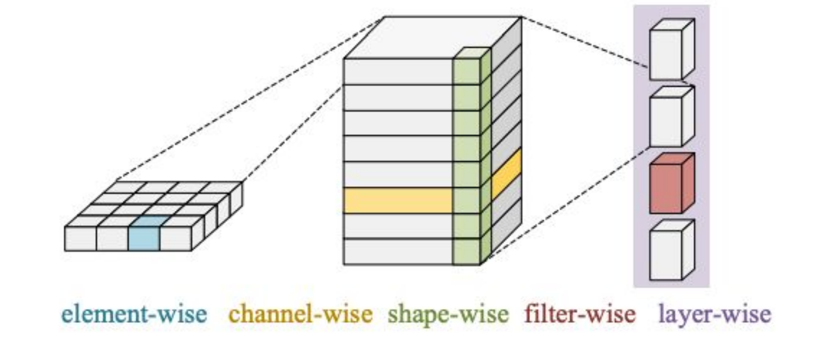
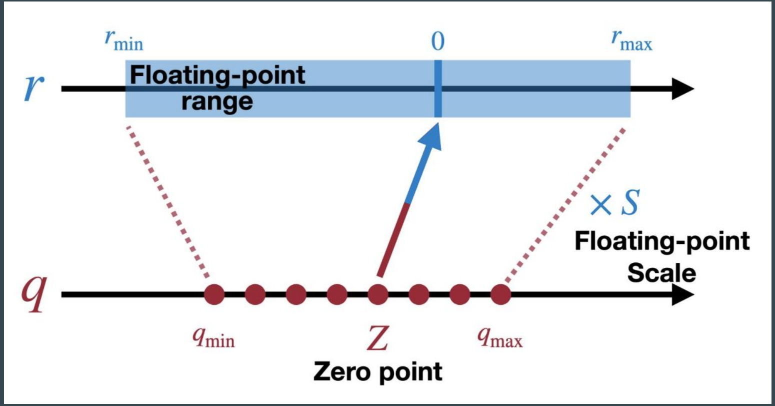


Figure 9: Pruning Opportunities: Different network sparsity results from the granularity of pruned structures. Shape-wise pruning was proposed by Wen [241].

Linearly quantizing a layer



Original layer:

$$y = \mathbf{w}\mathbf{x},$$

Layer after quantization:

$$y = Q(\mathbf{w})\mathbf{x}$$

Quantization function (using absolute max quantization):

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

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

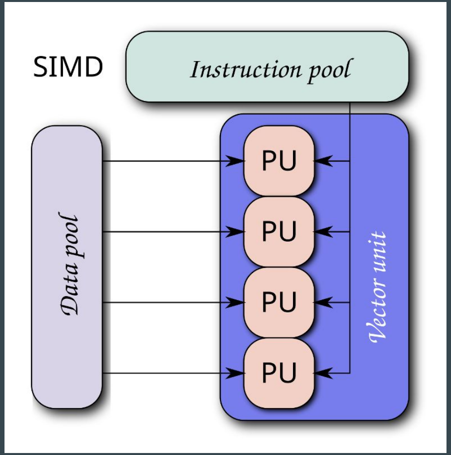
N - 1 because 1 bit is reserved for the sign

Quantized weights

N is the quantization bits, Δ is the quantization scaler

TinyChat

- What is TinyChat?
  - A system for AWQ model inference that handles the many problems with de-quantization while ensuring speed
- Properties of TinyChat:
  - On-the-fly weight de-quantization
  - SIMD-aware weight packing
    - SIMD (single instruction, multiple data): a parallel computing paradigm
    - TinyChat should try to take advantage of SIMD and should thus use platform-specific weight packing
  - Kernel Fusion
    - Attempt to fuse all operators into a single kernel to minimize number of separate operations


SIMD
Instruction pool
Data pool
PU
PU
PU
PU
Vector unit